TKOM

Język do zarządzania planami zajęć stworzony do współpracy z programem

tworzącym plany zajęć

Bartłomiej Gajda

Krótki opis wstępny:

W ramach projektu na przedmiot techniki kompilacji stworzony został język, który wspomaga przeglądanie i zarządzanie plikami formatu txt zawierającymi plany zajęć (pliki mają określony format wewnętrzny, każdy plan zajęć składa się właściwie z dwóch plików: pierwszy jest to plik z danymi – zawierający informacje, które mają znaleźć się w planie np. nazwiska nauczycieli, ilość klas; drugi natomiast zawiera sam plan zajęć, który możliwy jest do odczytania w oparciu o pierwszy plik z danymi oraz zawiera również wartość funkcji oceny planu. Plik z danymi i plik z planem posiadają odpowiadające sobie nazwy – plik z danymi: CiagZnakow.txt, plik z planem: CiagZnakow1.txt, CiagZnakow2.txt itd. Język umożliwia iterowanie po planach zajęć znajdujących się w jednym folderze i uzyskiwać konkretne informacje, którymi mogą być wartość funkcji oceny planu czy ilości sal dostępnych w danej szkole. Projekt został stworzony przy pomocy języka JAVA. Dopuszczalne instrukcje to deklaracja zmiennych, przypisanie do zmiennych, instrukcja warunkowa, pętla while, pętla forEach stworzona specjalnie do iterowania po planach zajęć, wywołania funkcji.

Typy zmiennych:

* TimetableData – zmienna reprezentująca dane do planu zajęć
* Timetable – zmienna reprezentująca pojedynczy plik z planem zajęć stworzony na podstawie danych z pliku danych do plany zajęć
* Integer – zmienna liczbowa do określania wartości funkcji oceny itp.
* String – zmienna reprezentująca łańcuch znakowy

Opis gramatyki:

**grammar** tt;  
  
script  
 : (statement)\*;  
  
instructions  
 : (LEFT\_CURLY\_BRACKET statement+ RIGHT\_CURLY\_BRACKET) | statement;  
  
condition  
 : LEFT\_ROUND\_BRACKET alternative RIGHT\_ROUND\_BRACKET;  
  
alternative  
 : conjunction ( OR conjunction )\*;  
  
conjunction  
 : comparison ( AND comparison )\*;  
  
comparison  
 : (VARIABLE\_NAME | INTEGER | return\_statement | mathExpression )  
 COMPARISON (VARIABLE\_NAME | INTEGER | return\_statement | mathExpression );  
  
mathExpression  
 : LEFT\_ROUND\_BRACKET (VARIABLE\_NAME | INTEGER | return\_statement | mathExpression)  
 (PLUS | MINUS | MULTIPLY | DIVIDE) (VARIABLE\_NAME | INTEGER | return\_statement | mathExpression) RIGHT\_ROUND\_BRACKET;  
  
statement  
 : if\_statement  
 | while\_statement  
 | for\_loop  
 | declaration\_statement SEMICOLON  
 | assignement\_statement SEMICOLON  
 | timetable\_statements SEMICOLON  
 | print\_statement SEMICOLON  
 | SEMICOLON;  
  
if\_statement  
 : IF condition instructions (ELSE instructions)?;  
  
while\_statement  
 : WHILE condition instructions;  
  
for\_loop  
 : FOREACH LEFT\_ROUND\_BRACKET timetable\_type VARIABLE\_NAME COLON VARIABLE\_NAME DOT list\_tt\_statement RIGHT\_ROUND\_BRACKET instructions;  
  
declaration\_statement  
 : type\_declaration  
 | string\_declaration  
 | int\_declaration  
 | timetable\_declaration  
 | timetable\_data\_declaration;  
  
assignement\_statement  
 : type\_assignment  
 | string\_assignment  
 | int\_assignment  
 | timetable\_assignment  
 | timetable\_data\_assignment;  
  
timetable\_statements  
 : gen\_timetable  
 | print\_tt\_statistics  
 | generate\_PDF  
 | print\_tt\_data\_statistics;  
  
print\_statement  
 : PRINT LEFT\_ROUND\_BRACKET print\_types (PLUS print\_types)\* RIGHT\_ROUND\_BRACKET;  
  
print\_tt\_statistics  
 : VARIABLE\_NAME DOT PRINT\_TT\_STATS LEFT\_ROUND\_BRACKET RIGHT\_ROUND\_BRACKET;  
  
print\_tt\_data\_statistics  
 : VARIABLE\_NAME DOT PRINT\_TT\_DATA\_STATS LEFT\_ROUND\_BRACKET RIGHT\_ROUND\_BRACKET;  
  
generate\_PDF  
 : VARIABLE\_NAME DOT GEN\_PDF LEFT\_ROUND\_BRACKET RIGHT\_ROUND\_BRACKET;  
  
list\_tt\_statement  
 : LIST\_TIMETABLE LEFT\_ROUND\_BRACKET RIGHT\_ROUND\_BRACKET;  
  
open\_timetable\_data  
 : OPEN\_TT\_DATA LEFT\_ROUND\_BRACKET STRING RIGHT\_ROUND\_BRACKET;  
  
gen\_timetable  
 : VARIABLE\_NAME DOT GEN\_TIMETABLE LEFT\_ROUND\_BRACKET (INTEGER | INTEGER COMMA INTEGER) (COMMA INTEGER)? RIGHT\_ROUND\_BRACKET;  
  
return\_statement  
 : get\_fitness | get\_classroom\_count;  
  
get\_fitness  
 : VARIABLE\_NAME DOT GET\_FITNESS LEFT\_ROUND\_BRACKET RIGHT\_ROUND\_BRACKET;  
  
get\_classroom\_count  
 : VARIABLE\_NAME DOT GET\_CLASSROOMS LEFT\_ROUND\_BRACKET RIGHT\_ROUND\_BRACKET;  
  
*//declarations*type\_declaration  
 : type VARIABLE\_NAME (ASSIGNMENT VARIABLE\_NAME)?;  
  
string\_declaration  
 : string\_type VARIABLE\_NAME ASSIGNMENT STRING;  
  
int\_declaration  
 : int\_type VARIABLE\_NAME ASSIGNMENT (mathExpression | return\_statement | INTEGER);  
  
timetable\_data\_declaration  
 : timetable\_data\_type VARIABLE\_NAME ASSIGNMENT open\_timetable\_data;  
  
timetable\_declaration  
 : timetable\_type VARIABLE\_NAME ASSIGNMENT gen\_timetable;  
*//declarations end  
  
//assignments*type\_assignment  
 : VARIABLE\_NAME ASSIGNMENT VARIABLE\_NAME;  
  
string\_assignment  
 : VARIABLE\_NAME ASSIGNMENT STRING;  
  
int\_assignment  
 : VARIABLE\_NAME ASSIGNMENT (mathExpression | return\_statement | INTEGER);  
  
timetable\_data\_assignment  
 : VARIABLE\_NAME ASSIGNMENT open\_timetable\_data;  
  
timetable\_assignment  
 : VARIABLE\_NAME ASSIGNMENT gen\_timetable;  
*//assignments end*print\_types  
 : VARIABLE\_NAME | return\_statement | STRING;  
  
type  
 : timetable\_type | string\_type | int\_type | timetable\_data\_type;  
  
timetable\_type  
 : TIMETABLE\_TYPE;  
  
timetable\_data\_type  
 : TIMETABLE\_DATA\_TYPE;  
  
int\_type  
 : INTEGER\_TYPE;  
  
string\_type  
 : STRING\_TYPE;  
  
INTEGER\_TYPE: 'Integer';  
STRING\_TYPE: 'String';  
TIMETABLE\_TYPE: 'Timetable';  
TIMETABLE\_DATA\_TYPE: 'TimetableData';  
LEFT\_ROUND\_BRACKET: '(';  
RIGHT\_ROUND\_BRACKET: ')';  
LEFT\_CURLY\_BRACKET: '{';  
RIGHT\_CURLY\_BRACKET: '}';  
DOT: '.';  
  
PRINT: 'print';  
LIST\_TIMETABLE: 'listAll';  
OPEN\_TT\_DATA: 'openTTData';  
GEN\_TIMETABLE: 'genTimetable';  
PRINT\_TT\_STATS: 'printStats';  
PRINT\_TT\_DATA\_STATS: 'printDataStats';  
GEN\_PDF: 'genPDF';  
GET\_FITNESS: 'getFitness';  
GET\_CLASSROOMS: 'getClassroomCount';  
  
SEMICOLON: ';';  
IF: 'if';  
ELSE: 'else';  
FOREACH: 'forEach';  
WHILE: 'while';  
COLON: ':';  
ASSIGNMENT: '=';  
OR: 'or';  
AND: 'and';  
COMPARISON: '==' | '!=' | '<' | '>' | '<=' | '>=';  
PLUS: '+';  
MINUS: '-';  
MULTIPLY: '\*';  
DIVIDE: '/';  
VARIABLE\_NAME: [a-zA-Z]([a-zA-Z0-9])\*;  
INTEGER: ('0'|[1-9][0-9]\*);  
STRING: '"'(~('"') | ('\\' ('"')))\*'"';  
COMMA: ',';  
COMMENT : '#' ~[\n\r]\* -> skip;  
WHITESPACE : (' ' | [\n\t\r]) -> skip ;

Zrealizowane wymagania:

-możliwość utworzenia drzewa rozbioru oraz wykonania skryptu,

-możliwość przejścia po cały drzewie rozbioru oraz poprawnym wykonaniu skryptu,

-przy tworzeniu drzewa wykonywana jest analiza leksykalna i składniowa. Błędy obsługiwane są przez obsługę wyjątków. Poprawność rzucania została sprawdzona dzięki implementacji testów jednostkowych,

- możliwość definiowania własnych zmiennych, różnych typów Timetable i TimetableData do obsługi plików związanych z planami zajęć, wykonywanie na nich różnych operacji np. otrzymanie wartości funkcji przystosowania lub ilości pomieszczeń w danej szkole. Możliwe są również operacje na liczbach całkowitych – różne operacje arytmetyczne, zmienne każdego typu można wyświetlać korzystając z metody print,

- możliwość tworzenia instrukcji warunkowych, instrukcji warunkowych z wieloma warunkami pomocniczymi, własnych pętli oraz specjalnego typu pętli do iterowania bo planach zajęć. Dużą zaletą jest możliwość zagnieżdżania zarówno instrukcji warunkowych jak i pętli.

- możliwość tworzenia wewnątrz instrukcji warunkowej if złożonych wyrażeń logicznych.

- dzięki stworzeniu klas wyjątków oraz ustaleniu sytuacji występowania błędów w łatwy sposób można określić miejsce oraz typ błędu.

Architektura:

Interpreter został napisany w języku Java z wykorzystaniem narzędzia ANTLR. ANTLR posłużył do wygenerowania podstawowych klas oraz interfejsów lexera oraz parsera. Lexer oraz parser wygenerowane zostały na podstawie gramatyki załączonej powyżej. Logika interpretera została przeze mnie zaimplementowana w klasie TTVisitor, która implementuje interfejs Visitor wygenerowany przez plugin ANTLR. Korzystając z tego interfejsu możliwe jest przejście po drzewie wyprowadzenia oraz wykonanie wszystkich instrukcji znajdujących się w danym przykładowym skrypcie. Parser działa w ten sposób, że przechodzi pomiędzy węzłami drzewa wyprowadzenia w sposób zdefiniowany w klasie TTVisitor. Dzięki temu można wykonywać wyrażenia w pętli oraz sprawdzanie za każdym razem czy warunki są nadal spełnione i czy nie powinien zostać rzucony wyjątek. Parser tworzy mapę, w której przechowywane są wszystkie zmienne. Kluczem jest unikalna nazwa zmiennej, wartością natomiast jest instancja specjalnie stworzonej klasy Variable, która zawiera typ zmiennej, unikatową nazwę oraz wartość zmiennej odpowiednia dla danego typu. W momencie zadeklarowania lub zdefiniowania zmiennej, umieszczana jest ona w mapie. Dzięki wykorzystaniu tego kontenera oraz stworzeniu klasy Variable sprawdzane jest, czy na przykład zmienna nie została już ponownie zadeklarowana albo czy nie próbujemy odwoływać się do zmiennej, która nie istnieje lub która jest innego niż wymagany typ. Przy przechodzeniu po drzewie wyprowadzenia i odwiedzaniu kolejnych węzłów w momencie, gdy ma być wykonana jakaś operacja, program odwołuje się do metod zdefiniowanych w klasie TTFunctions, która zawiera implementacje wszystkich funkcji, które mają już ścisły związek z implementacją programu generującego plany zajęć. Dzięki temu zabiegowi, w łatwy sposób, nie ingerując już w parser, lekser lub gramatykę można zmienić implementację na przykład generowania pliku pdf z pliku Timetable.

Przykładowy skrypt 1:

TimetableData = openTTData("ax123");  
Integer fitness = 25;  
Timetable bestTimetable;  
forEach(Timetable t: timetableData.listAll())  
 {  
 if(t.getFitness()> fitness)  
 t.genPDF();  
 print("Fitness: " + t.getFitness());  
 }  
#generating timetables if fitness is greater than 25

Przykładowy skrypt 2:

#program used to generate timetables with given parameters  
#also printing stats  
TimetableData ttdata = openTTData("qt53V");  
print("TimetableData file opened. Number od classrooms: " + ttdata.getClassroomCount());  
Integer i=10;  
Timetable temp;  
while(i>=0)  
{  
 temp = ttdata.genTimetable(2);  
 temp.printStats();  
 i=(i-1);  
}

Testowanie:

Testowania wymagała gramatyka oraz implementacja metod w klasie TTVisitor, która dziedziczyła po interfejsie Visitor. Aby zapewnić odpowiednie testowanie, stworzona została hierarchia dodatkowych typów wyjątków, które rzucane są w przypadku wystąpienia błędów np. składniowych. Rzucane są również w przypadku innych błędów, np. gdy próbujemy przypisać zmienną do nieodpowiedniego typu. Testowanie odbywa się z pomocą narzędzia JUnit 4, dla którego został stworzony specjalny moduł testów jednostkowych. Testowane są wszystkie rodzaje błędów, które mogą wystąpić w programie. Program przechodzi wszystkie testy w 100%. Testowaniu podlegają na przykład: niewłaściwy sposób deklarowania zmiennych, niewłaściwie przypisywanie wartości, złe użycie pętli forEach lub while, niepoprawne zdania logiczne wewnątrz instrukcji if.

Uruchamianie:

Aby uruchomić dany skrypt napisany w pliku .txt należy ścieżkę pliku wprowadzić w funkcji main programu. W celach demonstracyjnych stworzone zostały przykłady czterech skryptów prezentujących różne funkcjonalności. Aby lepiej zobrazować, które operacje wykonuje skrypt, po każdej wyświetlana jest informacja co zostało zrobione. W programie został stworzony dodatkowy tryb uruchamiania skryptu, który generuje i wyświetla na ekranie drzewo rozbioru skryptu. Dzięki temu trybowi można w prosty sposób prześledzić, jak skrypt jest „rozkładany” korzystając ze stworzonej gramatyki.