Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Compiladores e Intérpretes

Proyecto
Sintaxis de MiniJava

Segundo Cuatrimestre de 2016

1. Introducción

Este documento describe la sintaxis de MiniJava. Como se menciona en el documento de especificación del proyecto, MiniJava es un lenguaje que puede verse como una simplificación de Java, donde por una parte no se consideran elementos avanzados como Genericidad, Excepciones o Hilos, y por otra parte la estructura de su sintaxis es más estricta.

En este documento, en primer lugar, se presentan los elementos léxicos correspondientes a la sintixis del lenguaje, donde en particular se identificarán las palabras reservadas, forma de los identificadores y literales del lenguaje. Luego se presentará la estructura sintáctica del lenguaje mediante su gramática; en esta última se podrán identificar especialmente aquellos elementos que alejan a MiniJava de Java.

2. Componentes Léxicos

El primer paso para compilar un programa de MiniJava radica en convertir una entrada de caracteres en una entrada de tokens, donde cada token se corresponderá con un lexema de MiniJava. Los lexemas de MiniJava son palabras clave, nombres de tipos primitivos, literales, símbolos de puntuación, operadores e identificadores. Cada uno de estos elementos se describe en las siguientes secciones.

2.1. Espacios en Blanco

Los espacios en blanco en MiniJava, al igual que en Java, simplemente hacen más fácil la lectura del programa por un humano y **no** son tokens. Un espacio en blanco es un espacio, tab o un salto de línea.

2.2. Palabras Clave

MiniJava tiene las siguientes palabras clave:

class	**extends**	**static**	**dynamic**	**public**	**private**
void	**boolean**	**char**	**int**	**String**	
if	**else**	**while**	**return**	**instanceof**	
this	**new**	**null**	**true**	**false**	

Note que varias de las palabras clave originales de Java no están presentes en MiniJava. Un compilador de MiniJava podría considerar esas palabras clave no presentes como prohibidas en MiniJava. Por otra parte, observe que MiniJava tiene palabras clave que no están presentes en Java, como por ejemplo **dynamic**. Estas palabras, como se podrá ver en la Sección 3, ayudarán a dar una estructura más estricta a los programas MiniJava.

2.3. Identificadores

En MiniJava hay dos tipos de identificadores: los identificadores de clases y los identificadores de métodos y variables. Un identificador de clase es una letra mayúscula seguida de cero o más letras

(mayúsculas o minúsculas), dígitos y underscores. Un identificador de método y variable es una letra minúscula seguida de cero o más letras (mayúsculas o minúsculas), dígitos y underscores.

En la Sección 3 los indentificadores de clase son denotados como **idClase**, mientras que los identificadores de metodos y variables son denotados como **IdMetVar**. Ningún identificador puede ser una palabra reservada.

2.4. Comentarios

Los comentarios son ignorados por el analizador léxico. MiniJava, al igual que Java, tiene dos estilos de comentarios:

- `/* comentario */` Comentarios multi-línea: Todos los caracteres desde `/*` hasta `*/` son ignorados.
- `// comentario` Comentario simple: Todos los caracteres de desde `//` hasta el final de la línea son ignorados.

2.5. Literales

En MiniJava hay cinco tipos de literales, los cuales se corresponden a los tipos primitivos del lenguaje.

2.5.1. Literales Enteros

Un literal entero es una secuencia de uno o más dígitos. El valor de un literal entero corresponde a su interpretación estándar en base 10.

2.5.2. Literales Caracteres

Un literal caracter puede ser:

- 'x' donde x es cualquier caracter excepto la barra invertida (\), el salto de línea, o la comilla simple ('). El valor del literal es el valor del caracter x.
- '\x' donde x es cualquier caracter excepto `n` o `t`. El valor del literal es el valor del caracter x.
- '`\t`'. Es valor del literal es el valor del caracter Tab.
- '`\n`'. Es valor del literal es el valor del caracter salto de línea.

2.5.3. Literales String

Un literal string se representa mediante una comilla doble (") seguida de una secuencia de caracteres y finaliza con otra comilla doble ("). El valor del literal corresponde a la cadena de caracteres entre las comillas. Se restringe el uso del caracter de salto de línea y las comillas dobles como parte de la cadena de un literal string.

2.5.4. Literales Booleanos

Los literales booleanos se representan mediante las palabras reservadas **true** y **false**.

2.5.5. Literal Nulo

El literal nulo se representa mediante la plabra reservada **null**.

2.6. Puntuación

MiniJava cuenta con los siguientes símbolos de puntuación, utilizados para estructurar los programas:
( ) { } ; , . [ ]
En particular [y] en MiniJava, a diferencia de Java, son utilizados para denotar el operador de *casting*.

2.7. Operadores

El lenguaje MiniJava cuenta con los siguientes operadores:

>	<	!	**instanceof**
==	>=	<=	!=
+	-	*	/
&&	||	%	

2.8. Asignación

Al igual que en Java, en MiniJava el simbolo de asignación es el =.

2.9. Otros caracteres

Cualquier otra entrada que no comforme con las reglas anteriormente presentadas debe generar un error.

3. Gramática de MiniJava

Cualquier programa MiniJava sintácticamente válido debe ser producto de la gramática que se presenta en esta sección. La gramática sigue la notación BNF-extendida, donde:

terminal	es un símbolo terminal
$<Class>$	es un símbolo no terminal (además la primer letra es una mayúscula)
ϵ	representa la cadena vacía
X^*	representa cero o más ocurrencias de X
X^+	representa una o más ocurrencias de X
$X^?$	representa cero o una ocurrencia de X
$X \rightarrow Y$	representa una producción
$X \rightarrow Y \vert Z$	es una abreviación de $X \rightarrow Y$ o $X \rightarrow Z$

Producciones BNF-Extendida

Las producciones de la Gramática de MiniJava son:

$<Inicial> \rightarrow <Clase>^+$

$<Clase> \rightarrow$ **class idClase** $<Herencia>^?$ **{** $<Miembro>^*$ **}**

$<Herencia> \rightarrow$ **extends idClase**

$<Miembro> \rightarrow <Atributo> \mid <Ctor> \mid <Metodo>$

$<Atributo> \rightarrow <Visibilidad> <Tipo> <ListaDecVars>$ **;**

$<Metodo> \rightarrow <FormaMetodo> <TipoMetodo>$ **idMetVar** $<ArgsFormales> <Bloque>$

$<Ctor> \rightarrow$ **idClase** $<ArgsFormales> <Bloque>$

<*Visibilidad*> → **public** | **private**

<*ArgsFormales*> → **(** <*ListaArgsFormales*>$^{?}$ **)**

<*ListaArgsFormales*> → <*ArgFormal*>
<*ListaArgsFormales*> → <*ArgFormal*> **,** <*ListaArgsFormales*>

<*ArgFormal*> → <*Tipo*> **idMetVar**

<*FormaMetodo*> → **static** | **dynamic**

<*TipoMetodo*> → <*Tipo*> | **void**

<*Tipo*> → <*TipoPrimitivo*> | **idClase**

<*TipoPrimitivo*> → **boolean** | **char** | **int** | **String**

<*ListaDecVars*> → **idMetVar**
<*ListaDecVars*> → **idMetVar** **,** <*ListaDecVars*>

<*Bloque*> → **{** <*Sentencia*>* **}**

<*Sentencia*> → **;**
<*Sentencia*> → <*Asignacion*> **;**
<*Sentencia*> → <*SentenciaLlamada*> **;**
<*Sentencia*> → <*Tipo*> <*ListaDecVars*> **;**
<*Sentencia*> → **if (** <*Expresion*> **)** <*Sentencia*>
<*Sentencia*> → **if (** <*Expresion*> **)** <*Sentencia*> **else** <*Sentencia*>
<*Sentencia*> → **while (** <*Expresion*> **)** <*Sentencia*>
<*Sentencia*> → <*Bloque*>
<*Sentencia*> → **return** <*Expresion*>$^{?}$ **;**

<*Asignacion*> → <*Primario*> **=** <*Expresion*>

<*SentenciaLlamada*> → <*Primario*>

<*Expresion*> → <*ExpOr*>
<*ExpOr*> → <*ExpOr*> || <*ExpAnd*> | <*ExpAnd*>
<*ExpAnd*> → <*ExpAnd*> **&&** <*ExpIg*> | <*ExpIg*>
<*ExpIg*> → <*ExpIg*> <*OpIg*> <*ExpComp*> | <*ExpComp*>
<*ExpComp*> → <*ExpAd*> <*OpComp*> <*ExpAd*> | <*ExpAd*> **instanceof idClase** | <*ExpAd*>
<*ExpAd*> → <*ExpAd*> <*OpAd*> <*ExpMul*> | <*ExpMul*>
<*ExpMul*> → <*ExpMul*> <*OpMul*> <*ExpUn*> | <*ExpUn*>
<*ExpUn*> → <*OpUn*> <*ExpUn*> | <*ExpCast*>
<*ExpCast*> → **[idClase]** <*Operando*> | <*Operando*>

<*OpIg*> → **==** | **!=**
<*OpComp*> → **<** | **>** | **<=** | **>=**
<*OpAd*> → **+** | **-**
<*OpUn*> → **+** | **-** | **!**
<*OpMul*> → ***** | **/** | **%**

<*Operando*> → <*Literal*>
<*Operando*> → <*Primario*>

<*Literal*> → **null** | **true** | **false** | **intLiteral** | **charLiteral** | **stringLiteral**

<*Primario*> → **(** <*Expresion*> **)** <*LlamadaoIdEncadenado*>*

$<Primario> \rightarrow$ **this** $<LlamadaoIdEncadenado>^*$
$<Primario> \rightarrow$ **idMetVar** $<LlamadaoIdEncadenado>^*$
$<Primario> \rightarrow$ **(** **idClase** **.** $<Llamada>$ **)** $<LlamadaoIdEncadenado>^*$
$<Primario> \rightarrow$ **new idClase** $<ArgsActuales>$ $<LlamadaoIdEncadenado>^*$
$<Primario> \rightarrow <Llamada> <LlamadaoIdEncadenado>^*$

$<LlamadaoIdEncadenado> \rightarrow$ **.** $<Llamada>$
$<LlamadaoIdEncadenado> \rightarrow$ **.** **idMetVar**

$<Llamada> \rightarrow$ **idMetVar** $<ArgsActuales>$

$<ArgsActuales> \rightarrow$ **(** $<ListaExps>^?$ **)**

$<ListaExps> \rightarrow <Expresion>$
$<ListaExps> \rightarrow <Expresion>$ **,** $<ListaExps>$