

Fondamenti di informatica B/LB

Esame del 6 Giugno 2012

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.
- 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)

In questo esercizio si chiede di realizzare una parte di un programma di analisi linguistica. Il progetto deve essere strutturato secondo le specifiche riportate di seguito.

Vi è un'interfaccia **Grammar** che rappresenta l'entità che mantiene le informazioni principali riguardanti la grammatica di una lingua generica. L'interfaccia ha i seguenti metodi:

- `String getLexCat(String w)`, che restituisce la categoria lessicale della parola (e.g., "verb");
- `boolean inDict(String w)`, che restituisce `true` se la parola è contenuta nel dizionario;
- `void addWord(String w, String lexCat)`, che permette di inserire una nuova parola, insieme alla sua categoria lessicale, nel dizionario. Il metodo lancia un'eccezione nel caso in cui la categoria lessicale non sia tra quelle ammesse.

L'interfaccia **Grammar** è implementata dalla classe **English**. Oltre ai metodi di **Grammar**, essa possiede:

- un campo, `List<String> lexCategories`, che mantiene la lista delle categorie lessicali ammesse;
- un metodo `boolean feasibleLexCat(String lc)` che restituisce `true` nel caso in cui `lc` sia contenuta in `lexCategories`;
- un costruttore, con parametro `List<String> lexCat`, tramite il quale si crea la lista delle categorie lessicali della grammatica. Il campo della classe deve quindi essere un nuovo oggetto, creato a partire da `lexCat`.

Si implementi tramite `HashMap<String,String>` la struttura dati che mantiene il dizionario (che sarà quindi un campo della classe). La chiave sia la parola e il valore la sua categoria lessicale. Per esempio: (*androids, noun*). Il metodo `getLexCat(String w)` deve restituire `null` nel caso in cui la parola non sia presente nel dizionario.

Si definisca, inoltre, la classe `CheckWord` dotata di:

- `Grammar gram`: campo che mantiene la grammatica;
- `boolean find(String word, String cat, List<String> sentence)`: metodo che restituisce `true` se nella frase `sentence` è presente una parola `word` con categoria lessicale `cat`. Per esempio, se la frase è data da (*do,androids,dream,of,electric,sheep*), `word = "sheep"`, `cat = "noun"` e il dizionario contiene (*sheep,noun*), allora il metodo deve restituire `true`;
- costruttore con un argomento di tipo `Grammar` che sarà utilizzato per inizializzare il campo relativo.

Si implementino le classi così come descritte e si implementi anche un programma che mostri come utilizzare le classi e i metodi implementati. In particolare, si crei una grammatica che riconosca le seguenti categorie: *verb* e *noun*. 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:

`LinkedList<E>` (equiv. `ArrayList<E>`):

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

`boolean contains(Object o)`: Returns true if this list contains
the specified element.

`LinkedList(Collection<? extends E> c)`: Constructs a list containing the elements
of the specified collection.

`HashMap<K,V>`:

`boolean containsKey(Object key)`: Returns true if this map contains
a mapping for the specified key.

`V get(Object key)`: Returns the value to which the specified key is mapped,
or null if this map contains no mapping for the key.

`V put(K key, V value)`: Associates the specified value with the specified key
in this map.

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 Morpheus {
    void talk();
}

class Trinity implements Morpheus {
    public Trinity() { System.out.println("Trinity"); }
    public void talk() { System.out.println("What is real?"); }
}

class Neo extends Trinity {
    public Neo() { super(); System.out.println("Neo"); }
    public void talk() {
        super.talk();
        System.out.println("How do you define real?");
    }
}

class Oracle {
    public Oracle(Morpheus r) {
        System.out.println("Blue pill");
        r.talk();
    }
    public Oracle(Neo r) {
        System.out.println("Red pill");
        r.talk();
    }
}

public class Esercizio2 {
    public static void main(String [] args){
        Morpheus obj1 = new Neo();
        Oracle obj2 = new Oracle((Neo) obj1);
        Oracle obj3 = new Oracle(obj1);
    }
}
```

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 denominato **prezzo**.
- Un campo di testo modificabile denominato **magg**.
- Un campo di testo non modificabile denominato **totale**.
- Un componente JCheckBox.
- Un bottone JButton.

Alla pressione del bottone, se la *checkbox* è selezionata, nel campo **totale** deve essere riportato il prezzo maggiorato della percentuale indicata nel campo di testo **magg**; altrimenti, deve essere riportato il prezzo intero. Nel caso di applicazione della maggiorazione, prima di effettuare il calcolo, si deve verificare che il valore di incremento sia strettamente maggiore di 0; se così non è, deve essere riportato il prezzo intero. La gestione degli eventi sia realizzata implementando un'opportuna classe ascoltatore, separata dal pannello.

Ricordare che l'interfaccia da implementare è **ActionListener** e il corrispondente metodo è `actionPerformed(ActionEvent e)`; l'ascoltatore è collegato alla sorgente tramite il metodo `addActionListener`. Per sapere se la *checkbox* è selezionata si usi il metodo `isSelected()`. 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 {
    JTextField prezzo, totale, magg;
    JButton calcola; JCheckBox ck;

    public MyPanel() {
        super();
        prezzo = new JTextField(10); magg = new JTextField(10);
        totale = new JTextField(10); totale.setEditable(false);
        ck = new JCheckBox("maggiorazione");
        calcola = new JButton("CALCOLA");
        ...[1]...
    }
}

public class ButtonListener implements ActionListener {
    ...[2]...
    public void actionPerformed(ActionEvent e) {
        ...[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 z = 7;

int elab(char * x, int * y) {
    int z = *y;
    int i = 0;
    while (x[i++] != '\0') {
        if (i % 2 == 1) x[i-1] = '0';
        else z++;
    }
    return i;
}

int main() {
    char x[] = "12345";
    int i;
    i = elab(&x[1], &z);
    printf("%s,%d,%d\n", x, i, z);
    return 0;
}
```

b) Data la funzione:

```
int beta(int a, int b)
{
    int x = b++;

    if (x + a > 6) return x;
    else return beta(2*a,b) + beta(a,2*b);
}
```

si scriva il risultato della funzione quando invocata come `beta(3,2)` 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.

Esercizio 4 (10/40) – per il corso di Fondamenti di informatica LB (6CFU)

Descrivere il pattern model-view-controller.