Sapy: un interprete BASIC

Sebastiano Michele Cossu

Introduzione

Sapy è un interprete per un dialetto del BASIC scritto in Java (JDK 1.7).
Un interprete è un programma che esegue un altro programma descritto in un codice sorgente eseguendolo direttamente nello stesso linguaggio ad alto livello in cui è scritto lo stesso interprete, quindi, senza compilarlo.
L'interpretazione attraversa tre fasi principali:

- Analisi lessicale
- Analisi sintattica
- Interpretazione

Analisi Lessicale

L'analisi lessicale consiste nella lettura, carattere per carattere, di un codice sorgente e nel riconoscimento di simboli conosciuti dall'interprete, i quali saranno raggruppati in parole, dette **token**, organizzati, infine, in una lista e restituita al livello superiore.

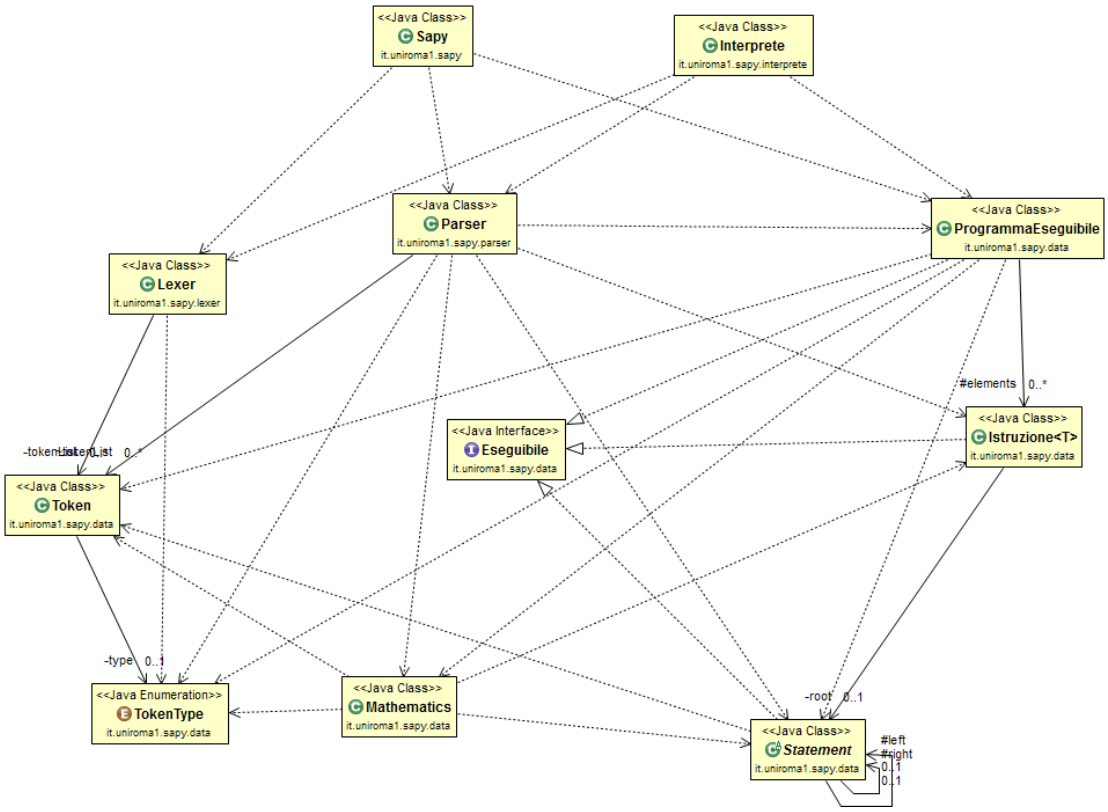
Analisi Sintattica

L'analisi sintattica consiste nell'analisi della lista di token generata dal lexer. Durante quest'analisi, il programma controlla le strutture sintattiche formate dai token e lancia una **SyntaxErrorException** nel caso di strutture malformate.
Al termine dell'analisi, viene prodotto un programma eseguibile che viene passato alla classe che si occuperà di interpretarlo.

Interpretazione

L'interpretazione consiste nell'esecuzione del programma eseguibile generato durante l'analisi sintattica. Il programma eseguirà la lista di istruzioni che contiene al suo interno, ognuna delle quali eseguirà ricorsivamente l'albero sintattico che contiene al suo interno.

Di seguito viene presentato il diagramma UML delle classi principali di Sapy.



Lexer

La classe **Lexer**, è quella che si occupa di eseguire l'analisi lessicale e di costruire la lista di **token** utilizzata nell'analisi sintattica.
Il processo è facilmente rappresentabile con un automa a stati finiti che inizia con la lettura di un carattere e continua con un flusso di controlli dipendente dal tipo di carattere seguente.
In realtà l'analisi lessicale, in questo caso, grazie alle potenzialità di Java, è stata orientata alle espressioni regolari, con un uso selezionato del controllo dei caratteri, indispensabile soltanto per il riconoscimento di stringhe, le quali, a differenza dei lessemi, devono mantenere, al loro interno, caratteri che normalmente sono scartati, come, ad esempio, gli spazi.
Utilizzando la potenza delle espressioni regolari, l'analisi lessicale si è ridotta a qualche controllo sulla struttura delle singole parole divise dagli spazi e a un controllo che registra gli spazi basandosi sulla precedente lettura di una coppia di virgolette non chiuse.
Durante l'analisi lessicale, le parole riconosciute vengono organizzate in token e aggiunte ad una lista che sarà poi utilizzata nell'analisi sintattica.
Un token è una tupla tipo-valore, dove il tipo è un oggetto di tipo enum, mentre il valore è un oggetto che rappresenta l'informazione significativa. Nel caso di un token di tipo variabile è il nome della variabile (o etichetta); nel caso di un numero, è il valore numerico e nel caso di una parola chiave, è la parola chiave stessa. Ogni valore enum ha un intero a esso associato che organizza i vari token in categorie utili

(operatori logici, aritmetici, comparativi, cicli, spaziatori, costanti, ecc..).
La lista dei token, implementata come un **ArrayList<Token>** viene passata al costruttore della classe **Parser**.

Parser

Parser è la classe che si occupa dell'analisi sintattica: il processo responsabile della correttezza della struttura formata dalla lista di token.
Data una grammatica G, in un linguaggio L, la classe parser si occupa di controllare che la lista di token che gli viene sottoposta, rispetti la grammatica G del linguaggio L, ossia, che le strutture sintattiche rappresentate dalla lista di token, siano corrette e coerenti.
Il parser riconosce i token per il loro tipo enum e, una volta riconosciuti dei token chiave, controlla se gli elementi successivi sono quelli attesi. In caso positivo, costruisce l'istruzione rappresentata da quei token come un albero sintattico astratto (AST, Abstract Syntax Tree). L'albero sintattico è contenuto dentro un oggetto della classe Istruzione e i nodi dell'albero sono istanze che estendono la classe astratta Statement. Un esempio di albero astratto può essere costruito su un'istruzione che stampa una comune espressione, come quella che segue:

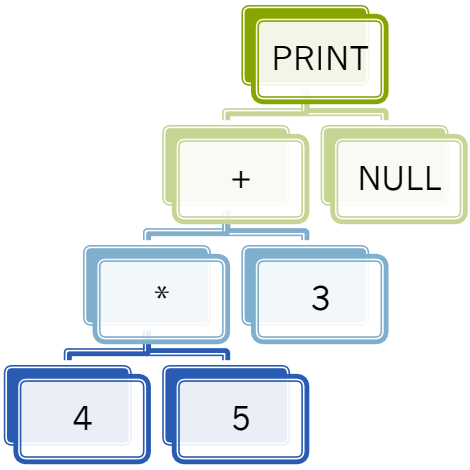
$$PRINT\ 3 + 4 * 5$$

L'istruzione, dopo il lexing, viene analizzata dal parser come la seguente sequenza di token:

$$FUNCTION\ INTEGER\ PLUS\ INTEGER\ MULTIPLIES\ INTEGER\ EOL$$

Il parser, dopo aver riconosciuto il token FUNCTION, legge tutti i token seguenti, fino al token EOL, che indica la fine della linea (End Of Line) e, quindi, dell'istruzione. Tutti i token compresi tra il token funzione e quello di fine riga, vengono presi come parametro e passati al costruttore della classe adatta che estende Statement (in questo caso PrintStatement). L'espressione viene, dunque, organizzata in un AST grazie alla libreria di classi statiche Mathematics che implementa metodi utili come la conversione da notazione postfissa a infissa (passaggio precedente alla conversione in albero), le quattro operazioni principali più il resto, le operazioni logiche, le comparazioni, un metodo di gestione delle precedenze degli operatori e il metodo che costruisce l'AST a partire da una lista di token. Per quanto riguarda il metodo di costruzione dell'AST, ho utilizzato l'algoritmo di risoluzione delle equazioni postfisse per costruire correttamente l'albero binario partendo dalla radice. Poiché il parser è ricorsivo discendente, la radice è l'ultimo nodo ad essere eseguito.

L'AST formato dagli statement e rappresentato dall'istanza di Istruzione, viene aggiunto all'istanza di ProgrammaEseguibile: una classe che contiene una lista di oggetti Istruzione che vengono eseguiti in sequenza e il cui flusso discendente viene alterato soltanto da strutture iterative e cicli quali IF THEN ELSE, FOR e WHILE.
L'AST, dopo il parsing dell'istruzione dell'esempio precedente, è il seguente:



L'albero sintattico binario, dopo essere stato immagazzinato nell'oggetto Istruzione, verrà eseguito ricorsivamente e, l'ultimo statement a essere eseguito (PRINT), rappresenterà la fine dell'istruzione cui seguirà l'avanzamento del program counter (l'indice che punta all'istruzione da eseguire in ProgrammaEseguibile).

Interpretazione

L'interpretazione è tutta implementata in ProgrammaEseguibile e viene avviata dalla classe Sapy o dalla classe Interprete, poiché entrambe contengono un main che permette di passare via argomento un file da interpretare.
ProgrammaEseguibile contiene, come attributi, quattro variabili di controllo e due ArrayList.
La memoria, nella classe ProgrammaEseguibile, è simulata con due ArrayList: il primo contiene la lista dei nomi delle variabili definite, l'altro contiene la lista dei loro valori. Gli indici dei vari elementi dei due ArrayList fungono da identificatore della posizione in memoria, ossia, simulano un indirizzo in memoria. La ricerca del valore di una variabile, infatti, è gestita cercando il nome della variabile nel primo ArrayList e restituendo l'indice, il quale sarà utilizzato per prelevare il valore della variabile dal secondo ArrayList. La scrittura della memoria, viene effettuata simmetricamente sui due ArrayList, sia per quanto riguarda l'addizione, che per quanto riguarda la sottrazione di variabili.

Le quattro variabili di controllo permettono il controllo delle strutture iterative annidate e sono: openIf, openElse, openFor e openWhile.
Ogni volta che l'espressione di una IF è verificata, viene incrementato openIf; ogni volta che viene incontrato un ELSE, se l'openIf è maggiore di zero, viene decrementato di una unità e incrementato il program counter finché non si incontra un ENDIF. Se ELSE è assente, quando si incontra un ENDIF, viene semplicemente decrementato openIf e incrementato di una unità il program counter per passare all'istruzione successiva.
Un esempio di IF in Sapy è:

```
IF $x = $w & $y = $h THEN
   PRINT "La figura è un punto"
ELSE
   PRINT "La figura non è un punto"
ENDIF
```

Altri alteratori del flusso del programma sono i cicli FOR e WHILE.
Il ciclo FOR è niente più che un contatore. Definito un valore iniziale e un valore finale, itera le istruzioni comprese tra DO e NEXT tante volte quante sono necessarie per raggiungere il valore finale. In aggiunta, al ciclo FOR, è possibile specificare un incremento con la parola chiave STEP. Se non viene specificato il valore associato a STEP, viene utilizzato l'incremento base di una unità.
Un esempio del ciclo FOR in Sapy è:

```
FOR $a = 0 TO 10 STEP 2 DO
   PRINT $a
NEXT
```

Il ciclo WHILE, invece, ripete la lista di istruzioni comprese tra se DO e LOOP, finché è vera l'espressione contenuta al suo interno (definita tra WHILE e DO).
Un esempio del ciclo WHILE in Sapy è:

```
WHILE $a < $b DO
   PRINT $a
   $a = $a + 1
LOOP
```

L'assegnazione delle variabili, insieme alle strutture di controllo, ha bisogno di una particolare attenzione a livello di esecuzione, poiché deve poter comunicare con la memoria che è accessibile soltanto da ProgrammaEseguibile e non dai singoli statement all'interno degli alberi. Ogni statement, al suo interno, dunque, contiene dei metodi per informare il ProgrammaEseguibile del possesso di variabili o meno. Le variabili sono contenute in uno dei figli di uno statement all'interno dell'istruzione in un'istanza della classe VarexStatement (VARiable

EXpression Statement), la quale, al suo interno, contiene un ArrayList rappresentante l'espressione di variabili tokenizzata (quindi ancora primitiva). L'espressione viene passata dall'istruzione tramite il metodo getVars() al ProgrammaEseguibile, che sostituisce i token delle variabili con i token dei valori trovati in memoria corrispondenti a quell'etichetta. Se ProgrammaEseguibile, non trova l'etichetta specificata, nel caso di un'istruzione diversa da un'assegnazione, restituisce un errore SyntaxErrorException, poiché si fa riferimento ad una variabile che non è mai stata definita prima.

Conclusione

Sapy è stato scritto utilizzando soltanto librerie standard di Java (JDK 1.7), in particolare dei package java.lang, java.util e java.io.
Il linguaggio è piuttosto semplice, poiché implementa:

- Strutture basilari come IF-THEN-ELSE-ENDIF, FOR-TO-STEP-DO-NEXT e WHILE-DO-LOOP;
- Variabili ibride stringa/numerico (i numeri sono implementati come double, per dare maggiore flessibilità e libertà, altrimenti non sarebbero possibili calcoli basilari quali l'area del cerchio, ad esempio);
- Gestione di espressioni aritmetiche, logiche e di comparazioni in modo misto (la precedenza è: aritmetica, comparazione, logica) sia per quanto riguarda le assegnazioni, che per quanto riguarda le strutture di controllo IF e WHILE;
- Gestione di strutture di controllo annidate;
- Una funzione PRINT, che stampa a video il valore di una variabile, il risultato di un'espressione o una stringa;
- Una funzione INPUT che prende in input (da tastiera) una sequenza e la salva in una variabile definita dal programmatore.

Nonostante la semplicità del linguaggio, Sapy ha le potenzialità di poter essere espanso e arricchito in modo da poter gestire task particolari in modo semplice e veloce e operando a prescindere dal sistema operativo, grazie alla flessibilità del linguaggio Java.