Análisis y procesamiento de información textual utilizando generadores de compiladores

Miguel Amarís Martos

12 de enero de 2024

Resumen

En el panorama actual, el procesamiento de información en diversos formatos se ha convertido en una práctica común. Para abordar esta tarea, se emplean diferentes herramientas diseñadas específicamente para analizar datos en diversos lenguajes de programación o formatos como JSON o XML. Sin embargo, la necesidad de una solución genérica que permita el análisis de una amplia variedad de gramáticas se ha vuelto evidente. En este contexto, se han desarrollado generadores de reconocedores gramaticales, como Java Compiler Compiler (JavaCC). JavaCC es una herramienta que permite la generación de analizadores léxicos y sintácticos a partir de una gramática definida por el usuario. Su versatilidad y capacidad para adaptarse a diversas gramáticas lo convierten en una elección atractiva para el procesamiento de información estructurada. Este proyecto se enfoca en explorar el potencial de JavaCC y su aplicación en la asignatura Procesamiento de la Información en Aplicaciones Telemáticas (PIAT). El objetivo principal es aprender a utilizar JavaCC, aplicarlo en prácticas de PIAT y crear una documentación que respalde su versatilidad para el análisis de gramáticas y el procesamiento de información.

Agradecimientos

Quiero agradecer...

Índice general

1. Introducción **4**
- 1.1. Contexto y motivación . . . 4
- 1.2. Objetivos . . . 4
- 1.3. Restricciones . . . 4
- 1.4. Estructura del documento . . . 4

2. Estado del arte **5**
- 2.1. Marco tecnológico . . . 5
- 2.2. JavaCC . . . 5
- 2.3. Funciones principales de JavaCC . . . 5
- 2.4. Ejemplo de gramática en JavaCC . . . 6
- 2.5. Analizador Léxico . . . 10
- 2.6. Estados Léxicos . . . 11
- 2.7. Analizador Sintáctico o Semántico . . . 11
- 2.8. Recursividad . . . 12
  - 2.8.1. Recursividad a derechas (LR) . . . 12
  - 2.8.2. Recursividad a izquierdas (LL) . . . 13
  - 2.8.3. ¿Por qué no se puede utilizar la recursividad a izquierdas en JavaCC? . . . 13
- 2.9. Procesamiento de la Información en Aplicaciones Telemáticas (PIAT) 13
  - 2.9.1. Introducción . . . 13
  - 2.9.2. Problemática con las prácticas de PIAT . . . 14

3. Desarrollo **15**
- 3.1. Introducción . . . 15
- 3.2. Implementación . . . 15
- 3.3. Procedimientos . . . 15
- 3.4. Entorno . . . 15
- 3.5. Objetivos de la implementación . . . 16
- 3.6. Conclusiones . . . 16
- 3.7. Análisis de archivos XML. Práctica 3 . . . 16
  - 3.7.1. Introducción . . . 16
  - 3.7.2. SAX . . . 16
  - 3.7.3. JavaCC vs SAX . . . 17
  - 3.7.4. Descripción de la práctica . . . 18
  - 3.7.5. Desarrollo de la práctica . . . 19
  - 3.7.6. Herramienta . . . 19
  - 3.7.7. Conclusiones . . . 19
  - 3.7.8. Preguntas Frecuentes . . . 20

3.7.9. Notas . . . 20
3.8. Análisis de archivos JSON. Práctica 4 . . . 20
3.8.1. Introducción . . . 20
3.8.2. GSON Streaming . . . 20
3.8.3. JavaCC vs GSON Streaming . . . 21
3.8.4. Descripción de la práctica . . . 21
3.8.5. Desarrollo de la práctica . . . 21
3.8.6. Herramienta . . . 21
3.8.7. Conclusiones . . . 21
3.8.8. Preguntas Frecuentes . . . 22
3.8.9. Notas . . . 22
3.9. Transformación de archivos XML. Práctica 5 . . . 22
3.9.1. Introducción . . . 22
3.9.2. XPath . . . 22
3.9.3. JavaCC vs XPath . . . 23
3.9.4. Descripción de la práctica . . . 23
3.9.5. Desarrollo de la práctica . . . 23
3.9.6. Herramienta . . . 23
3.9.7. Conclusiones . . . 23
3.9.8. Preguntas Frecuentes . . . 24
3.9.9. Notas . . . 24

4. Resultados **25**

5. Conclusiones **26**

A. Preguntas Frecuentes **27**
A.1. ¿Que es una produccion BNF o ENBF? . . . 27
A.1.1. BNF . . . 27
A.1.2. EBNF . . . 27
A.2. ¿Que es un no-terminal? . . . 28
A.3. ¿Como empiezo a desarrollar en JavaCC? . . . 29
A.4. ¿Donde puedo encontrar información adicional? . . . 29

B. Herramientas utilizadas **31**
B.1. Manual de instalación y configuración . . . 31
B.1.1. Instalación de JavaCC . . . 31
B.2. Símbolos de Expresiones regulares en JavaCC . . . 33

Capítulo 1

Introducción

1.1. Contexto y motivación

El proyecto...

1.2. Objetivos

El objetivo principal del proyecto es evaluar la viabilidad de utilizar una herramienta de generadores genéricos como JavaCC para abordar las prácticas de PIAT. Esto implica:

- Aprender a utilizar JavaCC de manera efectiva.
- Aplicar JavaCC en prácticas específicas de PIAT. (Quitar los bucles y los loops de piat)
- Generar documentación que sirva como recurso para estudiantes y profesores interesados en utilizar JavaCC en proyectos relacionados con el procesamiento de información.

El proyecto busca proporcionar una solución versátil y eficaz para el análisis de gramáticas y el procesamiento de información en el ámbito de PIAT

1.3. Restricciones

En la realización del proyecto, debemos tener en cuenta las siguientes restricciones:

- El proyecto se centrará en la utilización de JavaCC como herramienta principal.
- Debe garantizarse que la documentación generada sea comprensible y útil para aquellos que deseen aplicar JavaCC en proyectos similares.
- Las prácticas de PIAT deben servir como un contexto relevante para la aplicación de JavaCC.

1.4. Estructura del documento

Capítulo 2

Estado del arte

2.1. Marco tecnológico

El procesamiento de información en diversos formatos es una tarea cada vez más importante. En un mundo cada vez más digital, es necesario poder procesar datos en una amplia variedad de formatos, desde lenguajes de programación hasta formatos de documentos y datos estructurados. En este contexto, los generadores de reconocedores gramaticales, como JavaCC, ofrecen una solución muy prometedora. Estos generadores permiten crear analizadores léxicos y sintácticos a partir de una gramática definida por el usuario. Esto los hace muy versátiles y adaptables a una amplia variedad de gramáticas. La aplicación de JavaCC en la asignatura Procesamiento de la Información en Aplicaciones Telemáticas (PIAT) es una idea muy acertada. PIAT es una asignatura que se centra en el procesamiento de información en diversos formatos. El uso de JavaCC podría proporcionar a los estudiantes una herramienta muy valiosa para abordar las prácticas de la asignatura.

2.2. JavaCC

En relación con lo anterior, aclaremos a qué hace referencia el concepto de JavaCC y los analizadores semánticos y sintácticos, y como se interrelacionan: Java Compiler Compiler, comúnmente conocido como JavaCC, es una poderosa herramienta ampliamente utilizada para generar analizadores sintácticos en aplicaciones Java. Su función principal es transformar una especificación gramatical en un programa Java capaz de reconocer y analizar la sintaxis según la gramática proporcionada. JavaCC se diferencia de otras herramientas similares al generar analizadores sintácticos de arriba hacia abajo (descenso recursivo) en lugar de analizadores de abajo hacia arriba. Esta característica permite el uso de gramáticas más generales y facilita la depuración, además de permitir el análisis en cualquier elemento no terminal de la gramática y la transferencia de valores (atributos) en ambas direcciones en el árbol de análisis.

2.3. Funciones principales de JavaCC

JavaCC ofrece una serie de características y funcionalidades clave que lo hacen destacar como un generador de analizadores sintácticos:

- Generación de analizadores sintácticos de arriba hacia abajo.
- Resolución de ambigüedades de cambio de turno localmente.
- Generación de analizadores 100 % Java puro.
- Especificaciones BNF extendidas.
- Integración de especificaciones léxicas y gramaticales en un solo archivo.
- Manejo de la entrada Unicode completa.
- Soporte para tokens no distinguibles entre mayúsculas y minúsculas.
- Herramientas adicionales como JJTree para construir árboles y JJDoc para generar documentación.
- Personalización a través de numerosas opciones.
- Informes de errores de alta calidad y mensajes de diagnóstico completos.

2.4. Ejemplo de gramática en JavaCC

A continuación, se presenta un ejemplo de cómo JavaCC puede utilizarse para generar un analizador sintáctico en Java. En este ejemplo se define un lenguaje para analizar expresiones matemáticas simples. Partiendo de este ejemplo, seremos capaz de ampliar las funcionalidades del programa, y realizar una calculadora:

```
// Define la gramática para analizar una simple expresión
   matemática.
void Expression() :
{}
{
  Term() ( "+" Term() | "-" Term() )*
}

void Term() :
{}
{
  Factor() ( "*" Factor() | "/" Factor() )*
}

void Factor() :
{}
{
  <NUMBER>
  | "(" Expression() ")"
}
```

Función Expression()

La función Expression() define la estructura general de una expresión. Una expresión consiste en un término seguido de cero o más términos conectados por operadores aritméticos. La función Expression() comienza con la función Term(). Esto significa que la primera parte de cualquier expresión debe ser un término. A continuación, la función Expression() utiliza la regla)* para especificar que puede seguir cero o más términos. Esto significa que las expresiones pueden tener cualquier longitud, desde una sola palabra hasta una expresión compleja con muchos términos. Los términos están conectados por operadores aritméticos. La regla Expression() utiliza la regla

)* para especificar que puede seguir cualquier número de operadores aritméticos. Esto significa que las expresiones pueden tener cualquier cantidad de operadores aritméticos, desde ninguno hasta muchos. Para más información acerca de operadores en JavaCC, vaya al anexo Operadores en JavaCC. Los operadores aritméticos permitidos son la suma (+), la resta (-), la multiplicación (*) y la división (/).

Función Term()

La regla Term() define la estructura de un término. Un término consiste en un factor seguido de cero o más factores conectados por operadores aritméticos. La regla Term() comienza con la regla Factor(). Esto significa que la primera parte de cualquier término debe ser un factor. A continuación, la regla Term() utiliza la regla)* para especificar que puede seguir cero o más factores. Esto significa que los términos pueden tener cualquier longitud, desde una sola palabra hasta un término complejo con muchos factores. Los factores están conectados por operadores aritméticos. La regla Term() utiliza la regla)* para especificar que puede seguir cualquier número de operadores aritméticos. Esto significa que los términos pueden tener cualquier cantidad de operadores aritméticos, desde ninguno hasta muchos. Los operadores aritméticos permitidos son la suma (+), la resta (-), la multiplicación (*) y la división (/)

Función Factor()

La regla Factor() define la estructura de un factor. Un factor puede ser un número o una expresión entre paréntesis. La regla Factor() tiene dos opciones. La primera opción es un número. La segunda opción es una expresión entre paréntesis. Si la opción elegida es un número, la regla Factor() utiliza la regla ¡NUMBER¿para especificar que el factor debe ser un número. Si la opción elegida es una expresión entre paréntesis, la regla Factor() utiliza la regla "(.Expression() ")"para especificar que la expresión debe estar entre paréntesis.

Ejemplos Aquí hay algunos ejemplos de expresiones que pueden ser analizadas por esta gramática:

- 1 + 2
- 3 * 4
- (5 - 6) / 7

Aunque también se puede analizar expresiones más complejas, como:

- (1 + 2) * (3 - 4)
- (5 * 6) / (7 + 8)

Ampliación de la funcionalidad a Calculadora

A continuación, vamos a ampliar las capacidades de nuestro programa, para analizar las expresiones y hacer las operaciones de una calculadora:

```
options {
  STATIC = false;
}
PARSER_BEGIN(NL_Xlator)
/**
 * New line translator.
 */
public class NL_Xlator
```

```
{
  /** Main entry point. */
  public static void main(String args []) throws ParseException
  {
    NL_Xlator parser = new NL_Xlator(System.in);
    parser.ExpressionList();
  }
}

PARSER_END(NL_Xlator)

SKIP :
{
  " "
| "\t"
| "\n"
| "\r"
}

TOKEN :
{
  < ID : [ "a"-"z", "A"-"Z", "_" ] ([ "a"-"z", "A"-"Z", "_",
   "0"-"9" ])* >
|
  < NUM : ([ "0"-"9" ])+ >
}

/** Top level production. */
void ExpressionList() :
{
    double e;
}
{
    {
      System.out.println("Please type in an expression followed by
   a \";\" or ^D to quit:");
      System.out.println("");
    }
  (
    e = Expression() ";"
      {
        System.out.println("Resultado =" + e);
        System.out.println("");
        System.out.println("Please type in another expression
   followed by a \";\" or ^D to quit:");
        System.out.println("");
      }
  )*
  < EOF >
}

/** An Expression. */
double Expression() :
{
    double e = 0, t = 0;
}
{
  t = Term()
    {
      e = t;
```

```
    }
  (
    "+" t = Term()
      {
        e = e + t;
      }
  |
    "-" t = Term()
      {
        e = e - t;
      }
  )*
  {
    return e;
  }
}

/** A Term. */
double Term() :
{
    double f = 0, t = 0;
}
{
  f = Factor()
    {
      t = f;
    }
  (
    "*" f = Factor()
      {
        t = t * f;
      }
  |
    "/" f = Factor()
      {
        t = t / f;
      }
  )*
  {
    return t;
  }
}

/** A Factor. */
double Factor() :
{
    Token t;
    double e;
}
{
  t = < NUM >
    {
      return Double.parseDouble(t.image);
    }
|
  "(" e = Expression() ")"
    {
      return e;
    }
}
```

Esta clase traduce expresiones matemáticas válidas en sus valores numéricos correspondientes. El usuario puede ingresar las expresiones matemáticas una por una, separándolas por punto y coma. La clase lee las expresiones del flujo de entrada estándar y las traduce a sus valores numéricos correspondientes. El resultado de la traducción se muestra en la salida estándar.

Análisis sintáctico

La clase utiliza JavaCC para analizar las expresiones matemáticas ingresadas por el usuario. JavaCC es una herramienta de generación de analizadores de sintaxis que permite crear analizadores personalizados para lenguajes de programación o lenguajes de dominio específicos.

Expresión raíz

La expresión raíz de la gramática es ExpressionList(). Esta producción analiza una lista de expresiones matemáticas separadas por punto y coma. Cada expresión matemática es analizada por la producción Expression().

Expresión

La producción Expression() analiza una expresión matemática completa. Esta producción analiza un término (Term()) seguido de cero o más operadores de suma o resta (+ o -) y términos.

Término

La producción Term() analiza un término matemático. Esta producción analiza un factor (Factor()) seguido de cero o más operadores de multiplicación o división (* o /) y factores.

Factor

La producción Factor() analiza un factor matemático. Esta producción analiza un número (NUM) o un par de paréntesis ((,)) que encierran una expresión matemática.

Errores La clase también maneja errores sintácticos. Cuando ocurre un error sintáctico, la clase muestra un mensaje de error en la salida estándar y termina la ejecución.

Ejemplo de uso

Para utilizar la clase, simplemente ejecute el archivo NL_Xlator.java. A continuación, se le pedirá que ingrese expresiones matemáticas una por una, separándolas por punto y coma. La clase le mostrará el resultado de la traducción de cada expresión. Aquí hay un ejemplo de cómo utilizar la clase:

2.5. Analizador Léxico

GramaticaTokenManager es literalmente el analizador lexico SimpleCharStream se usa para leer los tokens. Usa por debajo clases como Reader

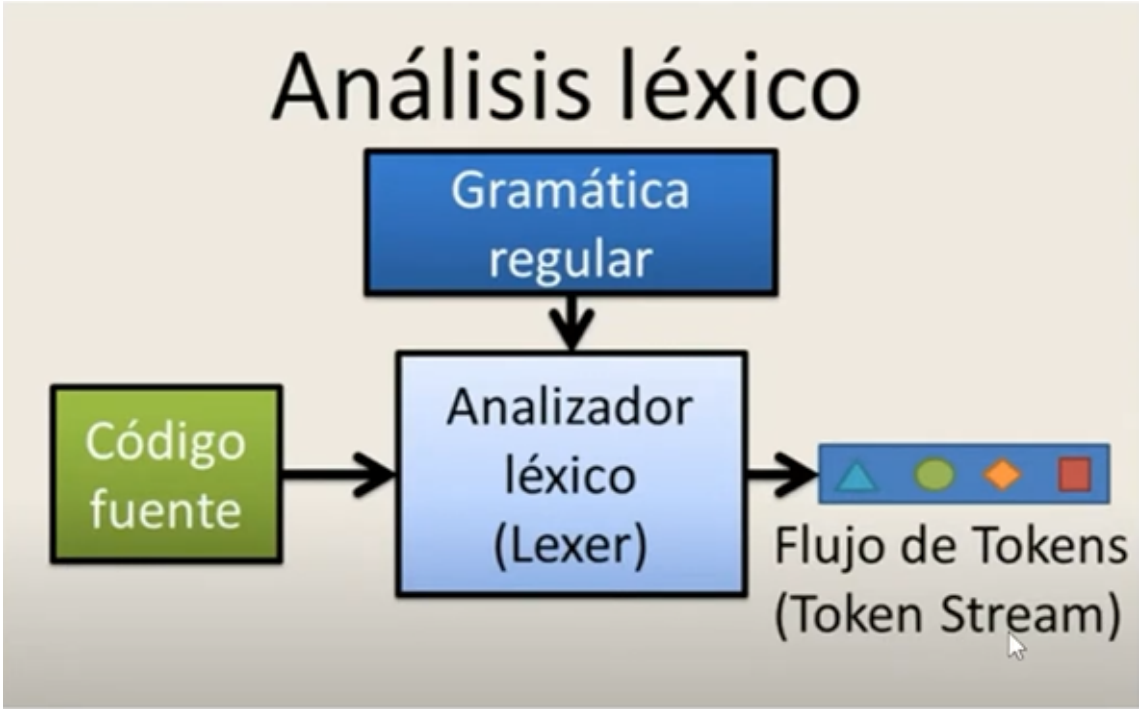

Figura 2.1: Funcionamiento del Analizador Léxico en JavaCC

2.6. Estados Léxicos

Los estados léxicos permiten efectuar un conjunto diferente de producciones de expresiones regulares en un caso concreto. Dicho de otra manera, los estados léxicos sirven para que un token este definido de una forma, y en cierto momento del análisis, pueda estar definido de otra forma.

Supongamos que desea escribir un procesador JavaDoc para extraer los comentarios JavaDoc de un documento. La mayor parte de Java está tokenizada de acuerdo con las reglas ordinarias regulares de Java. Pero dentro de los comentarios JavaDoc, se aplica un conjunto diferente de reglas en las que las palabras clave deben ser reconocidas y donde las nuevas líneas son significativas. /***/@param

Para resolver este problema, podríamos usar dos estados léxicos: uno para la tokenización regular de Java y otro para la tokenización dentro de los comentarios de JavaDoc.

2.7. Analizador Sintáctico o Semántico

Nos dice como tienen que estar acomodadas las cosas en el código fuente que tenemos para hacerlo un programa valido.

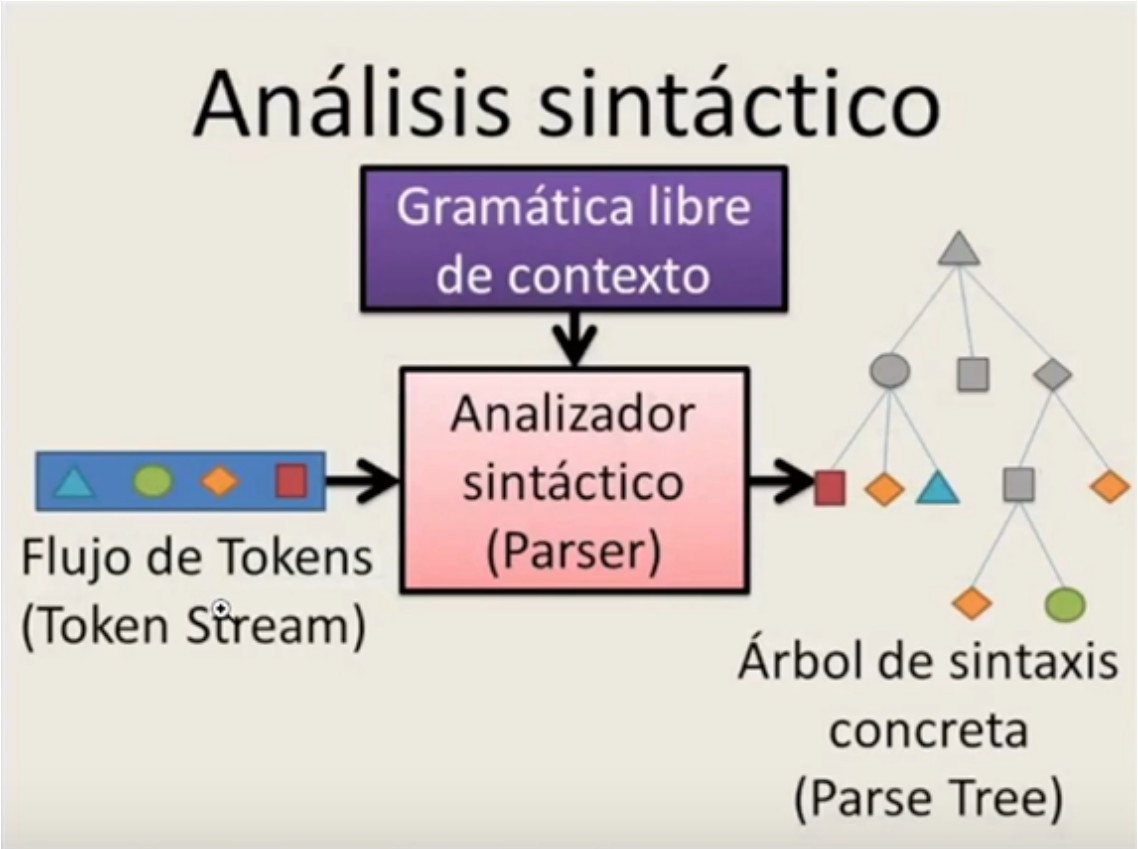

Figura 2.2: Funcionamiento del Analizador Sintáctico en JavaCC

2.8. Recursividad

En el ámbito de los compiladores, la recursividad es una técnica que se utiliza para definir una estructura gramatical que se repite en sí misma.

Esto permite resolver problemas complejos dividiéndolos en subproblemas más pequeños. Este proceso se repite hasta que se llega a subproblemas que son triviales de resolver.

En JavaCC, la recursividad se utiliza para definir producciones gramaticales que pueden analizar expresiones complejas.

2.8.1. Recursividad a derechas (LR)

En la recursividad a derechas, la llamada recursiva se realiza al final de la definición de la estructura gramatical. Por ejemplo, la gramática para la expresión aritmética básica se puede definir recursivamente a la derecha de la siguiente manera:

```
Expression() :
{
  Token t;
}
{
  "(" e = Expression() ")"
  {
    return e;
  }
}
```

2.8.2. Recursividad a izquierdas (LL)

En la recursividad a izquierdas, la llamada recursiva se realiza al principio de la definición de la estructura gramatical. Por ejemplo, la gramática para la expresión aritmética básica se puede definir recursivamente a la izquierda de la siguiente manera:

```
Expression() :
{
  double e;
}
{
  e = Term()
  (
    "+" e = Term()
  |
    "-" e = Term()
  )*
  {
    return e;
  }
}
```

2.8.3. ¿Por qué no se puede utilizar la recursividad a izquierdas en JavaCC?

La clase de analizador sintáctico producida por JavaCC funciona por descenso recursivo. La recursión a la izquierda está prohibida para evitar que las subrutinas generadas se llamen a sí mismas recursivamente de forma infinita.

Si considerasemos una producción recursiva a la izquierda, en caso de que la condición fuera verdadera alguna vez, tendríamos una recursión infinita. JavaCC producirá un mensaje de error si tiene producciones recursivas a la izquierda.

2.9. Procesamiento de la Información en Aplicaciones Telemáticas (PIAT)

2.9.1. Introducción

Las prácticas de PIAT son una serie de ejercicios que se utilizan para enseñar a los estudiantes de la asignatura los fundamentos de la programación orientada a objetos, y como se puede utilizar para analizar y extraer información de distintos formatos. Estas prácticas incluyen ejercicios de análisis de ficheros XML y JSON, principalmente.

La aplicación de JavaCC en las prácticas de PIAT tiene varias ventajas potenciales. En primer lugar, puede ayudar a los estudiantes a aprender los fundamentos de la programación orientada a objetos de una manera más eficiente y de una forma que todavía no han tenido la oportunidad de aprender y explotar. En segundo lugar, puede ayudar a los estudiantes a desarrollar sus habilidades de programación, y por consecuencia, ayudar a los estudiantes a crear código más eficiente y de mejor calidad.

2.9.2. Problemática con las prácticas de PIAT

Las prácticas de PIAT, que se centran en el análisis de archivos XML o JSON, utilizan soluciones como SAXParser o XPath – las veremos en detalle mas adelante –. Estas soluciones, sin embargo, presentan una serie de problemas que pueden dificultar el desarrollo de código eficiente y mantenible.

Uno de los principales problemas de SAXParser y XPath es que generan mucho código repetitivo. Los bucles while y loops que se utilizan para recorrer el árbol de datos pueden ser muy largos y complejos, lo que dificulta la comprensión y el mantenimiento del código.

Otro problema de estas soluciones es que pueden ser ineficientes en el rendimiento. SAXParser, por ejemplo, se basa en un modelo de eventos, lo que significa que el código debe estar preparado para procesar cualquier tipo de evento que pueda ocurrir. Esto puede provocar que el código se ejecute de forma innecesaria, lo que puede afectar al rendimiento de la aplicación.

Una buena solución a este tipo de soluciones es un generador de analizadores sintácticos como JavaCC. JavaCC permite crear analizadores sintácticos a partir de una gramática definida por el usuario. Esto permite generar código más eficiente y fácil de mantener, ya que el analizador sintáctico se genera automáticamente a partir de la gramática.

Al definir la gramática del analizador sintáctico de forma personalizad, este ofrece una gran flexibilidad, ya que permite adaptar el analizador a las necesidades específicas de la aplicación. Por ejemplo, si queremos analizar un archivo XML que tiene una estructura específica, podemos definir una gramática que refleje esa estructura. Esto nos permitirá generar un analizador que sea más eficiente y fácil de mantener.

Además, una vez que se ha definido la gramática, JavaCC genera el código fuente del analizador sintáctico. Este código fuente se puede modificar a conveniencia, lo que nos permite adaptar el analizador a las necesidades específicas de la aplicación. Esto permite que, si queremos añadir nuevas funcionalidades al analizador, podemos hacerlo fácilmente modificando el código fuente.Esto nos permite tener un mayor control sobre el analizador y adaptarlo a las necesidades cambiantes de la aplicación, característica clave en el entorno de las practicas de PIAT, ya que se caracterizan por tener un estilo incremental y con modificaciones a lo largo de cada ejercicio.

Por otra parte, si bien es cierto que JavaCC puede ser una herramienta compleja para aprender a utilizar, ya que requiere un conocimiento básico de gramáticas y de la sintaxis de Java, el potencial que ofrece JavaCC para desarrollar analizadores sintácticos eficientes y adaptables es muy alto.

Una vez que se ha aprendido a utilizar JavaCC, se puede utilizar para desarrollar analizadores sintácticos para una amplia gama de lenguajes, incluyendo XML, JSON, HTML, C++, PHP, Cobol, SQL, IDL, entre otros.

Capítulo 3

Desarrollo

3.1. Introducción

En el capítulo anterior se han expuesto las principales tecnologías que forman parte del proyecto de evaluación de la viabilidad de JavaCC para el procesamiento de información en la asignatura PIAT. En este capítulo se desarrollarán los aspectos relativos a la implementación del proyecto, los procedimientos seguidos y el entorno utilizado para los desarrollos y las pruebas.

3.2. Implementación

La implementación del proyecto se ha realizado en Java, utilizando la herramienta JavaCC para la generación de los analizadores léxico y sintáctico. Para la creación de las gramáticas se ha utilizado el lenguaje EBNF.

3.3. Procedimientos

El proceso de implementación se ha dividido en las siguientes fases:

- Fase 1: Aprendizaje de los conceptos básicos de JavaCC.
- Fase 2: Práctica con JavaCC para crear analizadores léxicos y sintácticos para gramáticas simples.
- Fase 3: Aplicación de JavaCC en prácticas de PIAT.
- Fase 4: Generación de documentación.

3.4. Entorno

Los desarrollos y las pruebas se han realizado en un entorno de desarrollo integrado (IDE) de Java. Para la depuración de los analizadores se ha utilizado el debugger del IDE.

3.5. Objetivos de la implementación

Los objetivos de la implementación son los siguientes:

- Implementar los analizadores léxico y sintáctico para las gramáticas de las prácticas de PIAT.
- Evaluar la viabilidad de utilizar JavaCC para abordar las prácticas de PIAT.
- Generar documentación que sirva como recurso para estudiantes y profesores interesados en utilizar JavaCC en proyectos relacionados con el procesamiento de información.

3.6. Conclusiones

En esta sección se ha presentado una introducción al desarrollo de la propuesta. En las siguientes secciones se desarrollarán los aspectos relativos a la implementación, los procedimientos seguidos y el entorno utilizado para los desarrollos y las pruebas.

3.7. Análisis de archivos XML. Práctica 3

3.7.1. Introducción

La práctica 3 de PIAT, centrada en el procesamiento de documentos XML, representa un hito significativo en el marco de este proyecto de fin de grado. En esta etapa, se aborda la implementación de un analizador de documentos XML utilizando la tecnología SAX (Simple API for XML). El objetivo principal es familiarizar a los estudiantes con SAX y fortalecer sus habilidades en el diseño de algoritmos eficientes para la extracción y transformación de información a partir de fuentes de contenidos estructurados. La práctica 3 de PIAT se centra en familiarizarse con la tecnología SAX y desarrollar un analizador de documentos XML basado en esta tecnología. Además, el objetivo secundario es diseñar algoritmos eficientes que permitan la extracción y transformación de información a partir de fuentes de contenidos estructurados.

3.7.2. SAX

La tecnología SAX (Simple API for XML) es una API que se encarga de procesar documentos XML. Está basada en eventos, lo que quiere decir que el analizador XML notifica al programa cuando encuentra un evento especifico en el documento. El programa puede tomar medidas en función del evento. Los eventos que puede notificar SAX son:

- Inicio de documento: Se produce cuando el analizador comienza a procesar el documento.
- Fin de documento: Se produce cuando el analizador finaliza el procesamiento del documento.
- Inicio de elemento: Se produce cuando el analizador encuentra el inicio de un elemento XML.

- Fin de elemento: Se produce cuando el analizador encuentra el final de un elemento XML.
- Caracteres: Se produce cuando el analizador encuentra caracteres no pertenecientes a un elemento XML.
- Error: Se produce cuando el analizador encuentra un error en el documento XML.

SAX es una buena opción para procesar documentos XML grandes o complejos. Esto se debe a que el analizador XML no necesita cargar todo el documento en memoria a la vez. En cambio, el analizador puede procesar el documento de forma secuencial, lo que puede ahorrar memoria.

SAX también es una buena opción para procesar documentos XML que se actualizan con frecuencia. Esto se debe a que el analizador XML puede procesar solo los cambios en el documento, lo que puede ahorrar tiempo.

Algunos ejemplos de cómo se puede utilizar SAX para procesar documentos XML incluyen:

- Leer un documento XML y extraer los datos que contiene.
- Validar un documento XML para asegurarse de que cumple con las reglas de un esquema XML.
- Transformar un documento XML a un formato diferente.

3.7.3. JavaCC vs SAX

JavaCC y SAX son dos API diferentes para procesar documentos XML. JavaCC es una herramienta de generación de analizadores que permite crear analizadores XML personalizados. SAX es una API estándar que proporciona un conjunto de métodos para procesar documentos XML. Hay varias razones por las que JavaCC puede ser una mejor opción que SAX para procesar documentos XML:

- Control: JavaCC ofrece un mayor control sobre el proceso de análisis que SAX. Esto permite a los desarrolladores crear analizadores que se adapten a sus necesidades específicas.
- Eficiencia: JavaCC puede generar analizadores que sean más eficientes que los analizadores SAX estándar. Esto se debe a que JavaCC puede aprovechar las características del lenguaje Java para optimizar el proceso de análisis.
- Flexibilidad: JavaCC permite crear analizadores XML que sean más flexibles que los analizadores SAX estándar. Esto se debe a que JavaCC permite a los desarrolladores definir sus propias reglas de análisis.

En concreto, JavaCC ofrece las siguientes ventajas sobre SAX:

- Puede generar analizadores personalizados que se adapten a las necesidades específicas del documento XML. Esto permite a los desarrolladores crear analizadores que sean más eficientes, precisos y fáciles de mantener.
- Puede generar analizadores que sean más eficientes que los analizadores SAX estándar. Esto se debe a que JavaCC puede aprovechar las características del lenguaje Java para optimizar el proceso de análisis.

- Puede generar analizadores que sean más flexibles que los analizadores SAX estándar. Esto se debe a que JavaCC permite a los desarrolladores definir sus propias reglas de análisis.

Sin embargo, JavaCC también tiene algunas desventajas con respecto a SAX:

- Es más complejo de aprender y usar que SAX. Esto se debe a que JavaCC requiere conocimientos de programación en Java.
- Puede ser más lento que SAX para documentos XML pequeños. Esto se debe a que JavaCC debe generar un analizador personalizado para cada documento XML.

En general, JavaCC es una buena opción para procesar documentos XML cuando se necesita un mayor control, eficiencia o flexibilidad. Sin embargo, SAX es una buena opción para procesar documentos XML pequeños o cuando se necesita una solución rápida y fácil de usar.

3.7.4. Descripción de la práctica

Se desea desarrollar una aplicación que extraiga cierta información de un fichero XML obtenido a partir de los datos publicados en el Portal de Datos Abiertos del Ayuntamiento de Madrid (http://datos.madrid.es).

La información (organismos, eventos, actividades, ...) publicada a través del Portal de Datos Abiertos presenta las siguientes características:

- Los elementos de información, en adelante recursos (concept), se identifican mediante una URI.
- Cada recurso se encuentra asociado a una categoría (elemento code de concept).
- Los recursos se encuentran agrupados en conjuntos de datos (elemento dataset) accesibles en formato JSON a través de un URI indicada en el atributo id del elemento dataset.
- En un dataset puede haber información sobre recursos asociados a varias categorías.
- Los recursos de una categoría pueden estar accesibles a través de diferentes datasets.
- Para la categorización de los recursos se utiliza un sistema de clasificación jerárquico basando en características temáticas.

Esta información se encuentra descrita en un Catálogo de Datos en formato XML (catalogo.xml) válido con respecto al esquema catalogo.xsd. La aplicación por desarrollar proporcionará una herramienta de búsqueda que posibilite la recuperación de recursos asociados a un código de categoría generando un documento XML con los resultados. La Figura 1 muestra un ejemplo de presentación de parte de la estructura jerárquica de los concepts del catálogo.

Figura 1.- Representación de la estructura jerárquica de los concepts del catálogo de datos

La aplicación por desarrollar recibirá como argumento el criterio de búsqueda, esto es, el código de la categoría de la que se desea información, y proporcionará información sobre los concepts y datasets pertinentes, aplicando para ello los siguientes criterios:

- Se considerarán pertinentes el concept cuyo código (elemento code) coincida con el criterio de búsqueda y todos los concepts descendientes del mismo.
- Se considerarán pertinentes los dataset que contengan información asociada a alguno de los concept pertinentes (contengan un elemento concept con el atributo id igual al atributo id del elemento concept pertinente).

La Figura 2 muestra un ejemplo de búsqueda del concept con código 0097-022 y los resultados que se obtendrían.

3.7.5. Desarrollo de la práctica

La implementación de la Práctica 3 se basa en la creación de un analizador de documentos XML denominado XMLParser, desarrollado utilizando JavaCC, una herramienta que permite generar analizadores léxicos y sintácticos en el lenguaje Java. Este analizador se ha diseñado para procesar documentos XML que contienen datos publicados en el Portal de Datos Abiertos del Ayuntamiento de Madrid (http://datos.madrid.es). Los documentos XML que se procesan a través de esta herramienta presentan ciertas características clave, como la identificación de recursos mediante URI, la asociación a categorías, la agrupación en conjuntos de datos y una clasificación jerárquica basada en características temáticas. La tarea principal del analizador es extraer información específica relacionada con códigos de categorías y, posteriormente, generar un nuevo documento XML que contiene los resultados deseados.

3.7.6. Herramienta

La principal herramienta desarrollada en esta práctica es el analizador XMLParser. Este analizador es capaz de llevar a cabo varias tareas esenciales:

- Validación de argumentos de entrada para garantizar que los parámetros cumplan con las especificaciones requeridas.
- Extracción de información relevante de los documentos XML, siguiendo las reglas y estructuras definidas en la gramática.
- Generación de documentos de resultados que cumplen con un esquema específico.

El analizador XMLParser representa una contribución significativa en términos de procesamiento y manipulación de documentos XML.

3.7.7. Conclusiones

La Práctica 3 ha brindado a los estudiantes una experiencia valiosa en el manejo de tecnologías de análisis de documentos XML. A través del uso de JavaCC y la tecnología SAX, se ha proporcionado una base sólida para procesar información estructurada de manera eficiente. Los estudiantes han tenido la oportunidad de aplicar conceptos relacionados con la validación de argumentos, el análisis de elementos

XML y la generación de documentos de resultados. Uno de los aspectos más destacados de esta práctica es la capacidad de filtrar y seleccionar información relevante basada en códigos de categorías. Esta habilidad es fundamental en situaciones donde la extracción selectiva de datos es esencial.

3.7.8. Preguntas Frecuentes

¿Qué sentido tiene utilizar la clase ManejadorXML, si el parser ya busca lo que quiere? Tiene sentido cronologico, ya que primero el parser realiza el analisis del documento y devuelve los datos, si se llama a una funcion que devuelva los datos es posible que se llamen a estas funciones antes de que se llame a la funcion de análisis

3.7.9. Notas

hacer clase concept con atributos code y label, hacer lista concepts de concept. hacer lo mismo con dataset. recoger los datos al igual que haciamos en NXCalculator Concepts, son objetos distintos, se tienen que reconocer de manera diferente. -¿Nuevo objeto conceptInDataset No hace falta un estado lexico, simplemente crear un objeto nuevo ya que son cosas diferentes, -¿IncludedConcepts String id -¿idConcepts XMLParser devuelve un objeto que tenga implementado la interfax ParserCatalogo

3.8. Análisis de archivos JSON. Práctica 4

3.8.1. Introducción

La Práctica 4 se centra en la exploración y aplicación de la tecnología GSON Streaming para el análisis y procesamiento eficiente de documentos JSON. El objetivo principal de esta etapa es familiarizar a los estudiantes con la API GSON Streaming y desarrollar un analizador de documentos JSON basado en esta tecnología. Como objetivo secundario, se busca capacitar a los estudiantes en el diseño de algoritmos eficientes para la extracción y transformación de información proveniente de fuentes de contenidos estructurados.

3.8.2. GSON Streaming

La API de transmisión de GSON es una API de Java que permite leer y escribir JSON de forma secuencial. Esto la hace útil en situaciones donde no es posible o deseable cargar el modelo de objeto completo en memoria, como cuando se trabaja con grandes cantidades de datos o cuando los datos se reciben de forma continua.

La API de transmisión de GSON se basa en dos clases principales: JsonReader y JsonWriter. JsonReader se utiliza para leer JSON de forma secuencial, mientras que JsonWriter se utiliza para escribir JSON de forma secuencial.

Las características principales de la API son su eficiencia en términos de memoria y su flexibilidad. Por una parte, la API de transmisión de GSON no necesita cargar el modelo de objeto completo en memoria, lo que la hace más eficiente en términos de memoria que la API de GSON tradicional. Además, GSON Streaming permite leer y escribir datos JSON de forma secuencial, lo que la hace muy flexible.

La API es ideal para trabajar con grandes cantidades de datos, ya que puede leer y escribir los datos sin necesidad de cargarlos todos en memoria. También es ideal

para recibir datos de forma continua, ya que puede leer los datos a medida que se reciben.

3.8.3. JavaCC vs GSON Streaming

3.8.4. Descripción de la práctica

La práctica busca ampliar la funcionalidad de la herramienta de búsqueda del Portal de Datos Abiertos del Ayuntamiento de Madrid, desarrollada en la Práctica 3. Se pretende dotar a esta herramienta de la capacidad para extraer información sobre los recursos asociados a una categoría específica (concept) del portal. Esto se logrará accediendo a los conjuntos de datos (dataset) en formato JSON y procesándolos con un analizador GSON Streaming.

El código desarrollado en la Práctica 3 servirá como punto de partida, y se completará para integrar un analizador GSON Streaming. Este analizador procesará la información de cada conjunto de datos pertinente y generará un documento XML válido conforme al esquema de documento ResultadosBusquedaP4.xsd. El nuevo elemento introducido en este esquema es resources, que contendrá información sobre los recursos asociados a las categorías pertinentes.

La generación del documento XML (indicado por ARG2) será similar al proceso de la Práctica 3, pero ahora se incorporará el elemento resources. Se utilizará la clase JSONDatasetParser para implementar el analizador GSON Streaming, y se analizarán los archivos .json indicados en el atributo id de cada dataset. Se añadirán como máximo cinco recursos (resource) a partir de cada dataset analizado.

3.8.5. Desarrollo de la práctica

3.8.6. Herramienta

La herramienta principal desarrollada en esta práctica es la extensión de la herramienta de búsqueda del Portal de Datos Abiertos, ahora mejorada con la capacidad de analizar y procesar datos JSON mediante la API GSON Streaming. La clase JSONDatasetParser representa la implementación de este analizador GSON Streaming. La herramienta final permite la extracción selectiva de información de archivos JSON, generando un documento XML conforme al esquema ResultadosBusquedaP4.xsd que incluye el nuevo elemento resources.

3.8.7. Conclusiones

La Práctica 4 ha proporcionado a los estudiantes una oportunidad única para aplicar sus conocimientos adquiridos en la Práctica 3 y explorar en profundidad la API GSON Streaming. La capacidad de extender la funcionalidad de la herramienta existente para manejar datos JSON y generar documentos XML enriquecidos demuestra una comprensión avanzada de las tecnologías de procesamiento de datos estructurados.

El uso de GSON Streaming versión 2.9.0 ha permitido a los estudiantes implementar un analizador eficiente para manejar grandes conjuntos de datos JSON, cumpliendo así con los requisitos de la práctica.

3.8.8. Preguntas Frecuentes

Pregunta 1: ¿Cuál es el propósito principal de la Práctica 4 en términos de procesamiento de información JSON? Respuesta: El objetivo principal es extender la herramienta de búsqueda para extraer información sobre recursos asociados a categorías específicas en el Portal de Datos Abiertos, utilizando la tecnología GSON Streaming.

Pregunta 2: ¿Qué elemento se introduce en el esquema ResultadosBusquedaP4.xsd? Respuesta: Se introduce el elemento resources, que contendrá información sobre los recursos asociados a las categorías pertinentes.

Pregunta 3: ¿Cuál es la versión de la API GSON Streaming utilizada en esta práctica? Respuesta: Se utiliza la versión 2.9.0 de la API GSON Streaming, disponible en este enlace.

3.8.9. Notas

3.9. Transformación de archivos XML. Práctica 5

3.9.1. Introducción

La Práctica 5 representa una etapa crucial en el proyecto, enfocándose en la familiarización con la tecnología de transformación de documentos XML mediante el Lenguaje de Rutas de XML (XPath), basado en el Modelo de Objetos de Documento (DOM). El objetivo principal es extraer información del documento XML generado en la Práctica 4 utilizando expresiones XPath y almacenarla en un nuevo documento en formato JSON. Esta práctica permite a los estudiantes explorar técnicas avanzadas de manipulación de datos estructurados.

3.9.2. XPath

XPath, abreviatura de XML Path Language, es un lenguaje de consulta para documentos XML. Se utiliza para seleccionar partes de un documento XML, como elementos, atributos, texto y datos binarios.

XPath se basa en una sintaxis similar a la de las expresiones regulares. Las expresiones XPath se utilizan para construir caminos a través de un documento XML.

Las principales características de XPath son las siguientes:

Selección de elementos: XPath se puede utilizar para seleccionar elementos individuales o conjuntos de elementos en un documento XML. Selección de atributos: XPath se puede utilizar para seleccionar atributos individuales o conjuntos de atributos de un elemento. Selección de texto: XPath se puede utilizar para seleccionar texto de un elemento o atributo. Selección de datos binarios: XPath se puede utilizar para seleccionar datos binarios de un elemento o atributo.

XPath se utiliza en una variedad de aplicaciones, incluyendo el procesamiento de XML, el desarrollo web, y la integración de datos. Esto permite que se utilize para procesar documentos XML, extraer datos, validar documentos y transformar documentos. Además, se utiliza en aplicaciones web para acceder a datos XML, como datos de formularios o datos de un servicio web. Por último, también se puede em-

plear para integrar datos de diferentes fuentes, como datos XML y datos de bases de datos.

3.9.3. JavaCC vs XPath

JavaCC y XPath son dos herramientas que se pueden utilizar para procesar documentos XML. Sin embargo, tienen algunas diferencias clave.

JavaCC es una herramienta de generación de analizadores sintácticos. Se utiliza para generar analizadores sintácticos que pueden analizar documentos XML.

Como ya hemos visto en secciones anteriores, las principales ventajas de JavaCC son su eficiencia en términos de memoria y su gran flexibilidad a los usuarios. Por otra parte, la principal desventaja de JavaCC es su dificultad de uso,

XPath es un lenguaje de consulta para documentos XML. Se utiliza para seleccionar partes de un documento XML.

Las principales ventajas de XPath son su sencillez, ya que es una herramienta relativamente sencilla de aprender a utilizar, y su generalidad, pues se puede utilizar para procesar cualquier documento XML, independientemente de su lenguaje.

Las principales desventajas de XPath son su eficiencia, debido a que es menos eficiente en términos de memoria que JavaCC, y sus limitaciones, como la imposibilidad de procesar datos binarios.

3.9.4. Descripción de la práctica

La implementación de la Práctica 5 se basa en extender el código desarrollado en la Práctica 4, integrando la funcionalidad de XPath para extraer información específica del documento XML resultante. El uso del Modelo de Objetos de Documento (DOM) facilita la navegación y manipulación de la estructura jerárquica del documento XML.

Se requerirá la introducción de expresiones XPath para identificar y seleccionar los elementos deseados en el documento XML. La información obtenida mediante XPath se almacenará en un nuevo documento en formato JSON. Este proceso de transformación garantiza que la información relevante se extraiga eficientemente y se represente en un formato JSON para su posterior análisis o intercambio.

3.9.5. Desarrollo de la práctica

3.9.6. Herramienta

La herramienta principal desarrollada en esta práctica es una extensión del código existente en la Práctica 4, ahora mejorado con la capacidad de utilizar expresiones XPath para extraer información específica del documento XML resultante. La información extraída se guarda en un nuevo documento en formato JSON, ofreciendo una representación alternativa y flexible de los datos.

3.9.7. Conclusiones

La Práctica 5 proporciona a los estudiantes una valiosa experiencia en la utilización de XPath para la extracción selectiva de información de documentos XML. La capa-

cidad de integrar esta tecnología con el código existente de la Práctica 4 demuestra la versatilidad en el manejo de datos estructurados.

El proceso de transformación a JSON destaca la flexibilidad de las tecnologías utilizadas, permitiendo la representación de datos en diferentes formatos según las necesidades del proyecto.

3.9.8. Preguntas Frecuentes

Pregunta 1: ¿Cuál es el propósito principal de la Práctica 5 en términos de transformación de datos? Respuesta: El objetivo principal es la extracción de información específica del documento XML mediante expresiones XPath y la posterior representación de esta información en un nuevo documento en formato JSON.

Pregunta 2: ¿Cómo se realiza la transformación de datos de XML a JSON en esta práctica? Respuesta: La transformación se realiza mediante la utilización de expresiones XPath para identificar y seleccionar información específica en el documento XML generado en la Práctica 4. La información seleccionada se almacena en un nuevo documento en formato JSON.

Pregunta 3: ¿Cómo se integra la funcionalidad XPath en el código existente de la Práctica 4? Respuesta: Se realizarán modificaciones en el código de la Práctica 4 para incorporar expresiones XPath que seleccionen la información deseada. La información extraída se utilizará para generar un nuevo documento en formato JSON

3.9.9. Notas

Capítulo 4

Resultados

Capítulo 5

Conclusiones

Apéndice A

Preguntas Frecuentes

A.1. ¿Que es una produccion BNF o ENBF?

En informática, una producción BNF o EBNF es una regla que define cómo se puede generar una secuencia de símbolos en un lenguaje formal. Las producciones BNF y EBNF se utilizan para definir la sintaxis de los lenguajes de programación, los sistemas de comandos y los protocolos de comunicación.

A.1.1. BNF

La notación de Backus-Naur (BNF) es un metalenguaje utilizado para expresar gramáticas libres de contexto. Una gramática libre de contexto es un tipo de gramática formal que se caracteriza por la ausencia de reglas de recursión izquierda.

Una producción BNF se compone de dos partes: el lado izquierdo y el lado derecho. El lado izquierdo de una producción es el símbolo que se está definiendo. El lado derecho de una producción es una expresión que especifica cómo se puede generar el símbolo del lado izquierdo.

Las expresiones en el lado derecho de una producción BNF pueden ser:

Símbolos simples: un símbolo simple es un símbolo que no se puede descomponer en otros símbolos. Por ejemplo, el símbolo + es un símbolo simple.

Secuencias de símbolos: una secuencia de símbolos es una lista de símbolos separados por espacios. Por ejemplo, la secuencia de símbolos a b c es una secuencia de tres símbolos.

Alternativas: una alternativa es una lista de opciones separadas por barras verticales (—). Por ejemplo, la alternativa a — b — c significa que el símbolo del lado izquierdo puede ser a, b o c.

Recursión derecha: una recursividad derecha es una producción que se refiere a sí misma en el lado derecho. Por ejemplo, la producción S -¿S + E — E significa que el símbolo S puede ser S más E o E.

A.1.2. EBNF

La notación EBNF (Extended Backus-Naur Form) es una extensión de la notación BNF. La notación EBNF incluye algunas características adicionales que hacen que

sea más fácil de usar para definir la sintaxis de los lenguajes de programación.

Las principales características adicionales de la notación EBNF son:

Recursiva izquierda: la notación EBNF permite la recursividad izquierda, lo que hace que sea más fácil de definir la sintaxis de los lenguajes de programación que utilizan recursividad izquierda, como los lenguajes de programación orientados a objetos.

Referencia de producciones: la notación EBNF permite referenciar producciones existentes en el lado derecho de una producción. Esto hace que sea más fácil de definir la sintaxis de los lenguajes de programación que utilizan estructuras de datos complejas, como los árboles.

Operadores de conjuntos: la notación EBNF incluye operadores de conjuntos que permiten especificar conjuntos de símbolos. Esto hace que sea más fácil de definir la sintaxis de los lenguajes de programación que utilizan conjuntos de símbolos, como los lenguajes de programación de patrones.

Ejemplos

Aquí hay algunos ejemplos de producciones BNF y EBNF:

BNF

S -¿E E -¿T + E — T T -¿id — num

Estas producciones definen la sintaxis de una expresión aritmética simple. El símbolo S representa una expresión, el símbolo E representa un término y el símbolo T representa un factor.

EBNF

$S ::= E\ E ::= T" + "E|T\ T ::= id|num$

Estas producciones son equivalentes a las producciones BNF anteriores.

Otro ejemplo

$S- > "("E")"|id\ E- > E" + "E|E" - "E|E" * "E|E"/"E|id$

Estas producciones definen la sintaxis de una expresión aritmética más compleja. El símbolo S representa una expresión, el símbolo E representa un término y el símbolo id representa un identificador.

Conclusiones

Las producciones BNF y EBNF son herramientas importantes para la definición de la sintaxis de los lenguajes formales. Las producciones BNF son más simples de usar, pero las producciones EBNF ofrecen algunas características adicionales que pueden ser útiles para definir la sintaxis de lenguajes de programación complejos.

A.2. ¿Que es un no-terminal?

En informática, un no terminal es un símbolo que representa un conjunto de cadenas de caracteres. Los no terminales se utilizan en gramáticas formales para definir la sintaxis de los lenguajes formales.

Un no terminal se representa generalmente con una letra mayúscula. Por ejemplo, el símbolo E podría representar el conjunto de todas las expresiones aritméticas.

Las reglas de una gramática formal definen cómo se pueden generar las cadenas de caracteres que representan los no terminales. Estas reglas se llaman producciones.

Por ejemplo, la siguiente producción define cómo se puede generar el no terminal E:

E -¿T + E — T

Esta producción significa que una expresión aritmética (E) puede ser una suma de dos términos (T + E) o un solo término (T).

Los no terminales son importantes porque permiten definir la sintaxis de los lenguajes formales de una manera jerárquica. Esto facilita la comprensión de la sintaxis de un lenguaje y la construcción de analizadores sintácticos que puedan verificar si una cadena de caracteres es válida para un lenguaje determinado.

Ejemplos de no terminales

Aquí hay algunos ejemplos de no terminales:

En la gramática de expresiones aritméticas, el símbolo E representa el conjunto de todas las expresiones aritméticas. En la gramática de oraciones en español, el símbolo SN representa el conjunto de todos los sintagmas nominales. En la gramática de programas en Python, el símbolo stmt representa el conjunto de todas las declaraciones.

Conclusiones

Los no terminales son una herramienta fundamental para la definición de la sintaxis de los lenguajes formales. Permiten definir la sintaxis de un lenguaje de una manera jerárquica y facilita la comprensión y la construcción de analizadores sintácticos.

A.3. ¿Como empiezo a desarrollar en JavaCC?

En el Apéndice B se encuentra una guía y manual para empezar a desarrollar sus proyectos con esta herramienta. En ella se encuentran contenidos de instalacion, configuración y primeros pasos.

A.4. ¿Donde puedo encontrar información adicional?

La mejor fuente de información sobre JavaCC es Internet. Hay muchos sitios web que ofrecen información sobre JavaCC, incluidos tutoriales, artículos y ejemplos.

Uno de los sitios web más útiles es la página web oficial de JavaCC. Esta página web contiene documentación completa sobre JavaCC, incluidas las especificaciones de la sintaxis de JavaCC, ejemplos de uso y una lista de preguntas frecuentes.

En la bibliografía de este documento se incluyen algunos libros y páginas web de referencia que pueden ser útiles para aprender más sobre JavaCC.

Estos recursos incluyen:

El libro "JavaCC: The Java Compiler Compiler"de Sanjiva Weerawarana El libro "JavaCC: A Tutorial"de Scott Ambler La página web de JavaCC Tutorial

Documentación oficial de JavaCC

La documentación oficial de JavaCC es una fuente importante de información sobre la herramienta. Esta documentación incluye las especificaciones de la sintaxis de JavaCC, ejemplos de uso y una lista de preguntas frecuentes.

Apéndice B

Herramientas utilizadas

B.1. Manual de instalación y configuración

En este apartado vamos a desarrollar los manuales necesarios para la instalación de la herramienta de JavaCC, el IDE de desarrollo que queramos utilizar, y el plugin de JavaCC para el IDE correspondiente.

B.1.1. Instalación de JavaCC

Se puede utilizar JavaCC directamente desde la línea de comandos, o puede optar por utilizar un IDE, que permite desarrollar proyectos con rapidez [1] e integra varias herramientas como para desarrollar proyectos de forma más eficiente y productiva.

Descarga e Instalación de JavaCC desde línea de comandos

Descarga

Descargue la última versión estable (al menos el código fuente y los binarios). Actualmente la versión estable más reciente es la 7.0.13, por lo que la descarga del binario se accedería a través de:

https://repo1.maven.org/maven2/net/java/dev/javacc/javacc/7.0.13/javacc.jar

y descargar el archivo javacc-7.0.13.jar

La descarga del código fuente se realiza a través del repositorio oficial de Java CC:

https://github.com/javacc/javacc/releases

Instalación

Una vez que haya descargado los archivos, navegue hasta el directorio de descarga y descomprima el archivo fuente, creando así el llamado directorio de instalación de JavaCC:

$unzip javacc - 7{,}0{,}13.zip$

o

$tar xvf javacc - 7{,}0{,}13.tar.gz$

A continuación, mueva el archivo binario bajo el directorio de descarga en un nuevo directorio bajo el directorio de instalación y cámbiele el nombre a javacc-7.0.13.jar target/javacc.jar

Posteriormente, añada el directorio del directorio de instalación de JavaCC a su archivo . Los scripts/ejecutables de invocación de JavaCC, JJTree y JJDoc residen en este directorio scripts/PATH En los sistemas basados en UNIX, es posible que los scripts no se puedan ejecutar inmediatamente. Esto se puede resolver usando el comando del directorio: javacc-7.0.13/ chmod +x scripts/javacc

Descarga e Instalación de JavaCC desde un IDE

Para poder usar JavaCC en un IDE se necesita como mínimo que el IDE tenga soporte para Java, y soporte para Maven con Java. IDEs como IntelliJ o Eclipse son compatibles con JavaCC mediante la instalación de un complemento para su desarrollo.

Descarga de IntelliJ: https://www.jetbrains.com/idea/

Plugin IntelliJ JavaCC: https://plugins.jetbrains.com/plugin/11431-javacc/

Descarga de Eclipse: https://www.eclipse.org/ide/

Plugin Eclipse JavaCC: https://marketplace.eclipse.org/content/javacc-eclipse-plug

Para Maven, hay que añadir la siguiente dependencia al archivo pom.xml:

```
<dependency>
    <groupId>net.java.dev.javacc</groupId>
    <artifactId>javacc</artifactId>
    <version>7.0.13</version>
</dependency>
```

En el caso de utilizar un IDE, deberá descargarse el binario, al igual que si desease desarrollar JavaCC desde la línea de comandos.

Configuración de JavaCC en Eclipse

Una vez instalado JavaCC, para poder desarrollar proyectos utilizando la JavaCC en Eclipse hay que seguir los siguientes pasos:

1. Crear un proyecto nuevo o elegir uno existente.
2. Abrir las propiedades del proyecto (Alt+Enter)
3. Ir a JavaCC ¿Global Options, y en “Set the default JavaCC jar file”, poner la ruta al binario descargado anteriormente (javacc.jar)

IMPORTANTE: Si mueve el binario descargado a la ruta por defecto, no es necesario realizar el paso 3. Es recomendable poner el binario en la ruta por defecto ya que, de lo contrario, para cada proyecto que quiera desarrollar va a tener que realizar este procedimiento. La ruta por defecto se indica debajo de “Set the default JavaCC jar file” del paso 3, (default: plugin’s jar).

Si la configuración se ha realizado correctamente, a la hora de guardar los archivos JavaCC, se generarán la compilación de las clases Java correspondientes.

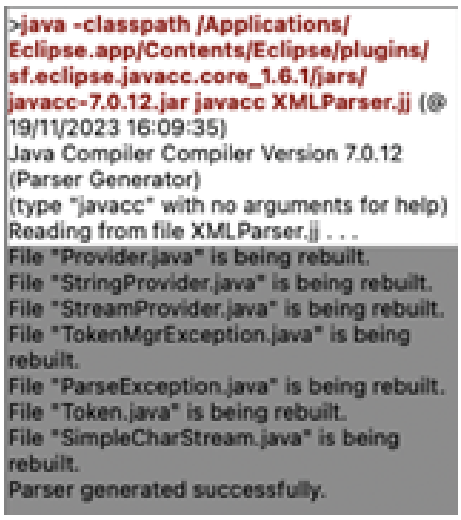

Figura B.1: Compilación de Archivos JavaCC en Eclipse

B.2. Símbolos de Expresiones regulares en JavaCC

‘ + ‘: El símbolo ‘+‘ se usa para indicar que un elemento puede aparecer una o más veces. Por ejemplo, si tienes ‘A+‘, significa que se espera que haya al menos una instancia de ‘A‘, pero puede haber más.

‘ ∗ ‘: El símbolo ‘ ∗ ‘ se usa para indicar que un elemento puede aparecer cero o más veces. Por ejemplo, si tienes ‘B ∗ ‘, significa que ‘B‘ es opcional y puede aparecer cero o más veces.

‘?‘: El símbolo ‘?‘ se usa para indicar que un elemento puede aparecer cero o una vez. Por ejemplo, si tienes ‘C?‘, significa que ‘C‘ es opcional y puede aparecer cero o una vez.

∼ : El símbolo ‘ ∼ ‘ se utiliza para excluir ciertos caracteres o elementos. Por ejemplo, ‘ ∼ A‘ significa çualquier cosa excepto ‘A‘. En una expresión regular, ‘ ∼ ‘ se usa para negar un conjunto de caracteres. Por ejemplo, ‘[∼ 0 − 9]‘ significa çualquier carácter que no sea un dígito del 0 al 9”.

Bibliografía

[1] *¿Qué es y para qué sirve un IDE?* Dirección: https://www.redhat.com/es/topics/middleware/what-is-ide.

[2] S. Viswanadha y S. Sankar. «JavaCC. The most popular parser generator for use with Java applications.» (1996), dirección: https://javacc.github.io/javacc/.

[3] A. V. Aho, *Compiladores : principios, técnicas y herramientas*. Pearson Educación, 2008, ISBN: 9789702611332.

[4] *introducción a JavaCC*. dirección: https://studylib.es/doc/6172839/introducci%C3%B3n-a-javacc.

[5] *Mi primer proyecto utilizando JavaCC - Erick Navarro*. dirección: https://www.ericknavarro.io/2020/02/10/23-Mi-primer-proyecto-utilizando-JavaCC/.

[6] *GitHub - RobertFischer/json-parser: JavaCC-built JSON Parser*. dirección: https://github.com/RobertFischer/json-parser.