

Fondamenti di informatica B

Esame del 22 Giugno 2010

Compito B

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'. Coloro i quali consegnano entro 2 ore mantengono il voto conseguito nella prova in itinere per quanto attiene all'esercizio 4. Si ricorda che la validità di tale prova termina con questo appello.

Esercizio 1 (12/40)

Si progetti la parte principale di un'applicazione che realizza un'agenda giornaliera di impegni. L'impegno generico è modellato tramite la classe **astratta** `ADuty` così strutturata:

- Campi `String name`, `int priority` e `int hour` che rappresentano, rispettivamente, il nome dell'impegno, la sua priorità e l'ora del giorno (0–23) in cui deve essere svolto.
- Metodi pubblici per il recupero dei valori dei campi sopra citati.
- Costruttore con tre parametri (per i tre campi) che lancia un'eccezione qualora il parametro attuale corrispondente a `hour` sia minore di 0 o maggiore di 23.

La classe `ADuty` è estesa dalle classi `SimpleTask` e `LongTask`. Entrambe le classi devono ridefinire il metodo `toString()` e definire un proprio costruttore che inizializzi i campi della classe, descritti di seguito.

La classe `SimpleTask` ha il seguente campi:

- `String description`: campo che descrive il task.

La classe `LongTask` ha i seguenti campi:

- `int duration`: campo che indica la durata dell'impegno.
- `String memo`: campo che riporta informazioni aggiuntive.

L'agenda è realizzata dalla classe `Scheduler` così strutturata:

- `LinkedList<ADuty> dutyList`: campo che contiene la lista degli impegni.
- Costruttore di default.
- `ADuty addDuty(String name, int priority, int hour, LinkedList<Object> info, String type)`: metodo pubblico che permette di creare e inserire un impegno nella lista. A seconda del valore del parametro `type` ("SimpleTask"/"LongTask"), si dovrà creare un'istanza di `SimpleTask` o `LongTask` invocando il relativo costruttore. Il parametro `info` contiene la lista dei parametri aggiuntivi rispetto a quelli comuni; nel caso di "SimpleTask", `info` avrà un primo elemento da interpretarsi come `String` per il campo `description`, mentre nel caso di "LongTask", la lista `info` sarà costituita da un primo elemento da convertire in `Integer` e un secondo da convertire in `String`, rispettivamente per `duration` e `memo`. Qualora si provi ad aggiungere un impegno con nome già esistente, deve essere generata un'eccezione. Il metodo restituisce un riferimento al task creato, se possibile, oppure a `null` altrimenti.
- `HashMap<String,Integer> exportAsMap()`: metodo pubblico che restituisce gli impegni come mappa in cui la chiave è il nome dell'impegno e il valore è l'ora del giorno in cui si svolge.

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:

```
Class LinkedList<E>
```

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

```
Class HashMap<K,V>
```

```
V put(K key, V value): Associates the specified value
```

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 Bach {
    protected int i;
    public Bach(){this.i = 2;}
    public String getNumber(){ return ""+(++i);} }

class Vivaldi extends Bach {
    public Vivaldi(){
        super();
        System.out.println("Vivaldi"); }

    public String getNumber(){return super.getNumber();} }

class Chopin extends Bach {
    public Chopin(){
        super();
        i -= 2;
        System.out.println("Chopin"); }
    public String getNumber(){ return ""+(i++);} }

class Corelli extends Chopin{
    public Corelli(int i){
        super();
        i++;
        System.out.println("Corelli"); }
    public String getNumber(){return "3";} }

public class Esercizio2 {
    public static void main(String [] args){
        Bach z = new Chopin(); Bach x = new Bach(); Chopin m = new Corelli(7);
        Bach y = new Vivaldi();
        System.out.println("x: "+x.getNumber());
        System.out.println("y: "+y.getNumber());
        System.out.println("z: "+z.getNumber());
        System.out.println("m: "+m.getNumber());
        Corelli f = (Corelli) m;
        System.out.println("f: "+f.getNumber());
        System.out.println("z: "+z.getNumber());
    }
}
```

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 modificabile `txt1`.
- Un campo di testo modificabile `txt2`.
- Un array di `JCheckBox` `ck`, sorgenti di eventi.

Quando viene selezionato o deselezionato un elemento in `ck`, nel campo di testo `txt2` deve essere riportata l'etichetta corrispondente all'elemento selezionato, eventualmente concatenata con il contenuto di `txt1`, ma solo se si tratta di un valore numerico (`int`). Si implementi l'ascoltatore tramite un oggetto che sia istanza di una classe opportunamente progettata.

```
class MyPanel extends JPanel {
    private JTextField txt1, txt2; private JCheckBox ck[];

    public MyPanel() {
        super();
        txt1 = new JTextField(20); txt2 = new JTextField(20);
        ck = new JCheckBox[2];
        ck[0] = new JCheckBox("Alpha"); ck[1] = new JCheckBox("Beta");
        ...[1]...
    }
}

public class MyListener implements ActionListener {
    private JTextField txt1, txt2;
    public MyListener(JTextField txt1, JTextField txt2) {...[2]...}
    public void actionPerformed(ActionEvent e) {...[3]...}
}
```

Documentazione di interesse:

```
class Integer
public static int parseInt(String s) throws NumberFormatException:
Returns a new double initialized to the value represented
by the specified String
```

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

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.

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 5

int value = 6;

char * fun1(char * s, int * v, int * n);

int main()
{
    int i;
    int d = value;
    int v[SIZE];
    char z[] = "abcdef";
    char *t,*s;

    d++;
    t = (char*)malloc(d * sizeof(char));

    for(i = 1; i <= SIZE; i++)
    {
        t[i] = *(z + i - 1);
        v[i-1] = i * i;
    }

    t[i] = '\0';

    s = fun1(&t[1],v,&i);

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

    return 0;
}
```

```

char * fun1(char * s, int * v, int * n)
{
    int value;
    *n = 2;

    for(value = 0; value <= *n; value++)
    {
        printf("%d %c\n",v[value],*(s+value));
        if (value % 2 == 1) s[value] = 'x';
    }

    return s;
}

```

b) Data la funzione:

```

int tau(int a, int b)
{
    int t;

    t = ++a;
    if (b < t + 2) return a - 1;
    else return tau(a+1,b) * tau(b-a,t);
}

```

si scriva il risultato della funzione quando invocata come `tau(1,5)` e si disegnino i corrispondenti record di attivazione. Si supponga che l'esecuzione delle funzioni nella chiamata ricorsiva sia effettuata da sinistra verso destra.