

Fondamenti di informatica B (9 CFU)

Esame del 26 Giugno 2012

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 le statistiche relative alle visite delle pagine di un sito web. Il progetto deve essere strutturato come segue:

- Vi è una classe `WebSite` che aggrega un certo numero di pagine, oggetti di tipo `WebPage`.
- Il tipo `WebPage` è definito tramite l'interfaccia omonima, che contiene solo il metodo `int getID()` che restituisce l'ID della pagina.
- La classe `StatWebPage` che implementa l'interfaccia `WebPage`.

La classe `StatWebPage` è caratterizzata dai seguenti campi e metodi:

- `WebPage prev`: campo che mantiene la pagina dalla quale si è giunti alla presente.
- `final int ID`: campo che mantiene l'identificativo della pagina.
- `static int nextID`: campo che costituisce il prossimo identificativo libero da assegnare. Il campo è statico poiché è necessario assegnare un ID univoco per ogni pagina di classe `StatWebPage`. Pertanto, il costruttore della classe dovrà anche occuparsi di incrementare tale valore alla costruzione di ogni nuovo oggetto.
- Costruttore (senza parametri) che imposta l'identificativo della pagina e inizializza il campo `prev` a `null`.
- `WebPage getPrev()` e `void setPrev(WebPage p)`: metodi per restituire ed impostare `prev`, rispettivamente. Si supponga che tali metodi siano già implementati.

La classe `WebSite` ha i seguenti campi e metodi:

- `LinkedList<WebPage> pages`: campo privato che mantiene l'insieme delle pagine web.
- Costruttore (senza parametri).
- `void addPage(WebPage p)`: aggiunge una pagina al sito.
- `LinkedList<WebPage> getChain(WebPage dest)`: metodo che restituisce la sequenza di pagine visitate fino a `dest`. Nella ricostruzione della sequenza si tenga conto che ci si deve arrestare non appena una pagina ha il campo `prev` uguale a `null` oppure quando una pagina non è di classe `StatWebPage` (per quest'ultimo controllo si può fare uso dei metodi tipici della *reflection*). Al limite, la lista sarà costituita dalla sola pagina `dest`. Il metodo deve lanciare un'eccezione qualora `dest` sia `null`.
- `WebPage getOrigin(WebPage dest)`: restituisce la pagina a partire dalla quale si è giunti alla pagina `dest`. Il metodo deve lanciare un'eccezione qualora `dest` sia `null`.

Si implementino le classi così come descritte e si implementi anche un programma che mostri come utilizzare le classi e i metodi implementati. Definire opportunamente campi e metodi come *private*, *protected* o *public*, se non specificato nel testo. È possibile aggiungere metodi, campi e variabili qualora lo si ritenga opportuno. Si riporta uno stralcio della documentazione di interesse:

* Class `LinkedList<E>`

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

`void addFirst(E e)`: Inserts the specified element at the beginning of this list.

* Class `Object`

`Class<?> getClass()`: Returns the runtime class of this `Object`.

* Class `Class<T>`

`String getName()`: Returns the name of the entity

(class, interface, array class, primitive type, or void)
represented by this `Class` object, as a `String`.

Esercizio 2 (10/40)

Si analizzi il codice seguente e si riporti l'output prodotto dall'esecuzione del programma (che, per semplicità, si suppone scritto in un unico *file*). Le risposte devono essere opportunamente motivate (per esempio tramite uno schema che riproduce legami e informazioni di tipo relative a riferimenti e oggetti).

```
interface Thomson {
    public void set(int v);
}

abstract class Michelson implements Thomson {
    protected int v;
    public Michelson(){this.v = 0;}
    public abstract void set(int v);
    public String toString() { return "value: " + v; }
}

class Marconi extends Michelson {
    public Marconi(){
        super();
        System.out.println("Marconi"); }
    public void set(int v){ this.v += v; }
}

class Planck extends Michelson {
    public Planck(){
        super();
        v++;
        System.out.println("Planck"); }
    public void set(int v){ v += v; }
    public String toString(){ return "constant value: " + v; }
}

public class Esercizio2 {
    public static void main(String [] args){
        Thomson alpha = new Marconi();
        Thomson beta = new Planck();
        Michelson gamma = new Marconi();
        alpha.set(1); beta.set(2); gamma.set(3);
        System.out.println(alpha); System.out.println(beta);
        System.out.println(gamma);
    }
}
```

Esercizio 3 (8/40)

Si completi l'implementazione sotto riportata del pannello di una semplice applicazione la cui interfaccia grafica è costituita da:

- Due campi di testo modificabili denominati rispettivamente **prezzo** e **variazione**.
- Un campo di testo non modificabile denominato **totale**.
- Un gruppo di 3 bottoni radio rispettivamente denominati **sconto**, **maggiorazione**, **intero**.
- Un bottone JButton **calcola**.

Alla pressione del bottone, nel campo **totale** deve essere riportato il prezzo modificato secondo il valore v indicato nel campo di testo **variazione**. L'operazione da eseguire dipende dal bottone radio selezionato, secondo il seguente schema:

- **sconto**: il prezzo deve essere decrementato del valore v ;
- **maggiorazione**: il prezzo deve essere aumentato del valore v ;
- **intero**: il prezzo rimane inalterato.

La gestione degli eventi sia realizzata implementando un'opportuna classe ascoltatore, separata dal pannello.

Per sapere se un bottone radio è selezionato si usi il metodo `isSelected()`. Si ricordi che i bottoni radio sono aggiunti al gruppo (`ButtonGroup`) tramite il metodo `add(JRadioButton jrb)`. Per convertire una stringa in *double* si usi il metodo statico `parseDouble(String s)` (che può lanciare un'eccezione) della classe `Double`.

```
class MyPanel extends JPanel {
    private JTextField prezzo, totale, variazione;
    private JButton calcola;
    private JRadioButton [] rb;
    private ButtonGroup g;

    public MyPanel() {
        ...[1]...
    }
}

public class MyListener implements ActionListener {
    ...[2]...

    public void actionPerformed(ActionEvent ev) {
        ...[3]...
    }
}
```

Esercizio 4 (10/40) – per il corso di Fondamenti di informatica B (9CFU)

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

```
#include <stdio.h>

int val = 5;

int turing(int * n, char * myc)
{
    int j;
    int temp = val - *n;

    for(j = temp; j < *n; j+=2)
    {
        myc[j] = 'm';
    }

    return j;
}

int main()
{
    char s[] = "ABCDEF";
    int i,j;

    i = val++;
    j = turing(&i,&s[0]);

    printf("%s,%d,%d\n",s,i,j);

    return 0;
}
```

b) Data la funzione:

```
#include <stdio.h>

int turing(int x, int y)
{
    int h = x % y;

    if (h == 0) return x + y;
    else return turing(x,y-1);
}
```

si scriva il risultato della funzione quando invocata come `turing(4,3)` e si disegnino i corrispondenti record di attivazione. Si supponga che le funzioni siano valutate in ordine da sinistra verso destra. Indicare anche se la funzione è *tail ricorsiva* e motivare la risposta.