

Corso di Programmazione e Algoritmi 2
Modulo di Laboratorio
A.A. 2019/2020

Progetto

Regole per lo svolgimento:

- Il progetto può essere svolto **singolarmente** o **in gruppi di qualsiasi dimensione**.
In ogni caso, ciascuno studente dovrà poi **singolarmente** discutere il progetto dimostrando di essere in grado di modificare e/o aggiungere alcune parti. In caso di esito sufficiente di tale discussione, **verrà attribuito il punteggio da 0 a 3 punti**.
- Il progetto deve essere consegnato **almeno 7 giorni prima** della prova orale di *Programmazione e Algoritmi 2* relativa all'appello in cui si desidera registrare l'esame.
- Il progetto deve contenere le classi, i campi e i metodi richiesti **rispettandone esattamente il nome, il tipo e l'ordine dei parametri formali, ed il tipo di ritorno**.
- Nel file allegato è presente la specifica (parziale), per ***qualcuna delle classi richieste***, dei campi e dei metodi che bisogna implementare.
Le altre classi devono essere opportunamente aggiunte da voi nella consegna. Trattandosi di classi che estendono altre classi presenti nel file, la loro specifica si deduce in modo inequivocabile.
- Si tenga presente che **non sono sempre presenti** tutti i modificatori (***public, private, protected, static, final, ...***) che devono essere opportunamente aggiunti da voi nella consegna.
- I **metodi accessori e modificatori** non sono tutti presenti nella specifica. Deve essere vostra cura aggiungerli in modo opportuno.
- Si è naturalmente liberi di sviluppare metodi aggiuntivi di supporto ai metodi richiesti, laddove lo si ritenga utile.
- Deve essere consegnato via email all'indirizzo luca.moscardelli@unich.it un file compresso contenente tutti i file sorgenti e avente come nome
[java] progetto 2019_2020.zip
l'email deve avere come oggetto
[java] consegna progetto programmazione 2019/2020
e deve contenere nel testo la lista (Cognome, Nome e Matricola) di tutti gli studenti che hanno svolto il progetto consegnato.
In caso di consegna in gruppo, la consegna deve essere effettuata da uno dei componenti del gruppo, mettendo in CC nell'email anche l'indirizzo di tutti gli altri componenti.

Testo del progetto:

Sviluppare in **Java** un package **minilinguaggio** per la rappresentazione ed esecuzione di programmi in un semplice linguaggio di programmazione in cui ci sono:

- Espressioni numeriche di tipo intero (equivalente al tipo **int** del Java);
- Variabili di tipo intero (equivalente al tipo **int** del Java);
- Espressioni booleane;
- 6 possibili comandi:
 - comando vuoto, che non fa nulla;
 - comando assegnamento, che assegna ad una variabile (identificata da una stringa) un valore intero;
 - comando che stampa di un'espressione numerica;
 - comando if ... else, con la consueta semantica;
 - comando while, con la consueta semantica;
 - comando sequenza, la cui esecuzione è data dall'esecuzione di altri 2 comandi in sequenza.

Si noti che per memorizzare il valore delle variabili si ha bisogno di uno **stato** che evolve durante l'esecuzione dei comandi. Per implementare lo stato si usi una **tabella hash** (sfruttando i file implementati durante le ore di laboratorio) in cui si memorizzano coppie di valori (identificatore, valore), in cui l'identificatore è una stringa e il valore un intero; *si noti che la chiave su cui effettuare l'hashing, ai fini del posizionamento in tabella, è esclusivamente l'identificatore.*

Il package deve contenere le seguenti classi/interfacce, tutte pubbliche:

1. Cmd (classe astratta)

Tipo ricorsivo per rappresentare i comandi del linguaggio. Tale classe astratta ha le seguenti classi che la estendono per rappresentare le espressioni le cui valutazioni sono indicate di seguito:

- **CmdVuoto** (caso base, il comando che non fa nulla),
- **CmdAssegnamento** (caso base, il comando che assegna ad una variabile, identificata da una stringa, la valutazione di un'espressione numerica di tipo intero),
- **CmdStampa** (caso base, il comando che stampa il valore di un'espressione numerica intera),
- **CmdIfElse** (comando che a seconda della valutazione di una **BoolExp bE** esegue il **Cmd cmdIf** se **bE** è vera, mentre esegue il **Cmd cmdElse** se **bE** è falsa, in modo analogo alla semantica dell'if...else del linguaggio Java),
- **CmdWhile** (comando che finché la valutazione della **BoolExp bE** è vera esegue il **Cmd cmd**, in modo analogo alla semantica del while del linguaggio Java),
- **CmdSequenza** (comando che esegue in sequenza i due comandi **Cmd cmd1** e **Cmd cmd2**).

2. Exp (classe astratta)

Tipo ricorsivo per rappresentare espressioni numeriche la cui valutazione è un numero intero. Tale classe astratta ha le seguenti classi che la estendono per rappresentare le espressioni le cui valutazioni sono indicate di seguito:

- o **ExpNumero** (il caso base, l'espressione formata semplicemente da un numero intero),
- o **ExpVariabile** (altro caso base, l'espressione formata semplicemente da una variabile e che viene valutata numero intero),
- o **ExpSomma** (somma tra le valutazioni A e B di due **Exp e1** ed **e2**),
- o **ExpDifferenza** (differenza tra le valutazioni A e B di due **Exp e1** ed **e2**),
- o **ExpProdotto** (prodotto tra le valutazioni A e B di due **Exp e1** ed **e2**),
- o **ExpQuoziente** (divisione intera tra le valutazioni A e B di due **Exp e1** ed **e2**),
- o **ExpResto** (resto della divisione intera tra le valutazioni A e B di due **Exp e1** ed **e2**),
- o **ExpOpposto** (opposto della valutazione A dell'**Exp e**)

3. **BoolExp** (classe astratta)

Tipo ricorsivo per rappresentare espressioni la cui valutazione è un valore booleano. Tale classe astratta ha le seguenti classi che la estendono per rappresentare le espressioni indicate tra parentesi:

- o **BoolExpTrue** (caso base, l'espressione che vale sempre *true*),
- o **BoolExpFalse** (caso base, l'espressione che vale sempre *false*),
- o **BoolExpUguale** (l'espressione che confronta le valutazioni A e B di due **Exp e1** ed **e2** e vale *true* se e solo se $A = B$),
- o **BoolExpLEq** (l'espressione che confronta le valutazioni A e B di due **Exp e1** ed **e2** e vale *true* se e solo se A è minore di o uguale a B),
- o **BoolExpGEq** (l'espressione che confronta le valutazioni A e B di due **Exp e1** ed **e2** e vale *true* se e solo se A è maggiore di o uguale a B),
- o **BoolExpAnd** (l'AND di due espressioni booleane **BoolExp bE1** ed **bE2**)
- o **BoolExpOr** (l'OR di due espressioni booleane **BoolExp bE1** ed **bE2**)
- o **BoolExpNot** (il NOT di una espressione booleana **BoolExp bE**)

4. **State** (interfaccia)

Interfaccia che deve essere implementata da una classe che **gestisce lo stato della memoria del programma**, ovvero l'associazione tra identificatori e valori. Si noti che deve quindi anche essere implementata una opportuna classe che implementa questa interfaccia (che utilizzi una tabella hash per memorizzare le variabili).

Note:

- Il programma deve sollevare **opportune eccezioni** in 2 casi:
 - o Qualora si valuti una **ExpQuoziente** o **ExpResto** in cui il divisore ha valutazione 0.
 - o Qualora si valuti una **ExpVariabile** in cui l'identificatore non si riferisce ad una variabile a cui è stato assegnato un valore
- Il file compresso allegato con una parte dei sorgenti java contenenti ulteriori specifiche costituisce parte integrante di questo documento.
- Tutte le classi (tranne eventualmente quelle astratte) devono sovrascrivere il metodo **toString()** in modo opportuno
- Non è necessario consegnare il metodo **main** (che voi potete chiaramente utilizzare per i vostri test)
- L'unica libreria del Java che potete usare nei file del package **minilinguaggio** è **java.util.ArrayList** (oltre naturalmente alle classi del package **java.lang**)