Esame di Programmazione II, 27 febbraio 2018

Si consideri un'implementazione di un sistema di gestione degli esami universitari. Agli esami è possibile iscrivere studenti e poi verbalizzare l'esito. La verbalizzazione è possibile solo per studenti che siano già iscritti e non è possibile verbalizzare due volte l'esito a uno studente. Per esempio, il seguente codice:

```
1  public class Main {
2    public static void main(String[] args)
3        throws VerbalizzazioneGiaEffettuataException, StudenteNonIscrittoException {
4
5      Esame esame = new EsamePerEsito();
6      Studente s1 = new Studente(151535, "Giorgio", "Levi");
7      Studente s2 = new Studente(200345, "Fausto", "Spoto");
8      Studente s3 = new Studente(324422, "Albert", "Einstein");
9      Studente s4 = new Studente(145066, "Camilla", "De Sanctis");
10     Studente s5 = new Studente(111044, "Eleonora", "Botticelli");
11     esame.iscrivi(s1, s2, s4, s5);
12     esame.verbalizza(s1, Esito.VENTI);
13     esame.verbalizza(s4, Esito.TRENTAELODE);
14     esame.verbalizza(s2, Esito.RITIRATO);
15     esame.verbalizza(s5, Esito.VENTI);
16     System.out.println(esame);
17     esame.verbalizza(s3, Esito.TRENTA);
18   }
19 }
```

deve stampare alla linea 16:

```
200345            Spoto            Fausto: rit
111044        Botticelli          Eleonora: 20
151535             Levi            Giorgio: 20
145066        De Sanctis            Camilla: 30L
```

e terminare alla linea 17 con una `StudenteNonIscrittoException: Lo studente Albert Einstein non e' iscritto all'esame`.

L'enumerazione degli esiti di un esame è già scritta:

```
public enum Esito {
  RITIRATO("rit"), INSUFFICIENTE("ins"), DICIOTTO("18"), DICIANNOVE("19"), VENTI("20"), VENTUNO("21"),
  VENTIDUE("22"), VENTITRE("23"), VENTIQUATTRO("24"), VENTICINQUE("25"), VENTISEI("26"),
  VENTISETTE("27"), VENTOTTO("28"), VENTINOVE("29"), TRENTA("30"), TRENTAELODE("30L");

  private final String name;
  private Esito(String name) { this.name = name; }
  @Override public String toString() { return name; }
}
```

Si ricordi che i suoi elementi sono già `Comparable<Esito>` secondo l'ordine con cui sono enumerati nel codice.

Esercizio 1 [5 punti] Si completi la seguente classe, che implementa uno studente. Uno studente è `equals()` solo a un altro studente con la stessa matricola:

```
public class Studente {
  public final int matricola;
  public final String nome;
  public final String cognome;

  public Studente(int matricola, String nome, String cognome) { ... }
  @Override public boolean equals(Object other) { ... }
  @Override public int hashCode() { ... } // non deve ritornare banalmente una costante
}
```

Esercizio 2 [2 punti] Si scrivano le due eccezioni controllate `VerbalizzazioneGiaEffettuataException` e `StudenteNonIscrittoException` il cui costruttore deve ricevere come argomento uno `Studente` per potere costruire un messaggio di eccezione che includa il suo nome e cognome (si veda l'esempio dell'eccezione generata dal `Main`).

Esercizio 3 [12 punti] Si completi la seguente classe, che implementa un esame a cui gli studenti possono iscriversi e per il quale è possibile verbalizzare un esito dopo l'iscrizione:

```
public abstract class Esame implements Iterable<Esame.Verbalizzazione> {
  private final Set<Studente> iscritti = new HashSet<>();
  private final SortedSet<Verbalizzazione> verbalizzazioni = ...; // si completi questa linea!

  // iscrive all'esame tutti gli studenti indicati
  public final void iscrivi(Studente... studenti) { ... }

  // verbalizza l'esito per uno studente;
  // - lancia una StudenteNonIscrittoException se lo studente non era iscritto all'esame
  // - lancia una VerbalizzazioneGiaEffettuataException se lo studente aveva gia' verbalizzato
  public final void verbalizza(Studente studente, Esito esito)
    throws VerbalizzazioneGiaEffettuataException, StudenteNonIscrittoException { ... }

  // ritorna la concatenazione del toString() delle verbalizzazioni, separate da "\n"
  @Override public final String toString() { ... }

  // ritorna l'iteratore sulle verbalizzazioni effettuate, nell'ordine dato da getComparator()
  @Override public final Iterator<Verbalizzazione> iterator() { ... }

  // ritorna il comparatore da usare per creare le verbalizzazioni: attenzione, e' abstract!
  protected abstract Comparator<Verbalizzazione> getComparator();

  public final static class Verbalizzazione { // classe interna
    private final Studente studente; private final Esito esito;
    private Verbalizzazione(Studente studente, Esito esito) { ... }
    public Studente getStudente() { ... }
    public Esito getEsito() { ... }

    // una Verbalizzazione e' equals() a un'altra che abbia stesso esito e stesso studente
    @Override public boolean equals(Object other) { ... }
    @Override public int hashCode() { ... } // non deve ritornare banalmente una costante
    @Override public String toString() {
      return String.format("%6d %20s %20s: %s", studente.matricola, studente.cognome, studente.nome, esito);
    }
  }
}
```

Si noti che è possibile iterare su un `Esame`, ottenendo una dopo l'altra le verbalizzazioni effettuate, nell'ordine specificato dal comparatore restituito da `getComparator()`. Un comparatore infatti è una interfaccia di libreria che ha un unico metodo che specifica chi viene prima fra due elementi:

```
public interface Comparator<T> {
  // restituisce un numero negativo se o1 viene prima di o2;
  // un numero positivo se o2 viene prima di o1; 0 se o1 e o2 sono uguali
  int compare(T o1, T o2);
}
```

Suggerimento: normalmente la classe di libreria `TreeSet<T>` realizza un insieme ordinato di `T` e tale tipo generico deve implementare `Comparable<T>`. È però possibile definire dei `TreeSet<T>` anche per tipi `T` che non implementino `Comparable<T>`, purché il criterio di confronto venga fornito al momento della costruzione dell'insieme, tramite un `Comparator<T>`, così: `new TreeSet<T>(comparator)`.

Esercizio 4 [9 punti] Si definisca una classe concreta `EsamePerMatricola` che estende `Esame` fissando come ordinamento delle verbalizzazioni quello crescente per matricola. Si definisca una classe concreta `EsamePerEsito` che estende `Esame` fissando come ordinamento delle verbalizzazioni quello crescente per esito e, a parità di esito, quello crescente per matricola.

Esercizio 5 [3 punti] Nella definizione della classe interna `Verbalizzazione` dell'Esercizio 3:

1. È possibile dichiarare tale classe `private` al posto che `public`? Perché ?
2. È possibile eliminare la parola chiave `static`? Cosa cambierebbe?