Esame di Programmazione II, 4 febbraio 2013

Esercizio 1 [**3 punti**] Si definisca una classe `Partito`, che rappresenta un partito politico, con costruttore e metodo pubblici:

- `Partito(String nome)`, che costruisce un partito col nome indicato
- `String getNome()`, che restituisce il nome del partito

Inoltre tale classe deve avere un metodo `equals()` che determina se un partito è uguale a un altro, cioè ha il suo stesso nome; e un metodo `hashCode()` compatibile con `equals()` e non banale.

Esercizio 2 [**7 punti**] Si definisca una classe `Coalizione`, che rappresenta una coalizione di uno o più partiti, con costruttore e metodi pubblici:

- `Coalizione(String nome, Partito... partiti)`, che costruisce una coalizione di cui fanno parte i partiti indicati. Se la sequenza di partiti è vuota, deve generare una `java.lang.IllegalArgumentException`
- `String getNome()`, che restituisce il nome della coalizione
- `boolean rimuovi(Partito partito)`, che elimina il partito indicato dalla coalizione, se c'era, altrimenti non fa nulla. Restituisce true se e solo se la coalizione è diventata vuota.

Inoltre tale classe deve avere un metodo `equals()` che determina se una coalizione è uguale a un'altra, cioè ha il suo stesso nome; e un metodo `hashCode()`, compatibile con `equals()` e non banale.

Esercizio 3 [**1 punti**] Si modifichi la classe `Coalizione` in modo da renderla iterabile (`java.lang.Iterable`) sui partiti che la compongono: iterando su una coalizione si devono quindi ottenere i partiti che la compongono, uno alla volta.

Esercizio 4 [**11 punti**] Si completi la seguente classe, che rappresenta un'elezione politica:

```
public class Elezione {
  /**
   * Le coalizioni registrate per questa elezione. Se un partito si presenta da solo,
   * stara' in una coalizione in cui e' presente solo esso stesso
   */
  private final Set<Coalizione> coalizioni = new HashSet<Coalizione>();
  /**
   * Una mappa che associa a ciascun partito i voti che ha preso fino ad ora
   */
  private final Map<Partito, Integer> votiPerPartito = new HashMap<Partito, Integer>();
  /**
   * Registra il partito indicato come un partecipante all'elezione, dentro una
   * coalizione di cui fa parte solo esso stesso. Se il partito e' gia' registrato,
   * genera una PartitoGiaRegistratoException.
   */
  public void registra(Partito partito) { .... }
  /**
   * Registra la coalizione indicata come partecipante all'elezione. Se esiste gia' una
   * coalizione uguale, deve generare una eccezione di classe CoalizioneGiaPresenteException.
   * Altrimenti i partiti della coalizione vengono eliminati da tutte le altre coalizioni,
   * se ne facevano parte. Tali coalizioni, se in tal modo diventassero vuote, devono venire
   * de-registrate (eliminate) dall'elezione.
   * Tutti i partiti della coalizione registrata sono automaticamente registrati all'elezione
   */
  public void registra(Coalizione coalizione) { .... }
  /**
   * Registra un nuovo voto per il partito indicato (e quindi anche per la coalizione di cui esso fa parte).
   * Se il partito non e' registrato all'elezione, genera una PartitoMaiRegistratoException.
   */
  public void registraVotoPer(Partito partito) { .... }
  @Override public String toString() { .... }
}
```

Scrivete le eccezioni `CoalizioneGiaPresenteException`, `PartitoGiaPresenteException` e `PartitoMaiRegistratoException`, tutte sottoclassi di `java.lang.IllegalArgumentException`.

Il metodo `toString()` deve restituire una stringa che ricapitola i voti a ciascun partito e coalizione, in modo simile a come si evince dall'esempio che trovate sotto.

Se tutto è corretto, l'esecuzione del seguente programma:

```
public class Main {
  private final static Partito[] partiti =
    new Partito[] {
      new Partito("Partito dei belli"), new Partito("Partito dei brutti"), new Partito("Partito mai tornato"),
      new Partito("Partito dei fiori"), new Partito("Partito di tutti"), new Partito("Partito di titti"),
      new Partito("Partito dei nonni")
    };

  public static void main(String[] args) {
    Elezione e = creaElezione(); generaVotiACaso(e); System.out.println(e);
  }

  private static Elezione creaElezione() {
    Elezione e = new Elezione();
    e.registra(partiti[0]); // il partiro dei belli fa inizialmente coalizione da solo
    e.registra(partiti[1]);
    e.registra(new Coalizione("Siamo i piu' forti", partiti[2], partiti[3]));
    // adesso inseriamo il partito dei belli dentro una coalizione a quattro
    e.registra(new Coalizione("Futuro radioso", partiti[0], partiti[4], partiti[5], partiti[6]));
    try {
      // questo genera un'eccezione perche' il "Partito di titti" e' gia' registrato
      // poiche' la sua coalizione "Futuro radioso" e' stata registrata
      e.registra(new Partito("Partito di titti"));
    }
    catch (PartitoGiaRegistratoException exc) { System.out.println(exc); }
    return e;
  }

  private static void generaVotiACaso(Elezione e) {
    Random random = new Random();
    for (int i = 0; i < 89; i++)
      e.registraVotoPer(partiti[Math.abs(random.nextInt()) % 7]);
  }
}
```

deve stampare qualcosa del tipo:

```
it.univr.elezioni.PartitoGiaRegistratoException: Partito di titti
Coalizione "Partito dei brutti":
  Partito dei brutti: voti 15 (16.85%)
Totale voti alla coalizione: 15 (16.85%)

Coalizione "Siamo i piu' forti":
  Partito dei fiori: voti 11 (12.35%)
  Partito mai tornato: voti 9 (10.11%)
Totale voti alla coalizione: 20 (22.47%)

Coalizione "Futuro radioso":
  Partito dei belli: voti 8 (8.98%)
  Partito di titti: voti 17 (19.1%)
  Partito di tutti: voti 15 (16.85%)
  Partito dei nonni: voti 14 (15.73%)
Totale voti alla coalizione: 54 (60.67%)
```

(si noti l'eccezione stampata all'inizio!)