Esame di Programmazione II, 3 febbraio 2017

Si vuole implementare un *doodle*, cioè uno strumento con cui decidere il momento in cui svolgere un evento, massimizzando la presenza dei partecipanti.

Esercizio 1 [8 punti] Uno slot temporale specifica un momento temporale in cui si potrebbe svolgere l'evento. Si tratta di un oggetto comparabile, che deve essere inseribile in una collezione ordinata ma anche in un insieme o mappa hash. Si completi la sua seguente implementazione:

```
public class Slot implements Comparable<Slot> {
  ...
  public Slot(int year, int month, int day, Moment moment) { ... }
  public int compareTo(Slot other) { ... }  // ordina per anno, mese, giorno e infine momento
  public boolean equals(Object other) { ... }  // confronta tutti e quattro le componenti degli Slot
  public int hashCode() { ... }
  public String toString() { ... }  // ritorna una stringa del tipo "3/2/2017 AFTERNOON"

  public enum Moment {
    MORNING,
    AFTERNOON,
    EVENING,
    NIGHT
  }
}
```

Si noti l'enumerazione interna `Slot.Moment`. Come tutte le enumerazioni in Java, essa è implicitamente comparabile e quindi ha un metodo `compareTo` aggiunto automaticamente dal compilatore. Se si stampa un elemento di una enumerazione, si ottiene la stringa del nome dell'elemento. Infine, gli elementi di una enumerazione in Java hanno anche i metodi `equals` e `hashCode`, scritti automaticamente dal compilatore e consistenti fra di loro e con `compareTo`.

Esercizio 2 [14 punti] La classe che implementa il doodle fornisce metodi per la specifica della disponibilità temporale di ciascun partecipante. Inoltre ha un metodo `toString` che ritorna una stringa che descrive il doodle. Se ne completi la seguente implementazione:

```
public class Doodle {
  // una mappa che al nome di ciascun partecipante associa l'insieme (non ordinato)
  // degli slot temporali in cui e' disponibile
  private final Map<String, Set<Slot>> availabilities = new HashMap<>();

  // prendono nota che un partecipante e' disponibile in certi slot
  public void available(String name, Slot... when) { ... }
  public void available(String name, Iterable<Slot> when) { ... }

  // restituisce una stringa che descrive il doodle, come nell'esempio nella pagina seguente; si noti
  // che gli slot sono presentati in ordine crescente e i nomi dei partecipanti in ordine alfabetico
  public String toString() {
    String result = "";

    // mette in slots tutti gli slot temporali di tutti i partecipanti, ordinati in senso crescente
    SortedSet<Slot> slots = new TreeSet<>();
    for (Set<Slot> set: availabilities.values())
      slots.addAll(set);

    ...
    return result;
  }

  // restituisce il peso (importanza) della presenza del partecipante indicato;
  // in questa implementazione, tutti i partecipanti hanno lo stesso peso (1.0)
  protected double weightOf(String name) {
    return 1.0;
  }
}
```

Esercizio 3 [10 punti, solo per chi non ha già fatto il progetto degli anni passati] Si definisca una sottoclasse `WeightedDoodle` di `Doodle`, in cui i partecipanti possono avere pesi diversi da 1.0. Se ne completi a tal fine la seguente implementazione:

```
public class WeightedDoodle extends Doodle {
  private final Map<String, Double> weights = new HashMap<>();  // una mappa da nome a peso

  @Override public void available(String name, Slot... when) { ... } // peso per default 1.0
  @Override public void available(String name, Iterable<Slot> when) { ... } // peso per default 1.0
  public void available(String name, double weight, Slot... when) { ... } // peso indicato
  public void available(String name, double weight, Iterable<Slot> when) { ... } // peso indicato
  @Override protected double weightOf(String name) { ... }
}
```

Suggerimento: Le mappe della libreria standard hanno il metodo `put(chiave, valore)` che lega la chiave al valore indicato; hanno il metodo `get(chiave)` che restituisce il valore legato alla chiave indicata (se esiste, altrimenti ritorna `null`); hanno il metodo `keySet` che restituisce l'insieme delle chiavi memorizzate nella mappa e il metodo `values` che restituisce l'insieme dei valori legati nella mappa a qualche chiave. Le collezioni della libreria standard hanno il metodo `contains(value)` che determina se la collezione contiene o no tale insieme. Tutti questi metodi funzionano modulo `equals`.

Se tutto è corretto, l'esecuzione del programma:

```
public class Main {
  public static void main(String[] args) {
    Slot s1 = new Slot(2017, 2, 4, Slot.Moment.MORNING);
    Slot s2 = new Slot(2017, 2, 4, Slot.Moment.AFTERNOON);
    Slot s3 = new Slot(2017, 2, 5, Slot.Moment.AFTERNOON);
    Slot s4 = new Slot(2017, 2, 5, Slot.Moment.EVENING);
    Slot s5 = new Slot(2017, 2, 4, Slot.Moment.AFTERNOON); // like s2

    Doodle doodle1 = new Doodle();
    doodle1.available("Fausto", s2, s4);
    doodle1.available("Giovanni", s1, s3, s4, s5);
    doodle1.available("Maria", s1, s2, s3, s5);
    doodle1.available("Alessandra", s3);
    System.out.println("doodle1:\n" + doodle1);

    WeightedDoodle doodle2 = new WeightedDoodle();
    doodle2.available("Fausto", 0.8, s2, s4);   // la presenza di Fausto e' abbastanza importante (0.8)
    doodle2.available("Giovanni", 0.5, s1, s3, s4, s5);   // la presenza di Giovanni e' mediamente importante (0.5)
    doodle2.available("Maria", 0.2, s1, s2, s3, s5);   // la presenza di Maria e' quasi irrilevante (0.2)
    doodle2.available("Alessandra", 1.0, s3);   // la presenza di Alessandra e' molto importante (1.0)
    System.out.println("doodle2:\n" + doodle2);
  }
}
```

dovrà stampare:

```
doodle1:
4/2/2017 MORNING        4/2/2017 AFTERNOON        5/2/2017 AFTERNOON        5/2/2017 EVENING
        no                      no                      yes                     no              Alessandra
        no                      yes                     no                      yes             Fausto
        yes                     yes                     yes                     yes             Giovanni
        yes                     yes                     yes                     no              Maria
        2.0                     3.0*                    3.0                     2.0

doodle2:
4/2/2017 MORNING        4/2/2017 AFTERNOON        5/2/2017 AFTERNOON        5/2/2017 EVENING
        no                      no                      yes                     no              Alessandra
        no                      yes                     no                      yes             Fausto
        yes                     yes                     yes                     yes             Giovanni
        yes                     yes                     yes                     no              Maria
        0.7                     1.5                     1.7*                    1.3
```

Si noti che in queste stampe gli slot temporali sono ordinati in modo crescente e i nomi dei partecipanti in modo alfabetico. Inoltre il numero sotto ogni slot indica la somma pesata delle disponibilità per quello slot. Nella classe `Doodle`, le disponibilità valgono sempre 1.0, per ogni partecipante, mentre nella classe `WeightedDoodle` il peso dipende dal partecipante, come specificato al momento della chiamata ai vari metodi `available`. Il primo slot temporale in cui si massimizza la somma pesata delle disponibilità dei partecipanti è indicato con un asterisco alla destra della somma pesata. Quello sarebbe lo slot temporale prescelto per l'evento.

È possibile definire campi, metodi, costruttori e classi aggiuntive, ma solo `private`.