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

Compiladores e Intérpretes

Manual de Usuario

García Amado Andrés
LU: 89522

Índice

Introducción:

En este informe contiene todo lo necesario para que un usuario pueda usar el compilador de MiniJava y ejecutar su archivo mediante la máquina virtual CeIVM. El compilador de MiniJava tiene la función de traducir el código, además de realizar todos los controles, ya sean léxicos, sintácticos y semánticos del programa fuente. Para su ejecución, el código debe ser escrito en un archivo de salida que luego podrá ser utilizado por el CeIVM.

Características del lenguaje MiniJava:

- No presenta elementos avanzados como son genericidad, manejo de excepciones o hilos, variables de clase, métodos o clases abstractas, etc.
- No se permite declarar ni interfaces, ni clases abstractas.
- No se puede manejar empaquetados ni importaciones.
- Todas las declaraciones se realizan en un único archivo fuente y todas las clases coexisten dentro del mismo espacio de nombres. También, todas las clases son visibles y no se permite tener dos o más clases con el mismo nombre o que contengan el nombre Object o System. Por último, solo puede haber un método main.
- El método main no contiene parámetros, es de tipo void y de forma estática.
- No se puede declarar, dentro de la misma clase, métodos con el mismo nombre, inclusive si estos difieren en la cantidad de parámetros.
- Si se permite sobreescribir los métodos de clases heredadas. Pero siempre y cuando tengan el mismo nombre, cantidad y tipos de parámetros, mismo resultado y forma de método (static o dynamic).
- Se permite la herencia simple entre clases.
- Todas las clases, tienen como heredero común Object, salvo él mismo.
- Se cuenta con las clases predefinidas Object y System:
    - Object: La superclase de todas clases de MiniJava. Que no posee ni métodos ni atributos.
    - System: Que contiene todos los métodos necesarios para la entrada/salida de valores. Es similar a la clase en Java java.lang.System pero solo brinda acceso a los streams de entrada (System.in) y salida (System.out), pero lo hace utilizando los siguientes métodos:
        - static int read(): lee el próximo byte del stream de entrada est´andar (originalmente en la clase java.io.InputStream).
        - static void printB(boolean b): imprime un boolean por salida est´andar (originalmente en la clase java.io.PrintStream).
        - static void printC(char c): imprime un char por salida est´andar (originalmente en la clase java.io.PrintStream).
        - static void printI(int i): imprime un int por salida est´andar (originalmente en la clase java.io.PrintStream).
        - static void printS(String s): imprime un String por salida est´andar (originalmente en la clase java.io.PrintStream).
        - static void println(): imprime un separador de línea por salida est´andar, finalizando la linea actual (originalmente en la clase java.io.PrintStream).
        - static void printBln(boolean b): imprime un boolean por salida est´andar y finaliza la línea actual (originalmente en la clase java.io.PrintStream).
        - static void printCln(char c): imprime un char por salida est´andar y finaliza la línea actual (originalmente en la clase java.io.PrintStream).

    - static void printlln(int i): imprime un int por salida estándar y finaliza la línea actual (originalmente en la clase java.io.PrintStream).
    - static void printSln(String s): imprime un String por salida estándar y finaliza la línea actual (originalmente en la clase java.io.PrintStream).
- MiniJava contiene dos tipos de indicadores: Los de clase que comienzan con mayúscula y los de métodos y variables que comienzan con minúscula.
- Se pueden declarar atributos de instancia, constructores y métodos. Una clase sólo puede tener un único constructor, (y en caso de no tener uno declarado de manera explícita, el compilador le asignará uno sin argumentos y de cuerpo vacío) y no puede tener más de una variable de instancia declarada con el mismo nombre. Al igual que en Java, es posible declarar una variable con el mismo nombre que una definida en una superclase, “tapando” la misma.
- Se cuenta con cinco tipos de literales, los cuales se corresponden a los tipos primitivos del lenguaje (int, char, boolean). Además se cuenta con el tipo void, String y el programador puede declarar tipos clase.
- Además cuenta con los tipos Arreglos de los tipos primitivos (int, char, boolean).
- Palabras reservadas:
    - class
    - extends
    - static
    - dynamic
    - String
    - boolean
    - char
    - int
    - public
    - private
    - void
    - null
    - if
    - else
    - while
    - return
    - this
    - new
    - true
    - false
- MiniJava cuenta con los siguientes símbolos de puntuación, utilizados para estructurar los programas:
  ( ) { } ; , . [ ]
- El lenguaje MiniJava cuenta con los siguientes operadores:
  > < !
  == >= <= !=
  + - * /
  && ||

Gramática MiniJava

```
<Inicial> -> <Clase>

<Clase> -> class idClase <Herencia> { <Miembro> }

<Herencia> -> extends idClase

<Miembro> -> <Atributo>
<Miembro> -> <Ctor>
<Miembro> -> <Metodo>

<Atributo> -> <Visibilidad> <Tipo> <ListaDecVars> ;

<Metodo> -> <FormaMetodo> <TipoMetodo> idMetVar <ArgsFormales> <Bloque>

<Ctor> -> idClase <ArgsFormales> <Bloque>

<ArgsFormales> -> <ArgFormal>

<ListaArgsFormales> -> <ArgFormal> , <ListaArgsFormales>

<ArgFormal> -> <Tipo> idMetVar

<FormaMetodo> -> static
<FormaMetodo> -> dynamic

<Visibilidad> -> public
<Visibilidad> -> private

<TipoMetodo> -> <Tipo>
<TipoMetodo> -> void

<Tipo> -> boolean
<Tipo> -> char
<Tipo> -> int
<Tipo> -> boolean[]
<Tipo> -> char[]
<Tipo> -> int[]

<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> -> <AccesoVar> = <Expresion>
<Asignacion> -> <AccesoThis> = <Expresion>

<SentenciaLlamada> -> ( <Primario> )

<Expresion> -> <ExpOr>

<ExpOr> -> <ExpOr> || <ExpAnd>
<ExpOr> -> <ExpAnd>

<ExpAnd> -> <ExpAnd> && <ExpIg>
<ExpAnd> -> <ExpIg>

<ExpIg> -> <ExpIg> <OpIg> <ExpComp>
<ExpIg> -> <ExpComp>

<ExpComp> -> <ExpAd> <OpComp> <ExpMul>
<ExpComp> -> <ExpMul>

<ExpAd> -> <ExpAd> <OpAd> <ExpMul>
<ExpAd> -> <ExpMul>

<ExpMul> -> <ExpMul> <OpMul> <ExpUn>
<ExpMul> -> <ExpUn>

<ExpUn> -> <OpUn> <ExpUn>
<ExpUn> -> <Operando>

<OpIg> -> ==
<OpIg> -> !=

<OpComp> -> <
```

```
<OpComp> -> >
<OpComp> -> <=
<OpComp> -> >=

<OpAd> -> +
<OpAd> -> -

<OpUn> -> +
<OpUn> -> -
<OpUn> -> !

<OpMul> -> *
<OpMul> -> /

<Operando> -> <Literal>
<Operando> -> <Primario>

<Literal> -> null
<Literal> -> true
<Literal> -> false
<Literal> -> intLiteral
<Literal> -> charLiteral
<Literal> -> StringLiteral

<Primario> -> <ExpresionParentizada>
<Primario> -> <AccesoThis>
<Primario> -> <AccesoVar>
<Primario> -> <LlamadaMetodo>
<Primario> -> <LlamadaMetodoEstatico>
<Primario> -> <LlamadaCtor>

<ExpresionParentizada> -> ( <Expresion> ) <Encadenado>

<AccesoThis> -> this <Encadenado>

<AccesoVar> -> idMetVar <Encadenado>

<LlamadaMetodo> -> idMetVar <ArgsActuales> <Encadenado>

<LlamadaMetodoEstatico> -> idClase <ArgsActuales> <Encadenado>

<LlamadaCtor> -> new idClase <ArgsActuales> <Encadenado>
<LlamadaCtor> -> new <TipoPrimitivo> [ <Expresion> ] <Encadenado>

<ArgsActuales> -> ( <ListaExps> )
```

```
<ListaExps> -> <Expresion>
<ListaExps> -> <Expresion> , <ListaExps>

<Encadenado> -> . <LlamadaMetodoEncadenado>
<Encadenado> -> . <AccesoVarEncadenado>
<Encadenado> -> <AccesoArregloEncadenado>

<LlamadaMetodoEncadenado> -> idMetVar <ArgsActuales> <Encadenado>

<AccesoVarEncadenado> -> idMetVar <Encadenado>

<AccesoArregloEncadenado> -> [ <Expresion> ] <Encadenado>
```

Modo de uso del compilador:

El compilador se encuentra como ejecutable java (JAR). Por eso para poder ejecutarlo, se debe utilizar un terminal e ingresar el siguiente comando:

java –jar compilador.jar <in_File> [<out_File>]

Donde el compilador.jar es el nombre del archivo ejecutable. in_file es el archivo fuente y el out_file es un parámetro opcional, que de ser usado, se crea un archivo con el código generado correspondiente, de ser ignorado el análisis solo mostrará en consola.

Errores:

Con respecto a los errores durante la compilación, estos se manejaron a través de excepciones. Una vez que se encuentre un error, ya sea léxico, sintáctico o semántico, la acción del compilador es marcar el error, de qué tipo es y por qué se produjo, y abortar el proceso de compilación. El mensaje de error se muestra por pantalla.

Decisiones de diseño:

No se permite utilizar el nombre ‘Main’ para las clases, si bien los controles en la etapa semántica lo permiten, el generador de código determina un error.

Ejemplos:

Creación de una clase:

La creación de la clase Dos con el constructor predefinido.

```
class Dos{

}
```

La creación de la clase Dos con el constructor.

```
class Dos{

	Dos(){
	}

}
```

La creación de la clase Dos con constructor con atributos.

```
class Dos{
	private int a,
	private int b;
	public boolean c;

	Dos(int x, int y){
		a = x;
		b = y;
		c = false;
	}

}
```

Métodos:

Creación y llamada a un método.

```
class A {

	A() {}

	dynamic void a1() {
		(System.printlln(1));
	}

	dynamic void m1() {
		(System.printlln(2));
	}

}
```

Sentencia While:

Creación de un While: El while utiliza la variable i para determinar la condición de corte. y dentro del bloque del mismo imprime el valor de i y le suma 1.

```
static void main() {
		int i;
		i = 0;
		while (i < 10) {
			(System.printlln(i));
			i = i + 1;
		}
	}
```

Sentencia IF-ELSE

Creación de IF: En el caso de que la variable b sea verdadero retorna 1, sino continúa con la ejecución del resto de sentencias dentro del método m1.

```
static int m1(boolean b) {
		if (b) {
			return 1;
		}
		int a;
```

```
                a = 3;
                return 3;
        }
}
```

Creación de IF-ELSE: En el caso de que la variable b sea verdadero retorna 1, sino ejecuta la sentencia que está dentro del else y luego continúa con la ejecución del resto de sentencias dentro del método m1.

```
static int m1(boolean b) {
                int a;
                a = 1;
                if (b) {
                        return 1;
                }
                else{
                      a = 3;
                    }

                return a;
        }
}
```

Manejo de Arreglos:

Creación, asignación y utilización de Arreglos:

```
static void m1(int a){
                int [] arregloInt;
                char [] arregloChar;
                boolean [] arregloBool;

                arregloInt = new int[a];
                arregloChar = new char[a];
                arregloBool = new boolean[a];

                arregloInt[0] = 0;

                arregloChar[0] = 'a';


                arregloBool[0] = true;
```

```
        int i;
        i = 0;

        while(i < a){
                (System.printlln(arregloInt[i]));
                (System.printCln(arregloChar[i]));
                (System.printBln(arregloBool[i]));
                i = i+1;
        }

}
```