Entrega Etapa 2: Analizador Sintáctico

Alumno : Javier Amorosi
LU: 94528

Índice:

Transformación de la gramática :

Gramática BNF : obteniendo una gramática bnf

Se eliminan los símbolos de la gramática BNF extendida.

1. <Inicial> -> <ListaClases>
2. <ListaClases> -> <Clase> <ListaClases> | <Clase>
3. <Clase> -> class idClase <Herencia> {<Miembro Aux>}
4. <Herencia> -> extends idClase | e
5. <ListaMiembros> -> <Miembro> <ListaMiembros> | e
6. <Miembro> -> <Atributo>| <Ctor> | <Método>
7. <Atributo> -> <Visibilidad> <Tipo> <ListaDecAtrs>;
8. <Método> -> <Forma Método> <Tipo Método> idMetVar <ArgsFormales> <Bloque>
9. <Ctor> → idClase <ArgsFormales> <Bloque>
10. <Visibilidad> -> public | private
11. <Tipo> -> <TipoPrimitivo>| idClase
12. <TipoPrimitivo> -> boolean | char | int | string
13. <ListaDecAtrs> -> idMetVar
14. <ListaDecAtrs> -> idMetVar , <ListaDecAtrs>
15. <FormaMetodo> -> static|dynamic
16. <TipoMetodo>-> <Tipo>| void
17. <ArgsFormales> -> (<ListaArgsFormalesAux>)
18. <ListaArgsFormalesAux>-> <ListaArgsFormales>|e
19. <ListaArgsFormales> -> <ArgFormal>
20. <ListaArgsFormales> -> <ArgFormal>, <ListaArgsFormales>
21. <ArgFormal> -> <Tipo> idMetVar
22. <Bloque>->{<ListaSentencias>}
23. <Sentencia> -> ; | <Asignación> ; | <Llamada>;|<Tipo><ListaDecVars>;
24. <Sentencia> -> if(<Expresion>)<Sentencia>
25. <Sentencia> -> if(<Expresion>)<Sentencia> else <Sentencia>
26. <Sentencia> -> while(<Expresion>)<Sentencia>
27. <Sentencia> -> <Bloque>
28. <Sentencia> -> return <ExpresionAux>;
29. <Asignacion> -> <Acceso><TipoAsignacion><Expresion>

30.<TipoAsignacion>-> ==| +=|-=
31.<Llamada> -> <Acceso>
32. <ListaDecVariables> -> idMetVar
33. <ListaDecVariables> -> idMetVar , <ListaDecVariables>
34. <Expresion>-> <Expresion><OperadorBinario><ExpresionUnaria>
35.<Expresion>-><ExpresionUnaria>
36. <OperadorBinario> -> = || | && | == | != | < | > | <= | >= | + | - | * | / | %
37.<ExpresionUnaria> -> <OperadorUnario><Operando>
38.<ExpresionUnaria>-><Operando>
39. <OpUn> -> +|-|!
40.<Operando> -> <Literal>
41. <Literal> -> null | true | false | intLiteral | charLiteral | stringLiteral
42.<Acceso> -><Primario><Encadenado>
43.<Primario> -> <AccesoThis>
44.<Primario> -> <AccesoVar>
45.<Primario> -> <AccesoEstatio>
46.<Primario> -> <AccesoConstructor>
47.<Primario> -> <AccesoMetodo>
48. <Primario> -> (<Expresion>)
49. <AccesoThis> -> this
50. <AccesoVar> -> idMetVar
51. <AccesoEstatico> -> idClase . <AccseoMetodo>
52. <AccesoConstructor> -> new idClase <ArgsActuales>
54. <AccesoMetodo> -> idMetVar<ArgsActuales>
55. <ArgsActuales> -> (<ListaExpAux>)
56. <ListaExpAux-> <ListaExps>|e
57. <ListaExps> -> <Expresion> | <Expresion>,<ListaExps>
58. <Encadenado> -> <VaroMetodoEncadenado> <Encadenado> | e
59. <VarOMetodoEncadenado> -> <VarEncadenada>
60.<VarOMetodoEcadenado> -> <MetodoEncadenado>
61.<VarEncadenado> -> . idMetVar
62.<MetodoEncadenado>-> . idMetVar <ArgsActuales>

Eliminar recursión a Izquierda:

- Se identifican primero cuales son las reglas de la gramática que tienen recursión a izquierda.

34. <Expresion>-> <Expresion><OperadorBinario><ExpresionUnaria>

Se reemplaza la producción por la siguiente regla

34.<Expresion>-> <ExpUnaria><ExpAux>

34.b <ExpAux> -> <OperadorBinario><ExpUnaria><ExpAux> | e

Factorización de la gramática :

Procedemos a identificar las producciones a ser factorizadas.

32. <ListaDecVariables> -> idMetVar
33. <ListaDecVariables> -> idMetVar , <ListaDecVariables>
24. <Sentencia> -> if(<Expresion>)<Sentencia>
25. <Sentencia> -> if(<Expresion>)<Sentencia> else <Sentencia>
26.<Sentencia> -> <Asignación> ; | <Llamada>;|
44.<Primario> -> <AccesoVar>
47.<Primario> -> <AccesoMetodo>
59. <VarOMetodoEncadenado> -> <VarEncadenada>
60.<VarOMetodoEcadenado> -> <MetodoEncadenado>

Factorización :

32.<ListaDecVariables>->idMetVar<ListaDecVariablesAux>
32b.<ListaDecVariablesAux>->,<ListaDecVariables> | e
24.<Sentencia> -> if(<Expresion>)<Sentencia> <SentenciaElse>
24b.<SentenciaElse>-> else <Sentencia> | e
26.<Sentencia> -> <AsignaciónLlamada>
26b.<AsignacionLlamada>-> <Acceso><TipoAsignacion><Expresion>| <Acceso>
44.<Primario> -> <AccesoVarMetodo>
44b.<AccesoVarMetodo>->idMetVar | idMetVar <ArgsActuales>
59.<VarOMetodoEncadenado>-> . idMetVar | . idMetVar <ArgsActuales>

Gramática Resultante :

1. <Inicial> -> <ListaClases>
2. <ListaClases> -> <Clase> <ListaClases> | <Clase>
3. <Clase> -> class idClase <Herencia> {<Miembro Aux>}
4. <Herencia> -> extends idClase | e
5. <ListaMiembros> -> <Miembro> <ListaMiembros> | e
6. <Miembro> -> <Atributo>| <Ctor> | <Método>
7. <Atributo> -> <Visibilidad> <Tipo> <ListaDecAtrs>;
8. <Método> -> <Forma Método> <Tipo Método> idMetVar <ArgsFormales> <Bloque>
9. <Ctor> → idClase <ArgsFormales> <Bloque>
10. <Visibilidad> -> public | private
11. <Tipo> -> <TipoPrimitivo>| idClase
12. <TipoPrimitivo> -> boolean | char | int | string
13. <ListaDecAtrs> -> idMetVar <ListaDecAtrsF>
14. <ListaDecAtrsF> -> , <ListaDecAtrs>
15. <FormaMetodo> -> static|dynamic
16. <TipoMetodo>-> <Tipo>| void
17. <ArgsFormales> -> (<ListaArgsFormalesAux>)
18. <ListaArgsFormalesAux>-> <ListaArgsFormales>|e
19. <ListaArgsFormales> -> <ArgFormal>
20. <ListaArgsFormales> -> <ArgFormal>, <ListaArgsFormales>
21. <ArgFormal> -> <Tipo> idMetVar
22. <Bloque>->{<SentenciaAux>}
23. <Sentencia> -> ; | <AsignaciónLlamada>;|<Tipo><ListaDecVars>;

23b.<AsignacionLlamada>-> <Acceso><TipoAsignacion><Expresion>| <Acceso>

24.<Sentencia> -> if(<Expresion>)<Sentencia> <SentenciaElse>
25.<SentenciaElse>-> else <Sentencia> | e
26. <Sentencia> -> while(<Expresion>)<Sentencia>
27. <Sentencia> -> <Bloque>
28. <Sentencia> -> return <ExpresionAux>;
29.//<AsignacionLlamada> -> <Acceso><TipoAsignacion><Expresion> | <Acceso>
30.<TipoAsignacion>-> ==| +=|-=
31.//<Llamada> -> <Acceso>
32.<ListaDecVariables>->idMetVar<ListaDecVariablesAux>
33.<ListaDecVariablesAux>->,<ListaDecVariables> | e
34.<Expresion>-> <ExpUnaria><ExpAux>
35 <ExpAux> -> <OperadorBinario><ExpUnaria><ExpAux> | e
36. <OperadorBinario> -> = || | && | == | != | < | > | <= | >= | + | - | * | / | %
37.<ExpresionUnaria> -> <OperadorUnario><Operando>
38.<ExpresionUnaria>-><Operando>
39. <OpUn> -> +|-|!
40.<Operando> -> <Literal>

41. <Literal> -> null | true | false | intLiteral | charLiteral | stringLiteral
42.<Acceso> -><Primario><Encadenado>
43.<Primario> -> <AccesoThis>
44.<Primario> -> <AccesoVarMetodo>
45.<Primario> -> <AccesoEstatio>
46.<Primario> -> <AccesoConstructor>
47.<AccesoVarMetodo>->idMetVar | idMetVar <ArgsActuales>
48. <Primario> -> (<Expresion>)
49. <AccesoThis> -> this
51. <AccesoEstatico> -> idClase . <AccseoMetodo>
52. <AccesoConstructor> -> new idClase <ArgsActuales>
55. <ArgsActuales> -> (<ListaExpAux>)
56. <ListaExpAux-> <ListaExps>|e
57. <ListaExps> -> <Expresion> | <Expresion>,<ListaExps>
58. <Encadenado> -> <VaroMetodoEncadenado> <Encadenado> | e
59.<VarOMetodoEncadenado>-> . idMetVar | . idMetVar <ArgsActuales>

Errores detectados :

Cada uno de los siguientes errores arrojará una excepción e imprimirá un mensaje por consola para luego detener la ejecución del analizador sintáctico.

1. Regla 1 : lo que sigue a <ClaseAux> no es EOF.
2. Regla 3:
    a. si no se encuentra la palabra clave class.
    b. luego no encuentra un identificador de clase.
    c. si no encuentra { luego de llamar a herencia.
    d. si no encuentra } al finalizar.

3. Regla 4: Luego del extends no encuentra un idClase.
4. regla 7: no encuentra un ; al final .
5. regla 8: no encuentra un idMetVar antes de llamar a args formales.
6. regla 10: no encuentra un idClase al principio.
7. regla 11: no encuentra un idMetVar al principio.
8. regla 13: si no encuentra (
9. .
10. paréntesis que abren y luego no encuentra) paréntesis que cierra al final. 11. regla 17:si no encuentra idMetVar al final de la producción.
12. regla 18: no encuentra static o dynamic.
13. regla 19: no encuentra public protected o private.
14. regla 22: no encuentra int char boolean o string.
15. regla 23: no encuentra un idMetVar.
16. regla 25: no encuentra { y luego no encuentra } al final.
17. reglas 27,28,31,33,34: son las reglas que agrupan <Sentencia>, si al final de las reglas que requieren ; no lo encuentra, entonces reporta el error.
18. regla 29: luego del if no encuentra (, luego de expresión no encuentra). 19. regla 32: luego del while no encuentra (, luego de expresión no encuentra). 20. regla 36: no se encuentra un = .
21. regla 37: no encuentra idMetVar.
22. regla 39: no encuentra .(punto) y luego del punto no encuentra un idMetVar, ambos errores son reportados.
23. regla 40: no encuentra (antes de la expresión, o no encuentra) luego. 24. regla 55: no encuentra == o !=.
25. regla 56 : no encuentra ninguno de los siguientes símbolos >, <, >=, <=.
26. regla 57: no entra +(más) o -(menos).
27. regla 58: no encuentra +(más) o -(menos) o !(not).
28. regla 59: no encuentra *,/ o %.
29. regla 63: no encuentra TRUE FALSE NUM NULL charLiteral stringLiteral.
30. regla 64,65,66:
    a. si encuentra (y luego no encuentra) reporta error.
    b. si no encuentra this,idClase,idMetVar o new reporta error.
31. regla 67:no encuentra idMetVar.
32. regla 68: no encuentra this.
33. regla 69: no encuentra idClase, o luego no encuentra '.'(punto). 34. regla 70: no encuentra new, o luego de new no encuentra idClase. 35. regla 72: no encuentra '.', luego del punto no encuentra un idMetVar.
36. regla 74: no encuentra un idMetVar al principio.
37. regla 75: no encuentra (paréntesis que abren al final, no encuentra luego) paréntesis que cierra al final.

Decisiones de Diseño:

- Las clases que se agregaron en esta etapa fueron:

    - Asintactico y AsintacticoException, en el paquete Sintaxis, que corresponden a la clase del analizador sintáctico y su excepción.
- En cuanto a los errores : Muchos errores son atajados previamente por otras reglas, pero por una cuestión de robustez, y a la vez simplicidad, se optó por preservar la excepción arrojada aunque tal excepción nunca llegue a producirse.
- Cada regla de producción de la gramática resultante está programada en la clase Asintactico, cada método tiene asociado un comentario de la regla correspondiente, lo que facilita su testeo.
- Se hacen uso de casos de test positivos y erróneos. Los casos de test positivos refieren los que el analizador sintáctico termina su análisis sin errores. Los casos de prueba erróneos tienen el error asociado al título. Ambos se encuentran en la carpeta casos de prueba.

Logros:

- Imbatibilidad sintáctica.