# Progetto *ALanguage*

Progetto per il corso di Programmazione 2008/09

## 1 Descrizione

Il progetto consiste nell'implementare in Java un semplice linguaggio di programmazione, in cui le variabili sono esclusivamente di tipo intero.

## 2 Premessa: la gerarchia `Map`

Per la realizzazione del progetto, sarà conveniente l'uso della gerarchia `Map` (nel pacchetto `java.util`). Se ne fornisce qui una breve descrizione (relativa alla versione Java 1.5) per rendere comprensibile il testo del progetto. `Map` è un'interfaccia generica le cui istanze rappresentano essenzialmente delle funzioni; più precisamente, dati due tipi `K` e `V` (chiamati, rispettivamente, tipo delle chiavi e tipo dei valori), una `Map<K,V>` è un insieme di coppie chiave/valore, dove la chiave è un'istanza di `K`, il valore è un'istanza di `V` e a non accade mai che alla stessa chiave siano associati due valori diversi.

Per esempio, una `Map<String,Integer>` è un'insieme di coppie

$$\{(s_1, v_1), \ldots, (s_n, v_n)\}$$

dove $s_1$, ..., $s_n$ sono $n$ stringhe distinte e $v_1$, ..., $v_n$ sono $n$ interi (oggetti di tipo `Integer`) non necessariamente distinti.

Fra i metodi disponibili nell'interfaccia `Map<K,V>` ricordiamo:

- `put(K x, V y)`: aggiunge alla mappa la coppia chiave/valore $(x, y)$; se la chiave $x$ risultava già associata a un qualche valore, la precedente associazione viene cancellata;
- `get(K x)`: restituisce il valore associato alla chiave $x$, oppure `null` se la chiave non è associata a nessun valore;
- `containsKey(K x)`: restituisce `true` se e solo se la chiave $x$ è associata a qualche valore.

L'interfaccia `Map` ha varie implementazioni; noi consigliamo di usare `HashMap`. Quindi, ad esempio, il seguente frammento di codice:

```
Map<String,Frazione> m;   // Dichiara una mappa stringhe -> frazioni
m = new HashMap<String,Frazione>(); // La crea vuota
m.put( "pippo", new Frazione( 3, 4 ) );
m.put( "pluto", new Frazione( 6, 5 ) );
```

crea una mappa che contiene due coppie (qui le chiavi sono stringhe e i valori sono frazioni).

# 3 Le classi da realizzare

È richiesto di risolvere il progetto descritto nella sezione precedente realizzando in Java le seguenti classi:

- `Istruzione`, che descrive una generica istruzione del linguaggio. La classe dovrà contenere:
  - la definizione di una `Map` di nome `memoria`, deputata a contenere i valori memorizzati nelle variabli, utilizzando come chiavi i nomi dati alle variabili (delle stringhe) e come valori i loro valori attuali (degli interi); attenzione: la definizione di questa mappa dovrà essere compatibile con il resto delle definizioni fatte nella classe, e, soprattutto, dovrà essere fatta in modo da essere leggibile e scrivibile da tutte le istanze delle classi derivate da `Istruzione`, nonché condivisa da queste istanze;
  - la definizione del metodo astratto `void esegui()`, da chiamare per eseguire l'istruzione; durante l'esecuzione di tale metodo potrà essere sollevata l'eccezione `ExecutionException`, rappresentante un generico errore di run time;
  - la definizione del metodo astratto `String toString(int x)` che restituisce la rappresentazione stampabile dell'istruzione *indentata a livello* $x$: con questo intendiamo che le varie righe che costituiscono la stringa devono iniziare ciascuna con $Kx$ spazi, dove $K$ è un campo statico finale intero (una costante) dichiarata in `Istruzione` con valore 4;
  - invocare il metodo `String toString()` su un'istruzione deve essere equivalente a invocare `toString(0)`.
- `Sequenza`, che descrive una generica sequenza di istruzioni. La classe dovrà contenere:
  - la variabile di istanza `Istruzione elenco[]`, deputata a contenere l'insieme delle istruzioni da eseguire in sequenza;
  - il costruttore `Sequenza(Istruzione e[])`, che inizializza la variabile di istanza della classe copiando in essa i contenuti dell'array passato come argomento;
  - l'implementazione del metodo `esegui()` della superclasse dovrà essere fatta in modo da eseguire in sequenza tutte le istruzioni contenute nell'array elenco, dalla prima all'ultima posizione;
  - l'implementazione del metodo `toString(x)` della superclasse dovrà restituire la stringa così costruita: $x$ spazi, il carattere `{` seguito da un a-capo, le varie istruzioni dell'elenco, ciascuna indentata di $x + 1$, infine $x$ spazi, il carattere `}` e un a-capo.
- `Selezione`, che descrive l'esecuzione condizionata di un'istruzione sulla base dei contenuti di una variabile. La classe dovrà contenere:
  - le variabili di istanza `Istruzione istrSe` e `Istruzione istrAltrimenti`, che identificano due istruzioni;

- la variabile di istanza `String variabile`, che contiene il nome di una variabile;
- la variabile di istanza `String operatore`, che dovrà contenere la descrizione dell'operatore di confronto da utilizzare; i valori possibili per questa variabile sono `>`, `>=`, `<`, `<=`, `==` e `!=`, il cui significato è mutuato da quello degli analoghi operatori in Java;
- la variabile di istanza `int valore`, che dovrà contenere il valore da confrontare con il contenuto della variabile;
- il costruttore

  ```
  Selezione(Istruzione se, Istruzione altr, String var, String o, int v)
  ```

  che inizializza le variabili di istanza della classe copiando in essa le istruzioni, le stringhe e la variabile intera passate come argomento, tenendo conto del fatto che l'ordine degli argomenti equivale all'ordine in cui sono state presentate le variabili di istanza;
- l'implementazione del metodo `esegui()` della superclasse dovrà essere fatta in modo da valutare una condizione booleana confrontando, utilizzando l'operatore descritto in operazione, il valore contenuto nella variabile il cui nome è contenuto in `variabile` con il contenuto di `valore`. Quando questa condizione è vera, verrà eseguita l'istruzione identificata da `istrSe`, mentre quando la condizione è falsa verrà eseguita `istrAltrimenti`; il metodo dovrà sollevare l'eccezione `InvalidVariableException` qualora non sia stata precedentemente dichiarata (attraverso l'istruzione `Dichiara`, descritta in seguito) una variabile il cui nome è contenuto in `variabile`; dovrà inoltre essere sollevata l'eccezione `InvalidOperatorException` nel caso in cui la descrizione dell'operazione non contenga uno dei simboli `>`, `>=`, `<`, `<=`, `==` o `!=`;
- l'implementazione del metodo `toString(x)` della superclasse dovrà restituire la stringa così costruita: $x$ spaxi, la costante `IF (`, il nome della variabile, l'operatore, il valore, la costante `)` seguita da un a-capo, l'istruzione `se` indentata di $x + 1$, la costante `ELSE` seguita da un a-capo, l'istruzione `altrimenti` indentata di $x + 1$.

- `Iterazione`, che descrive l'esecuzione iterata di un'istruzione sulla base del contenuto di una variabile. La classe dovrà contenere:
  - la variabile di istanza `Istruzione corpo`, che identifica l'istruzione da eseguire;
  - la variabile di istanza `String variabile`, che contiene il nome di una variabile;
  - la variabile di istanza `String operatore`, che dovrà contenere la descrizione dell'operatore di confronto da utilizzare; i valori possibili per questa variabile sono `>`, `>=`, `<`, `<=`, `==` e `!=`, il cui significato è mutuato dal significato degli analoghi operatori in Java;
  - la variabile di istanza `int valore`, che dovrà contenere il valore da confrontare con il contenuto della variabile;

- il costruttore `Iterazione(Istruzione c, String v, String o, int val)`, che inizializza le variabili di istanza della classe copiando in esse la stringa contenente il nome della variabile, l'istruzione, la stringa che identifica l'operatore di confronto e la variabile intera passate come argomento, tenendo conto del fatto che l'ordine degli argomenti equivale all'ordine in cui sono state presentate le variabili di istanza;
- l'implementazione del metodo esegui della superclasse dovrà essere fatta in modo da eseguire l'istruzione identificata da corpo fintantoché confrontando il valore contenuto nella variabile identificata da variabile con il valore contenuto in valore utilizzando l'operatore indicato da operatore si ottenga un'asserzione logicamente vera (in analogia con il ciclo while del linguaggio Java); il metodo dovrà lanciare l'eccezione `InvalidVariableException` qualora non sia stata precedentemente dichiarata (attraverso l'istruzione `Dichiara`, descritta in seguito) una variabile il cui nome è contenuto in variabile; dovrà inoltre essere lanciata l'eccezione `InvalidOperatorException` nel caso in cui la descrizione dell'operazione non contenga uno dei simboli `>`, `>=`, `<`, `<=`, `==` o `!=`;
- l'implementazione del metodo `toString(x)` della superclasse dovrà restituire la stringa così costruita: $x$ spaxi, la costante `WHILE (`, il nome della variabile, l'operatore, il valore, la costante `)` seguita da un a-capo, l'istruzione `corpo` indentata di $x + 1$.

- `Dichiara`, che descrive la dichiarazione di una variabile. La classe dovrà contenere:
  - la variabile di istanza `String variabile`, che contiene il nome di una variabile;
  - il costruttore `Dichiara(String v)`, che inizializza la variabile di istanza della classe copiando in essa il valore passato come argomento;
  - l'implementazione del metodo `esegui()` della superclasse dovrà essere fatta in modo da inserire nella mappa della superclasse una variabile avente nome uguale a quello contenuto in variabile e inizializzata a 0; il metodo dovrà sollevare l'eccezione `ExistingVariableException` qualora l'argomento variabile identifichi una variabile precedentemente dichiarata;
  - l'implementazione del metodo `toString(x)` della superclasse dovrà restituire la stringa così costruita: $x$ spaxi, la costante `DECLARE` , il nome della variabile, la costante `;` e un a-capo.

- `Assegna`, che descrive l'esecuzione di assegnamento di un valore ad una variabile. La classe dovrà contenere:
  - la variabile di istanza `String variabile`, che identifica il nome di una variabile;
  - la variabile di istanza `int valore`, che identifica il valore intero da assegnare alla variabile;
  - il costruttore `Assegna(String var, int val)`, che inizializza le variabili di istanza della classe copiando in esse i valori passati come argomento;

  - l'implementazione del metodo `esegui()` della superclasse dovrà essere fatta in modo da assegnare il valore alla variabile il cui nome è identificato da `variabile`, cancellandone il contenuto precedente; il metodo dovrà sollevare l'eccezione `InvalidVariableException` qualora l'argomento `variabile` non identifichi una variabile precedentemente dichiarata;
  - l'implementazione del metodo `toString(x)` della superclasse dovrà restituire la stringa così costruita: $x$ spaxi, il nome della variabile, la costante `<-`, il valore, la costante `;` e un a-capo.

- `Incrementa`, che descrive l'incremento del valore memorizzato in una variabile di una preassegnata quantità. La classe dovrà contenere:
  - la variabile di istanza `String variabile`, che identifica il nome di una variabile;
  - la variabile di istanza `int valore`, che identifica il valore intero di cui incrementare la variabile;
  - il costruttore `Incrementa(String var, int val)`, che inizializza le variabili di istanza della classe copiando in esse i valori passati come argomento;
  - l'implementazione del metodo `esegui()` della superclasse dovrà essere fatta in modo da incrementare il valore contenuto nella variabile dell'incremento specificato; il metodo dovrà sollevare l'eccezione `InvalidVariableException` qualora l'argomento `variabile` non identifichi una variabile precedentemente dichiarata;
  - l'implementazione del metodo `toString(x)` della superclasse dovrà restituire la stringa così costruita: $x$ spaxi, il nome della variabile, la costante `^^`, il valore, la costante `;` e un a-capo.

- `Stampa`, che descrive l'output a video di un generico messaggio o del contenuto di una variabile. La classe dovrà contenere:
  - la variabile di istanza `String cosa`, che identifica che cosa visualizzare;
  - la variabile di istanza `boolean isVar` che dice se `cosa` è da intendersi come il nome di una variabile di cui stampare il contenuto oppure se è da intendersi come una stringa costante da visualizzare a video;
  - il costruttore `Stampa(String cosa, boolean isVar)`, che inizializza le variabili di istanza della classe copiando in essa i valori passati come argomenti;
  - l'implementazione del metodo `esegui()` della superclasse dovrà essere fatta in modo da stampare a video la stringa contenuta nella variabile di istanza o il valore della variabile, a seconda dei casi; nel caso si voglia stampare il valore di una variabile ma la variabile non fosse dichiarata, il metodo dovrà sollevare una `InvalidVariableException`;
  - l'implementazione del metodo `toString(x)` della superclasse dovrà restituire la stringa così costruita: $x$ spaxi, la costante `PRINT(`, eventualmente il carattere `"` (solo nel caso di stampa di una stringa), il valore di `cosa`, eventualmente il carattere `"` (solo nel caso di stampa di una stringa), la costante `);` e un a-capo.

- `Leggi`, che descrive l'input di un valore intero in una variabile, preceduto dall'emissione a video di un messaggio. La classe dovrà contenere:
  - la variabile di istanza `String msg`, che identifica il messaggio;
  - la variabile di istanza `String var`, che identifica il nome della variabile;
  - il costruttore `Leggi(String msg, String var)`, che inizializza le variabili di istanza della classe copiando in essa i valori passati come argomenti;
  - l'implementazione del metodo `esegui()` della superclasse dovrà essere fatta in modo da stampare a video il messaggio e, senza andare a capo, di attendere l'inserimento da parte dell'utente di un valore; nel caso che l'utente inserisca un valore non intero il programma deve chiedere di ripetere l'inserimento; dopodiché, il valore inserito deve essere inserito nella variabile;
  - l'implementazione del metodo `toString(x)` della superclasse dovrà restituire la stringa così costruita: $x$ spaxi, la costante `INPUT("`, il messaggio, la costante `")->"`, il nome della variabile, la costante `;` e un a-capo.
- Le classi delle eccezioni, definite in modo opportuno.

A parte quanto espressamente richiesto, è lasciata piena libertà sull'implementazione delle singole classi e sull'eventuale introduzione di classi aggiuntive, a patto di seguire le regole del paradigma ad oggetti ed i principi di buona programmazione. Si suggerisce di porre particolare attenzione alla scelta dei modificatori relativi a variabili d'istanza e metodi, nonché alla dichiarazione e alla gestione delle eccezioni che possono venire lanciate dai vari metodi e alle relazioni di ereditarietà tra le varie classi.

Si suggerisce altresì di porre particolare cura all'implementazione dei metodi `esegui()` e `toString()`, tenendo conto anche di casi in cui l'istruzione da eseguire o da visualizzare contenga in realtà una struttura complessa, costruita utilizzando più volte `Sequenza`, `Selezione` ed `Iterazione`.

Non è richiesto l'utilizzo di particolari modalità grafiche di visualizzazione: è sufficiente una qualunque modalità di visualizzazione basata sull'uso dei caratteri.

È invece espressamente richiesto di non utilizzare package non standard di Java (si possono quindi utilizzare java.util, java.io e così via), con l'unica eccezione package prog.io incluso nel libro di testo per gestire l'input da tastiera e l'output a video.

# 4 Esempio

Non verranno presi in considerazione progetti le cui classi non permetteranno almeno di compilare il seguente sorgente:

```
class Programma {
     try {
         Istruzione dichiaraEAssegna = new Sequenza(
           new Istruzione[] {
             new Dichiara("s"),
             new Dichiara("arg1"),
             new Dichiara("arg2"),
             new Leggi("Inserisci primo valore: ", "arg1"),
             new Leggi("Inserisci secondo valore: ", "arg2"),
                  }
         );
         Istruzione ciclo1 = new Iterazione(
           new Sequenza(
             new Istruzione[] {
               new Incrementa("arg1", -1),
               new Incrementa("s", 1)
                    }
           ), "arg1", "!=", 0 );
         Istruzione ciclo2 = new Iterazione(
           new Sequenza(
             new Istruzione[] {
               new Incrementa("arg2", -1),
               new Incrementa("s", 1)
                    }
           ), "arg2", "!=", 0 );
         Istruzione controllo = new Selezione(
           new Stampa("ed è maggiore di 5", false),
           new Stampa("ed è minore o uguale a 5", false),
           "s",
           ">",
           5
         );
         Istruzione programma1 = new Sequenza(
           new Istruzione[] {
             dichiaraEAssegna,
             ciclo1,
             ciclo2,
             new Stampa("La somma è ", false),
             new Stampa("s", true),
             controllo
                  }
         );
         System.out.println("Sto per eseguire il seguente programma");
         System.out.println(programma1);
         System.out.println("========== OUTPUT ==========");
         programma1.esegui();
         System.out.println("========== FINE OUTPUT ==========");
            } catch(ExecutionException e) {
         e.printStackTrace();
            }
       }
}
```

Ecco un esempio d'esecuzione del programma precedente (il vostro programma, a fronte degli stessi input, deve produrre risultati identici):

```
Sto per eseguire il seguente programma
{
    {
        DECLARE s;
        DECLARE arg1;
        DECLARE arg2;
        INPUT("Inserisci primo valore: ")->arg1;
        INPUT("Inserisci secondo valore: ")->arg2;
    }
    WHILE (arg1!=0)
        {
            arg1^^-1;
            s^^1;
        }
    WHILE (arg2!=0)
        {
            arg2^^-1;
            s^^1;
        }
    PRINT("La somma è ");
    PRINT(s);
    IF (s>5)
        PRINT("ed è maggiore di 5");
    ELSE
        PRINT("ed è minore o uguale a 5");
}
========== OUTPUT ==========
Inserisci primo valore: 5
Inserisci secondo valore: 7
La somma è
12
ed è maggiore di 5
========== FINE OUTPUT ==========
```

# 5 Valutazione

La valutazione del progetto sarà fatta in base alla

- presenza di funzionalità aggiuntive (p.es., possibilità di dichiarare e manipolare variabili di tipo diverso ecc.);
- conformità dell'implementazione scelta per risolvere il problema con il paradigma di programmazione a oggetti;
- conformità del codice presentato alle regole di buona programmazione;
- adeguatezza del manuale utente presentato a descrivere il modo in cui un utente può utilizzare il programma;

- adeguatezza della documentazione e dei commenti;
- assenza di errori nel programma.

## Ringraziamenti

Il progetto è ispirato al testo di un problema simile scritto da Dario Malchiodi.