Esame di Programmazione II, 30 settembre 2015

Un sistema per visualizzare e modificare un orario è organizzato secondo lo schema model/view/controller. Il modello contiene le informazioni sull'orario e può essere legato a una o più view, che ne mostrano il contenuto:

```
public interface Model {
  void linkToView(View view); // connette il modello a una view
  void set(int hours, int minutes, int seconds) throws IllegalArgumentException; // modifica ora, minuti e secondi
  int getHours();
  int getMinutes();
  int getSeconds();
}
```

Esercizio 1 [**4 punti**] Si completi la seguente implementazione del modello, senza dimenticare di lanciare l'eccezione se l'orario è illegale (le ore devono essere valori da 0 a 23 inclusi; i minuti e secondi da 0 a 59 inclusi):

```
public class Time implements Model {
  ...
  private final Set<View> views = new HashSet<>(); // tutte le view a cui e' connesso

  @Override
  public void set(int hours, int minutes, int seconds) throws IllegalArgumentException {
    ...
    for (View view: views)  // notifica tutte le view del nuovo orario
      view.onTimeChanged(hours, minutes, seconds);
  }
  ...
  @Override
  public void linkToView(View view) {
    views.add(view);
    view.onTimeChanged(hours, minutes, seconds);
  }
}
```

Esercizio 2 [**4 punti**] Una view visualizza l'orario ed è notificata di ogni suo cambiamento:

```
public interface View {
  void onTimeChanged(int hours, int minutes, int seconds);
}
```

Se ne realizzi l'implementazione `TextDateView` che, ad ogni cambiamento dell'orario, stampa sul video un messaggio del tipo `15:05:09` oppure `1:05:19`. Si osservi, in questi due esempi, dove vanno messi gli `0` e dove invece non vanno messi e si scriva il codice di conseguenza.

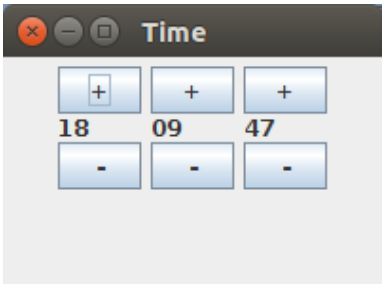
Esercizio 3 [**5 punti**] Un controllore riceve ordini di modifica dell'orario e li implementa sul modello:

```
public interface Controller {
  void onIncreaseHours(); // e' stato richiesto l'incremento (circolare) dell'ora
  void onDecreaseHours(); // e' stato richiesto il decremento (circolare) dell'ora
  void onIncreaseMinutes(); // e' stato richiesto l'incremento (circolare) dei minuti
  void onDecreaseMinutes(); // e' stato richiesto il decremento (circolare) dei minuti
  void onIncreaseSeconds(); // e' stato richiesto l'incremento (circolare) dei secondi
  void onDecreaseSeconds(); // e' stato richiesto il decremento (circolare) dei secondi
  void onResetTime(); // e' stato richiesto di resettare l'orario al momento corrente
}
```

Se ne completi l'implementazione seguente:

```
public class ControllerImpl implements Controller {
  private final Model model; // dove effettuare le modifiche

  public ControllerImpl(Model model) {
    this.model = model;
    onResetTime(); // alla creazione del controller, il modello viene settato al tempo corrente
  }
  ...
  @Override
  public void onResetTime() {
    Calendar calendar = Calendar.getInstance();
    model.set(calendar.get(Calendar.HOUR_OF_DAY), calendar.get(Calendar.MINUTE), calendar.get(Calendar.SECOND));
  }
}
```

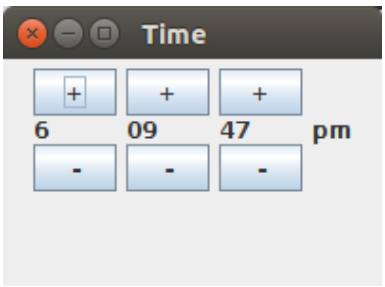
Esercizio 4[9 punti] Una versione grafica della view può essere realizzata tramite la libreria Swing, includendo anche dei bottoni per incrementare e decrementare le tre componenti dell'orario, come nel disegno seguente:



Per esempio si può usare il seguente codice:

```
public class SwingDateView extends JFrame implements View {
  private final JLabel hours = new JLabel(), minutes = new JLabel(), seconds = new JLabel();

  public SwingDateView(Controller controller) {
    super("Time"); setMinimumSize(new Dimension(200, 120));
    add(buildClockPanel(controller), BorderLayout.CENTER);  // aggiunge le componenti grafiche necessarie
    pack(); setVisible(true);
  }

  protected JPanel buildClockPanel(Controller controller) {
    JPanel clock = new JPanel();

    // crea la colonna delle ore
    JPanel hoursPanel = new JPanel();
    hoursPanel.setLayout(new BorderLayout());
    JButton increaseHours = new JButton("+");
    hoursPanel.add(increaseHours, BorderLayout.NORTH);
    hoursPanel.add(hours, BorderLayout.CENTER);
    JButton decreaseHours = new JButton("-");
    hoursPanel.add(decreaseHours, BorderLayout.SOUTH);
    clock.add(hoursPanel);
    ... codice simile per la colonna dei minuti e dei secondi: NON SCRIVETELO, TANTO E' IDENTICO

    // settaggio dei listeners di attivazione dei bottoni
    increaseHours.addActionListener(e -> controller.onIncreaseHours());
    decreaseHours.addActionListener(e -> controller.onDecreaseHours());
    increaseMinutes.addActionListener(e -> controller.onIncreaseMinutes());
    decreaseMinutes.addActionListener(e -> controller.onDecreaseMinutes());
    increaseSeconds.addActionListener(e -> controller.onIncreaseSeconds());
    decreaseSeconds.addActionListener(e -> controller.onDecreaseSeconds());

    return clock;
  }

  @Override
  public void onTimeChanged(int hours, int minutes, int seconds) {
    this.hours.setText(String.valueOf(hours));
    this.minutes.setText(String.format("%02d", minutes));
    this.seconds.setText(String.format("%02d", seconds));
  }
}
```

Si scriva una estensione `Swing12DateView` di `SwingDateView`, che visualizza l'orario nel formato a 12 ore inglese, in cui *am* indica gli orari precedenti al mezzogiorno e *pm* indica gli orari dal mezzogiorno in poi. Si ricordi che, in tale formato, il 12 sostituisce lo 0, per cui la mezzanotte è indicata come 12am e il mezzogiorno come 12pm. L'estensione dovrà essere una componente grafica del tipo:



Ovviamente l'indicazione am/pm dovrà variare sulla base degli aggiornamenti di orario che arrivano alla componente.