

Fondamenti di informatica LB

Esame del 22 Giugno 2010

NOTE:

- Scrivere su ogni foglio utilizzato cognome, nome, numero di matricola (tutto in forma leggibile).
- Consegnare tutti i fogli ricevuti (anche il testo).
- Durante la prova non è consentita la consultazione di alcun tipo di materiale, né è consentito uscire.
- La prova è ritenuta sufficiente se almeno una parte significativa di ogni esercizio proposto è svolta correttamente. Il voto finale sarà espresso in ventisettesimi.
- Tempo a disposizione per l'intero compito: 2h 30'.

Esercizio 1 (12/40)

Si progetti la parte principale di un'applicazione che realizza un semplice simulatore di eventi. L'evento generico è definito dalla classe **AbstractEvent** così strutturata:

- Campo **int when** che rappresenta l'istante di tempo in cui l'evento avviene.
- Metodo pubblico concreto **int getTime()** che restituisce il valore di **when**.
- Metodo pubblico astratto **void happen()** che sarà implementato dalle sottoclassi concrete.

La classe **AbstractEvent** è estesa dalla classe **SimpleEvent** così organizzata:

- Campi **String info** e **int duration** che rappresentano, rispettivamente, le informazioni relative all'evento e la sua durata.
- Costruttore con tre argomenti, rispettivamente per inizializzare i campi **when**, **info** e **duration**. Qualora si provi a creare un evento con durata minore di 1, deve essere generata un'eccezione.
- Metodo pubblico **int getDuration()** per recuperare la durata dell'evento.
- Si implementi il metodo **happen()** mostrando a video un messaggio di testo con le informazioni sull'evento.

Il simulatore è realizzato dalla classe `Simulator` così strutturata:

- Campo `int time` che rappresenta l'istante di tempo corrente.
- `ArrayList<SimpleEvent> eventList`: campo che contiene la lista degli eventi.
- Costruttore di default che inizializza `eventList` e `time` (a zero).
- `SimpleEvent findNextEvent()`: metodo privato che restituisce il primo evento in lista che abbia il campo `when` uguale al valore di `time` del simulatore. Nel caso in cui un evento con queste caratteristiche non esista, si restituisca `null`.
- `void addEvent(SimpleEvent e)`: metodo pubblico che permette di inserire un evento in lista.
- `void simulate(int maxTime)`: metodo pubblico che effettua la simulazione fino al tempo `maxTime`. L'algoritmo da eseguire è il seguente:

```
Fintanto che time <= maxTime :  
    Stampa a video il valore di time  
    Sia ev il prossimo evento  
    Se (ev != null) allora :  
        Rimuovi ev dalla lista  
        Fai avvenire ev  
        Incrementa time di un valore pari alla durata di ev  
    altrimenti: Incrementa time di una unità
```

NOTA: definire opportunamente campi e metodi come *private*, *protected* o *public*.

Si implementino le classi così come descritte e si implementi anche un programma che mostri come utilizzare le classi e i metodi implementati. Si riporta uno stralcio della documentazione di interesse:

```
java.util  
Class ArrayList<E>  
boolean add(E o): Appends the specified element to the end of this list.
```

Esercizio 2 (10/40)

Si analizzi il codice seguente e si riporti l'output prodotto dall'esecuzione del programma. Le risposte devono essere opportunamente commentate e motivate.

```
class Caos {
    private Gea a;
    private Urano d;

    public Caos(Gea a) {
        this.a = a;
        d = new Urano();
        System.out.println("Caos: "
                           + a.toString() + "," + d.toString());}
}

class Gea {
    public Gea(){System.out.println("Gea");}
    public String toString(){
        return "Oggetto di classe Gea";
    }
}

class Teti extends Gea {
    public String toString(){return "Teti";}
}

class Urano {
    public Urano(){System.out.println("Urano");}
    public String toString(){
        return "Urano";
    }
}

public class Esercizio2 {
    public static void main(String [] args){
        Teti t = new Teti();
        Caos c = new Caos(t);
    }
}
```

Esercizio 3 (8/40)

Si completi l'implementazione sotto riportata del pannello di una semplice applicazione la cui interfaccia grafica è costituita da un campo di testo non modificabile `txt`, due `JCheckBox` `ck1` e `ck2` e un bottone. Alla pressione del bottone, nel campo di testo deve essere riportata una scritta, a seconda dello stato di selezione delle caselle:

- Una sola è selezionata: si riporti l'etichetta associata a tale casella.
- Entrambe selezionate: si riporti la scritta 'Green'.
- Nessuna è selezionata: si scriva sul campo di testo una stringa vuota.

```
public class MyPanel extends JPanel implements ActionListener {
    JTextField txt;  JCheckBox ck1, ck2;  JButton b;

    public MyPanel() {
        super();
        txt = new JTextField(22);
        txt.setEditable(false);
        ck1 = new JCheckBox("Blue"); ck2 = new JCheckBox("Yellow");
        b = new JButton("EXEC");
        ...[1]...
    }

    public void actionPerformed(ActionEvent e) {
        ...[2]...
    }
}
```

Documentazione di interesse:

```
class JCheckBox
public String getText(): Returns the button's text
public boolean isSelected(): Returns the state of the button.
                           True if the toggle button is selected,
                           false if it's not.
```

Esercizio 4 (10/40)

Si introduca il *subclassing* e se ne illustrino le proprietà principali, fornendone anche un esempio.