

Fondamenti di informatica B

Esame del 19 Luglio 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 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 time` che rappresenta l'istante di tempo in cui l'evento avviene.
- Campo `int duration` che rappresenta la durata dell'evento.
- Campo `boolean blocking` che indica se l'evento è bloccante, cioè se nessun altro evento può avvenire durante l'esecuzione di questo.
- Campo `String info` che rappresenta le informazioni relative all'evento.
- Metodi pubblici concreti `int getTime()`, `int getDuration()`, `String getInfo()` che restituiscono i valori dei rispettivi campi. Per semplicità, si supponga che tali metodi siano già implementati.
- Costruttore con quattro argomenti, rispettivamente per inizializzare i campi `time`, `duration`, `info` e `blocking`. Qualora si provi a creare un evento con durata minore di 1, deve essere generata un'eccezione.
- Metodo pubblico astratto `void happen(State state)` che ha lo scopo di modificare lo stato della simulazione e sarà implementato dalle sottoclassi concrete.

Lo stato della simulazione è rappresentato dalla classe **State**, così organizzata:

- Campo `String state` che mantiene le informazioni dello stato.

- Costruttore senza parametri che inizializza `state` con la stringa “empty”.
- Metodo `String getValue()` che restituisce `state`.
- Metodo `void update(AbstractEvent ev)` che sovrascrive `state` con il valore del campo `info` dell’evento `ev`.

La classe `AbstractEvent` è estesa dalle classi `SimpleEvent` e `ConditionalEvent`. La classe `SimpleEvent` implementa il metodo `happen()` invocando semplicemente il metodo `update()` sullo stato, passando il riferimento a sé stesso. Invece, la classe `ConditionalEvent` è così organizzata:

- Il costruttore ha un quinto parametro, di tipo `String`, che rappresenta la condizione, mantenuta nel campo `String condition`.
- Nel metodo `happen()`, l’invocazione al metodo `update()` dello stato è effettuata soltanto se lo stato corrente contiene una sottostringa uguale a `condition`.

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

- Campo `int time` che rappresenta l’istante di tempo corrente.
- Campo `State state` che rappresenta lo stato della simulazione.
- `ArrayList<AbstractEvent> eventList`: campo che contiene la lista degli eventi.
- Costruttore di default che inizializza `eventList` e `time` (a zero).
- `ArrayList<AbstractEvent> getNextEvents()`: metodo privato che restituisce la lista degli eventi che abbiano il campo `time` uguale al valore di `time` del simulatore. Nel caso in cui non esistano eventi con queste caratteristiche, si restituisca una lista vuota.
- `void addEvent(AbstractEvent e)`: metodo pubblico che permette di inserire un evento in lista.
- Si supponga che sia già implementato il metodo privato `int getRandomInt(int a, int b)` che restituisce un intero $x \in [a, b)$ scelto secondo una distribuzione uniforme.

- `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 :
    Sia l la lista degli eventi al tempo time
    se l non è vuota allora :
        Sia ev un evento scelto a caso in l
        Rimuovi ev dalla lista degli eventi eventList
        Fai avvenire ev
        Stampa il tempo corrente e lo stato
        se ev è bloccante incrementa time di un valore
                                pari alla durata di ev
        altrimenti incrementa time di una unità
    altrimenti:
        Stampa il tempo corrente e lo stato
        Incrementa time di una unità

```

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. Si riporta uno stralcio della documentazione di interesse:

```

Class ArrayList<E>
boolean add(E o): Appends the specified element to the end of this list.
E get(int index): Returns the element at the specified position in this list.
boolean remove(Object o): Removes the specified element from this list,
                        if it is present.

Class String
boolean contains(String s): Returns true if and only if this string contains
                        the specified string s.

```

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 Omega{
    public Omega(){System.out.println("Class Omega");}
    public abstract void method();}

class Tau extends Omega{
    public Tau(){
        super();
        System.out.println("Class Tau");
    }
    public void method(){System.out.println("Tau method");}
}

class Sigma extends Tau{
    public Sigma(){
        super();
        System.out.println("Class Sigma");
    }
    public void method(){System.out.println("Sigma method");}
}

class Rho {
    public Rho(Omega obj){
        System.out.println("First constructor");
        obj.method();
    }

    public Rho(Sigma obj){
        System.out.println("Second constructor");
        obj.method();
    }
}

public class Esercizio2 {
    public static void main(String [] args){
        Omega obj = new Sigma();
        Rho x = new Rho((Sigma) obj);
        Rho y = new Rho(obj);
    }
}
```

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 `txt`, un oggetto `JComboBox list` e un pulsante `JButton b` che è sorgente di eventi.

Alla pressione del pulsante, il contenuto del campo di testo deve essere aggiunto come *item* in `list` alla seguente condizione: il campo di testo non deve contenere la stringa vuota e il numero di *item* già inseriti non deve aver già raggiunto il limite impostato.

```
class MyPanel extends JPanel {
    private JTextField txt;
    private JComboBox list;
    private JButton b;

    public MyPanel() {
        super();
        txt = new JTextField(30);
        list = new JComboBox();
        b = new JButton("add item");
        ...[1]...
    }
}

public class MyListener implements ActionListener {
    private JTextField txt;
    private JComboBox cb;
    private int maxItem;
    private int nItem;

    public MyListener(JTextField txt, JComboBox cb, int maxItem) {
        ...[2]...
    }

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

Documentazione di interesse:

```
class JComboBox
void addItem(Object anObject): Adds an item to the item list.
```

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 N 5

int n = -1;

int fun(int * x, int * y, int n);

int main()
{
    int i;
    int n = N;
    int u[N] = {0,1,2,3,4};
    int v[N];
    int *s,*t;

    for(i = 0; i < N; i++) v[i] = i+2;
    s = u;
    t = &v[1];
    n = fun(s,t,n);
    printf("%d\n",n);

    return 0;
}

int fun(int * x, int * y, int n)
{
    int *p;
    int sum = 0;
    n = 3;

    for(p = x; p < x+n; p++) {
        printf("%d %d\n",*p,y[*p]);
        if (y[*p] % 2 == 0) *p = 0;
        sum += *p;
    }
    return sum;
}
```

b) Data la funzione:

```
int zeta(int a, int b)
{
    if (b >= 10) return a;
    else return zeta(a,b+a) - 2;
}
```

si scriva il risultato della funzione quando invocata come **zeta(3,1)** e si disegnino i corrispondenti record di attivazione. La funzione è *tail ricorsiva*?