

Fondamenti di informatica LB (6 CFU)

Esame del 21 Febbraio 2011

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.
- Il tempo a disposizione per il compito è 2h 30'. Coloro i quali consegnano perdono il voto eventualmente conseguito in precedenza.

Esercizio 1 (12/40)

Si vuole progettare una parte di un sistema software per il controllo di un sistema domotico in cui sia possibile accendere e spegnere vari dispositivi. I dispositivi devono implementare l'interfaccia `IDevice`, caratterizzata dai metodi `void on()` e `void off()`, rispettivamente per accendere e spegnere il dispositivo e dal metodo `boolean isOn()` che restituisce `true` o `false` a seconda dello stato del dispositivo. L'interfaccia ha, inoltre, un metodo `double getPower()` che restituisce la potenza (in Watt) del dispositivo.

L'interfaccia `IDevice` è implementata, in particolare, dalla classe `Light`. Il costruttore di questa classe accetta un argomento `double` che permette di impostare il consumo di potenza del dispositivo, mantenuto in un campo della classe (`double power`). Se il valore introdotto è negativo, allora il costruttore deve lanciare un'eccezione. La classe ha, inoltre, un campo booleano `stateOn` per mantenere lo stato di accensione del dispositivo, impostato a `false` nella costruzione.

Il sistema di gestione e controllo è costituito da un oggetto della classe `DSystem`, caratterizzata, in particolare, dai seguenti campi e metodi:

- `ArrayList<IDevice> deviceList`: campo privato che mantiene la lista dei dispositivi.
- `double maxPower`: campo privato che rappresenta il massimo consumo di potenza del sistema, impostato all'atto della costruzione.
- `boolean on(int i)`: metodo pubblico che permette di accendere il dispositivo in posizione *i* della lista. Il dispositivo deve essere acceso solo se sono verificate contemporaneamente le seguenti condizioni: (i) l'indice *i* sia valido, (ii) il dispositivo sia

spento e (iii) vi sia potenza sufficiente; in tal caso, viene restituito `true`, altrimenti `false`.

- `void addDevice(IDevice d)`: metodo pubblico che permette di aggiungere un dispositivo al sistema.

Si implementino le classi così come descritte (definendo opportunamente campi e metodi come *private*, *protected* o *public*) e i costruttori, anche ove non esplicitamente specificato. È anche possibile implementare metodi non menzionati nel testo, se lo si ritiene utile. 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.
public E get(int index): Returns the element at the specified position
                        in this list.
                        Throws IndexOutOfBoundsException - if the index
                        is out of range (index < 0 || index >= size()).
```

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 Haendel{
    public String toString(){
        return "I am Haendel";
    }
    public void play(){};
}

class Telemann extends Haendel{
    public Telemann(){
        super();
        System.out.println(super.toString() + "");
        System.out.println("Telemann");
    }
    public String toString(){
        return "I am Telemann";
    }
    public void play(){
        System.out.println(this.toString() + " - " + super.toString());
    }
}

class Vivaldi extends Haendel{
    public Vivaldi(){
        super();
        System.out.println("Vivaldi");
    }
    public String toString(){
        return "I am Vivaldi";
    }
}

public class Esercizio2{
    public static void main(String [] args){
        int i;
        Haendel [] h = new Haendel[2];
        h[0] = new Telemann(); h[1] = new Vivaldi();
        for (i = 0; i < 2; ++i) { h[i].play(); }
    }
}
```

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, un insieme di checkbox e un pulsante. Si devono gestire due tipi di evento: quelli generati dalle checkbox e quelli generati dal pulsante. Quando una casella viene **selezionata**, nel campo di testo deve comparirne l'etichetta corrispondente. Quando il pulsante è premuto, tutte le caselle devono essere deselectionate.

```
class MyPanel extends JPanel implements ActionListener {
    private JTextField text;
    private JCheckBox ck[];
    private JButton unsel;
    public MyPanel(){
        super();
        text = new JTextField(10);
        text.setEditable(false);
        ck = new JCheckBox[3];
        ck[0] = new JCheckBox("Telemann");  ck[1] = new JCheckBox("Vivaldi");
        ck[2] = new JCheckBox("Haendel");
        desel = new JButton("Deselect");
        ...[1]...
    }

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

Documentazione di interesse:

```
public Object getSource():
    The object on which the Event initially occurred.

public boolean isSelected():
    Returns the state of the button. True if the toggle button is selected,
    false if it's not.

public String getText():
    Returns the button's text.
```

Esercizio 4 (10/40)

Descrivere sinteticamente i principi della programmazione ad eventi.