Autarquia Educacional do Vale do São Francisco – AEVSF **Faculdade de Ciências Aplicadas e Sociais de Petrolina – FACAPE**	
CURSO: Ciência da Computação	**DISCIPLINA: Compiladores**
PROFESSOR: Sérgio F. Ribeiro	
PROJETO Nº 2	

Especificação Sintática de um Subconjunto da Linguagem Java

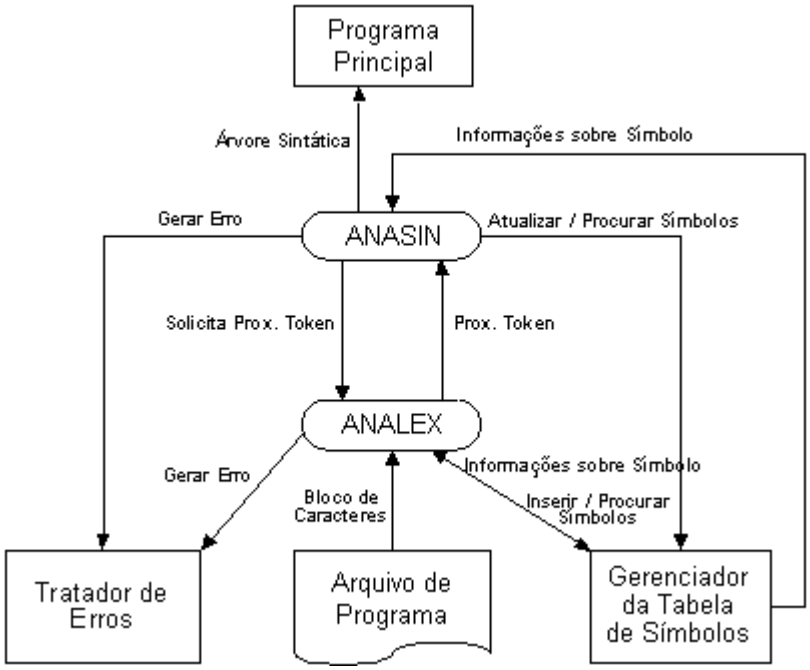
Objetivo: Projetar e Desenvolver a fase de Análise Sintática de um Compilador para a linguagem de programação Java.

Observações:

1. O Projeto poderá ser implementado utilizando-se uma das seguintes linguagens: Pascal, Object Pascal (Delphi), Java, C ou C++.
2. A avaliação deste projeto terá peso cinco (5) na geração da 2ª Nota desta disciplina.
3. Deverão ser entregues junto com o Projeto:
    a. Código-Fonte e Executável completo e documentando do projeto;
    b. Documentação do Projeto (Descrição, Membros da Equipe, Instruções de Uso, Simplificações nas Gramáticas, Exemplos de Teste, etc);
4. **Importante:** este projeto exige como pré-requisito obrigatório o desenvolvimento do 1º Projeto. A equipe que não implementou o 1º Projeto não poderá implementar este Projeto e, portanto, ficará com ZERO na nota da 2ª Avaliação. Após a correção do 1º Projeto, a equipe deverá corrigir as falhas indicadas para poder implementar este 2º Projeto sem erros. Serão de responsabilidade da equipe as falhas ocorridas neste 2º Projeto em função de erros não corrigidos no 1º Projeto.

Análise Sintática

O Analisador Sintático (ANASIN) deverá avaliar a sintaxe do programa-fonte em linguagem Java fornecendo os eventuais erros sintáticos encontrados. Se não encontrar erros sintáticos, o programa deverá construir uma árvore sintática correspondente ao programa analisado. A estrutura deve ser a seguinte:



Os erros sintáticos identificados devem possuir mensagens precisas. Por exemplo:

- '**;**' esperado na linha nn
- ')' esperado na linha nn, etc.

O programa principal deve exibir em tela ou arquivo a estrutura da árvore sintática em alguma forma clara que permita a avaliação da correção dos resultados apresentados pelo programa.

Critérios de Avaliação

O Projeto será avaliado de acordo com os seguintes critérios:

2ª Fase – ANASIN – **Valor: 5,0 (cinco pontos)**

- Reconhecimento Sintático: se o programa reconhece as estruturas sintáticas especificadas corretamente. **Valor: 2,0 (dois pontos)**
- Construção da Árvore Sintática: Se o programa constrói árvores sintáticas corretamente. **Valor: 0,5 (meio ponto)**
- Documentação: qualidade da documentação apresentada, incluindo documentação do código-fonte, instruções de uso, autômatos, etc. **Valor: 2,0 (dois pontos)**
- Qualidade do Programa: se foram utilizadas estruturas de programação orientada a objetos e/ou estruturada tornando o programa fácil de entender e dar manutenção. **Valor: 0,5 (meio ponto)**

Especificação Sintática da Linguagem Java

Gramática BNF:

classDeclaration → **public class identifier** *classBody*

classBody → **abre_chave** *classBodyDeclaration* **fecha_chave**

classBodyDeclaration → *declarationList methodDeclaration*

declarationList → *declarationList* *declaration* **pontoevirgula**
| *declaration* **pontoevirgula**
| ∈

declaration → *type* *identifierList*

type → **boolean**
| **char**
| **float**
| **int**
| **string**
| **void**

identifierList → *identifierList* **virgula** **identifier**
| **identifier**

methodDeclaration → *methodDeclaration* *methodName*
| *methodName*

methodName → **public** *type* **identifier** **abrepar** *formalParameters* **fecharpar** **abre_chave** *methodBody* **fecha_chave**
| **public** **static** *type* **identifier** **abrepar** *formalParameters* **fecharpar** **abre_chave** *methodBody* **fecha_chave**
| **public** **static** **void** **main** **abrepar** **string** **operador_arg** **fecharpar** **abre_chave** *methodBody* **fecha_chave**

methodBody → *declarationList* *statementList*

formalParameters → *formalParameters* **virgula** *formalParameter*
| *formalParameter*
| ∈

formalParameter → *type* **identifier**

statementList → *statementList* *statement*
| *statement*
| ∈

statement → *ifStatement*
| *whileStatement*
| **return** *expression* **pontoevirgula**
| **break** **pontoevirgula**
| **identifier** **opatr** *expression* **pontoevirgula**
| **identifier** **abrepar** *expressionList* **fecharpar** **pontoevirgula**

literal → **identifier**
| **number**
| **const_caractere**
| **const_booleana**

expression → *expression* **operador_aditivo_arit** *expression*
| *expression* **operador_multiplicativo** *expression*
| *expression* **operador_aditivo_logic** *expression*
| *expression* **operador_relacional** *expression*
| *literal*
| **abrepar** *expression* **fecharpar**
| **operador_aditivo_arit** *expression*

expressionList → *expressionList* **virgula** *expression*
| *expression*
| ∈

ifStatement → **if** **abrepar** *expression* **fechapar** *statementBlock* *elseAlternative*

elseAlternative → **else** *statementBlock* | ∈

whileStatement → **while** **abrepar** *expression* **fecharpar** *statementBlock*

statementBlock → *statement*
| **abre_chave** *statementList* **fecha_chave**

Programa-Exemplo

Teste a funcionalidade do analisador sintático para o programa-exemplo abaixo. Nenhum erro deverá ser relatado pelo analisador. Uma vez funcionando corretamente retire um e outro elemento do programa-exemplo e verifique que tipo de erro o analisador relata.

```
public class Soma
{
     public static void main(String arg[])
     {
          int a, b, c;
          b = 10;
          c = 3;
          a = func(b,b+c);
     }
     public static int func(int x, int y)
     {
          int ret;
          ret = x+y;
          return ret;
     }
}
```