

Fondamenti di informatica B

Esame del 13 Settembre 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.
- 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 realizzare un programma per gestire un negozio per vendite *online* di computer la cui configurazione può essere scelta dal cliente. Il progetto deve essere strutturato come segue:

- Vi è una classe **Computer** che rappresenta il calcolatore. Gli oggetti **Computer** sono composti dall'aggregazione degli oggetti **Processor**, **Memory** e una lista di componenti opzionali, per esempio , **HD** e **DVDWriter**.
- Le classi **Processor**, **Memory** e i componenti opzionali estendono la classe astratta **AComponent** che a sua volta implementa l'interfaccia **IComponent**.
- Vi è la classe **Store** che mantiene l'archivio dei componenti disponibili.

La classe **Computer** ha i seguenti campi e metodi:

- Campi privati per rappresentare processore e memoria.
- Campo privato `ArrayList<IComponent> optionals` che rappresenta la lista dei componenti opzionali.
- Metodo `toString()` ridefinito in modo da restituire la composizione del computer. Questo metodo utilizzerà i metodi `getModel()` che saranno ridefiniti nei componenti.
- `double getPrice()`: metodo pubblico che restituisce il prezzo del computer, come somma dei prezzi dei singoli componenti.

- Costruttore con parametri formali `Processor p`, `Memory m` e `ArrayList<IComponent>`.

L'interfaccia `IComponent` ha i seguenti metodi:

- `String getModel()`: restituisce il modello del componente.
- `double getPrice()`: restituisce il prezzo del componente.

La classe astratta `AComponent` ha i seguenti campi (protetti) e metodi:

- `String model`, `double price`.
- Costruttore a due parametri per inizializzare i campi.

Inoltre, implementa l'interfaccia `IComponent`.

Le classi `Processor`, `Memory` e i componenti opzionali hanno tutte la stessa struttura: estendono `AComponent` e hanno un costruttore con due parametri. Ai fini dello svolgimento dell'esercizio, è sufficiente mostrare l'implementazione di una sola di queste classi.

La classe `Store` aggrega i componenti come `ArrayList` di `IComponent`.

La classe `Store` ha i seguenti metodi pubblici:

- Costruttore di default.
- `IComponent isAvailable(String model)`: restituisce il riferimento al componente di modello `model` se è presente nel magazzino, altrimenti restituisce `null`.
- `void addComponent(IComponent c)`: aggiunge un componente al magazzino.
- `Computer build(String procModel, String memoryModel, ArrayList<String> optionals)`: permette di creare un computer con i componenti indicati. Per ciascun componente, si dovrà effettuare la verifica di disponibilità (basandosi sul modello); se tutti i componenti richiesti sono presenti, si può procedere alla creazione del computer, eliminando dal magazzino i componenti utilizzati. Se invece non tutti i componenti sono presenti nel magazzino, il metodo deve lanciare un'eccezione.
- `boolean buy(Computer c, double money)`: permette di acquistare il computer `c`; se `money` è maggiore o uguale del prezzo del computer allora il metodo restituisce `true`, altrimenti restituisce `false`.

Si implementino le classi così come descritte e si implementi anche un programma di test (supponendo, per esempio, di disporre anche dei componenti `HD` e `DVDWriter`). Si riporta stralcio della documentazione di interesse:

```
boolean add(E e): Appends the specified element to the end of this list.
```

```
boolean remove(Object o): Removes the first occurrence of the
                           specified element from this list,
                           if it is present.
```

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.

```
abstract class Yanomami {
    public Yanomami() { System.out.println("Yanomami"); }
    public void talk(Yanomami a) { System.out.println("White Horse"); }
}

class Navajo extends Yanomami {
    public Navajo(){
        super();
        System.out.println("Navajo");
    }
    public void talk(Navajo a) { System.out.println("Red Cloud"); }
}

class Lakota extends Navajo {
    public Lakota(){
        super();
        System.out.println("Lakota");
    }
    public void talk(Yanomami a) { System.out.println("Wicked Bear"); }
}

public class Esercizio2 {
    public static void main(String [] args){
        Lakota lak = new Lakota();
        Navajo nav = new Navajo();
        nav.talk((Navajo) lak);
        lak.talk((Yanomami) nav);
        nav.talk(lak);
    }
}
```

Esercizio 3 (8/40)

Si completi l'implementazione sotto riportata del pannello di una semplice applicazione la cui interfaccia grafica è costituita da tre campi di testo e un pulsante JButton. Si devono gestire due eventi diversi:

- Quando il cursore è sul campo `txt1` oppure sul campo `txt2`, alla pressione del tasto ENTER (della tastiera) si deve scrivere nel campo `dest` la concatenazione del contenuto di `txt1` e `txt2`.
- Quando si preme sul pulsante, lo stato modificabile (risp. non modificabile) del campo di testo `txt2` deve essere cambiato.

```
class MyPanel extends JPanel implements ActionListener{
    private JTextField txt1,txt2,dest;
    private ButtonGroup bg;
    private JRadioButton [] rb;
    public MyPanel() {
        super();
        txt1 = new JTextField(30);
        txt2 = new JTextField(30);
        dest = new JTextField(30);
        dest.setEditable(false);
        JButton b = new JButton("Active/Not active");
        ...[1]...
    }

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

Documentazione di interesse:

```
public Object getSource():
```

The object on which the Event initially occurred.

```
public boolean isEditable():
```

Returns the boolean indicating whether this TextComponent is editable or not.

Esercizio 4 (10/40)

a) Analizzare il seguente programma e indicare quali sono i valori stampati a tempo di esecuzione. Si motivi opportunamente la risposta.

```
#define LEN 5
int i = 2;

void modify(int n, int * x, int * y, char * c)
{
    for (i = 0; i < n; ++i){
        if (x[i] >= 10) y[i] = 1;
        else y[i] = 0;
    }

    i -= 5;
    while (i < LEN){
        if (*(y+i) > 0) c[i] = 'z';
        i++;
    }
}

int main()
{
    int *gamma;
    int alpha[LEN] = {7,3,5,1,10};
    char beta[LEN+1] = {'c','h','a','o','s','\0'};
    i = *(alpha) - i;
    gamma = (int*)malloc(i * sizeof(int));
    modify(i,alpha,gamma,beta);
    for (i = 0; i < LEN; ++i) printf("%d ",gamma[i]);
    printf("\n%s\n",beta);
    return 0;
}
```

b) Data la funzione:

```
int omega(int p, int q)
{
    if (q < 1) return p;
    else return omega(p+q,q-1);
}
```

si scriva il risultato della funzione quando invocata come `omega(2,1)` e si disegnino i corrispondenti record di attivazione. Si supponga che le espressioni siano valutate da sinistra verso destra. La funzione è *tail ricorsiva*?