[MiniJava]

Manual de usuario

Ramiro Agis - LU. 96248
Victoria Martínez de la Cruz - LU. 87620

Índice

Introducción

En el presente manual de usuario se detallará toda la información de interés para los programadores que utilicen el compilador de MiniJava y el intérprete CeIVM.
Primero se presentará la estructura sintáctica del lenguaje MiniJava mediante la especificación de su gramática, que luego será acompañada de explicaciones y ejemplos de uso. Además se explicará el modo de uso para ejecutar el compilador y el intérprete CeIVM, y se detallarán todas las limitaciones y decisiones del lenguaje, del compilador o del intérprete que los programadores deben conocer para hacer un uso correcto de éstas herramientas.

Gramática

Cualquier programa MiniJava sintácticamente válido debe ser producto de la gramática que se presentará a continuación Ésta sigue la notación BNF-extendida, donde:

terminal	es un símbolo terminal
<*NoTerminal*>	es un símbolo no terminal
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 → Y	representa una producción
X → Y | Z	es una abreviación de X → Y o X → Z

Las producciones de la gramática de MiniJava son:

<Inicial> → *<Clase>*$^+$

<Clase> → **class** *identificador* *<Herencia>*$^?$ **{** *<Miembro>** **}**

<Herencia> → **extends** *identificador*

<Miembro> → *<Atributo>* | *<Ctor>* | *<Metodo>*

<Atributo> → **var** *<Tipo>* *<ListaDeVars>* **;**

<Metodo> → *<ModMetodo>* *<TipoMetodo>* *identificador* *<ArgsFormales>* *<VarsLocales>* *<Bloque>*

<Ctor> → *identificador* *<ArgsFormales>* *<VarsLocales>* *<Bloque>*

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

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

<ArgFormal> → *<Tipo>* *identificador*

<VarsLocales> → *<Atributo>**

<ModMetodo> → **static** | **dynamic**

<TipoMetodo> → *<Tipo>* | **void**

<Tipo> → *<TipoPrimitivo>* | *identificador*

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

<ListaDecVars> → *identificador*
<ListaDecVars> → *identificador* **,** *<ListaDecVars>*

<Bloque> → **{** *<Sentencia>** **}**

<*Sentencia*> → **;**
<*Sentencia*> → <*Asignacion*> **;**
<*Sentencia*> → <*SentenciaSimple*> **;**
<*Sentencia*> → **if (** <*Expresion*> **)** <*Sentencia*>
<*Sentencia*> → **if** (<*Expresion*> **)** <*Sentencia*> **else** <*Sentencia*>
<*Sentencia*> → **while** (<*Expresion*> **)** <*Sentencia*>
<*Sentencia*> → **for (** <*Asignacion*> **;** <*Expresion*> **;** <*Expresion*> **)** <*Sentencia*>
<*Sentencia*> → <*Bloque*>
<*Sentencia*> → **return** <*Expresion*>? **;**

<*Asignacion*> → *identificador* **=** <*Expresion*>

<*SentenciaSimple*> → **(** <*Expresion*> **)**

<Expresion> → <*Expresion*> <*OperadorBinario*> <*Expresion*>
<*Expresion*> → <*OperadorUnario*> <*Expresion*>
<*Expresion*> → <*Primario*>

<*OperadorBinario*> → && | || | == | != | < | > | >= | <= | * | / | % | + | -
<*OperadorUnario*> → ! | + | -

<*Primario*> → **this**
<*Primario*> → <*Literal*>
<*Primario*> → **(** <*Expresion*> **)** <*Llamada*>*
<*Primario*> → *identificador* <*Llamada*>*
<*Primario*> → **new** *identificador* <*ArgsActuales*> <*Llamada*>*
<*Primario*> → *identificador* <*ArgsActuales*> <*Llamada*>*

<*Llamada*> → **.** *identificador* <*ArgsActuales*>

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

<*ArgsActuales*> → **(** <*ListaExps*>? **)**

<*ListaExps*> → <*Expresion*>
<*ListaExps*> → <*Expresion*> **,** <*ListaExps*>

Características del lenguaje y ejemplos de uso

Identificadores

Los identificadores en MiniJava son sensibles a las mayúsculas y minúsculas.
Un identificador puede contener:

- Cualquier caracter ASCII que sea una letra o un dígito.
- El separador _ .

Un identificador no puede:

- Comenzar con un dígito o el separador _ .
- Ser igual a una palabra reservada, al literal nulo *null* o a los literales booleanos *true* y *false*.

Palabras reservadas

boolean	char	class	dynamic	else
extends	for	if	int	new
return	static	String	this	var
void	while			

Literales

El conjunto ∑ de caracteres válidos en MiniJava está conformado por los caracteres imprimibles del ASCII básico y por tres caracteres de control: '\0', '\n' y '\t'.

∑ = { ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ '\0' '\n' '\t' <blank>}

Literal boolean	true | false
Literal char	x ∈ ∑ - { \, \n, ‘ } | \x ∈ ∑ - { ‘\n’, ‘\t’, ‘\0’ } | \t | \n
Literal int	[1-9][0-9]* | 0
Literal String	x ∈ ∑ - { \n, “ }
Literal null	null

Declaración de una clase

Las clases son los fundamentos de un lenguaje orientado a objetos como MiniJava. Estas contienen miembros que almacenan datos (variables de instancia) y manipulan datos (métodos). Pueden haber varias clases declaradas dentro de un mismo archivo fuente, cada una con su bloque de código correspondiente, pero no puede haber nombres de clase repetidos.

```
class Foo {

}

class Bar {

}
```

Variables

Las variables son identificadores asociados a valores. Son declaradas escribiendo la palabra reservada *var* seguida del tipo y nombre de la variable. Una variable sólo puede ser declarada al comienzo del bloque de una clase (variable de instancia)o luego de la definición de la signatura de un método (variable local). Además sólo pueden ser inicializadas dentro del bloque de un método o constructor. También pueden declararse variables en la signatura de un método (parámetros), pero no requiere la palabra reservada *var.*

```
class Clase {
	var char caracter; // Declaración de una variable de instancia.
	var int valor; // Declaración de otra variable de instancia.

	dynamic void bar(int a, int b) // Declaración de dos parámetros.
	var int numero; // Declaración de una variable local.
	{
		numero = 1; // Inicialización de la variable local.
	}
}
```

Múltiples variables del mismo tipo pueden ser declaradas en una misma sentencia usando la coma como delimitador.

```
var int variable1, variable2, variable3;
```

Bloques de código

Los separadores { y } son usados para indicar un bloque de código. Los miembros de una clase (variables de instancia, métodos y un constructor) y el cuerpo de un método se encuentran dentro de un bloque de código. Las sentencias de control *if-then, if-then-else, while* y *for* utilizan bloques de código para agrupar

el conjunto de sentencias que estas ejecutan.

```
class Foo {
	// Cuerpo de una clase.
	var int x;

	static void main() {
		// Cuerpo de un método.
	}
}
```

Comentarios

MiniJava tiene dos tipos de comentarios: comentarios de bloque y comentarios de fin de línea.
Los comentarios de bloque comienzan con /* y terminan con */ y pueden abarcar varias líneas de código.

```
/* Este es un comentario de bloque.
Puede abarcar más de una línea. */
```

Los comentarios de fin de línea comienzan con // y se extienden hasta el final de esa línea.

```
// Este es un comentario de fin de línea.
```

Estructura de un programa

Los programas MiniJava consisten de clases y sus miembros (variables de instancia, métodos y un constructor).
El punto de entrada de todo programa MiniJava es el método *main*. Puede haber más de una clase con un método *main*, pero por defecto se elegirá la primera aparición de *main* como punto de inicio de la ejecución del programa. Este método debe ser estático y nunca retorna un valor ya que su tipo de retorno debe ser *void.*

```
class Foo {
	var int variable1;
	var int variable2;

	dynamic void metodo1() {

	}

	dynamic void metodo2() {

	}

}
```

```
class Bar {
	static void main() {

	}
}
```

Operadores

Precedencia	Operador	Descripción	Asociatividad
1	() .	Invocación a un método	Izquierda a derecha
2	! new	Negación lógica Creación de instancia de clase	Derecha a izquierda
3	* / %	Multiplicación División Módulo	Izquierda a derecha
4	+ -	Suma Resta	Izquierda a derecha
5	< <= > >=	Relacional “menor que” Relacional “menor o igual que” Relacional “mayor que” Relacional “mayor o igual que”	Izquierda a derecha
6	== !=	Relacional “igual a” Relacional “no igual a”	Izquierda a derecha
7	&&	Conjunción lógica	Izquierda a derecha
8	||	Disyunción lógica	Izquierda a derecha
9	=	Asignación	-

Estructuras de control

Sentencia if-then

Las sentencias *if-then* evalúa una condición. Si la condición es *true*, las sentencias dentro del bloque *if* son ejecutadas. Caso contrario *(false)*, la ejecución continúa al final de la sentencia *if-then*.

```
if (i == 1)
	(hacerAlgo());

if (i == 2) {
```

```
        (hacerOtraCosa1());
        (hacerOtraCosa2());
}
```

Sentencias if-then-else

Las sentencias *if-then-else* también evalúa una condición, pero a diferencia de la sentencia *if-then*, si la condición es *false,* las sentencias dentro del bloque *else* son ejecutadas y luego la ejecución continúa al final de la sentencia *if-then-else.*

```
if (i == 3) {
        (hacerAlgo());
} else {
        (hacerOtraCosa());
}
```

Ciclos while

Los ciclos *while* evalúan una condición al comenzar una iteración. Si la condición es *true*, las sentencias dentro del bloque *while* son ejecutadas y el ciclo *while* realiza una nueva iteración. Caso contrario *(false)*, la ejecución continúa al final del ciclo *while.*

```
while (i < 10) {
        (hacerAlgo());
}
```

Ciclos for

Los ciclos *for* incluyen la inicialización de una variable, una condición y una expresión de incremento de la variable. Primero es ejecutada la inicialización de la variable y luego en cada iteración, si la condición es *true*, las sentencias dentro del bloque *for* son ejecutadas junto con la expresión de incremento y el ciclo *for* realiza una nueva iteración. Caso contrario *(false)*, la ejecución continúa al final del ciclo *for.*

```
for (i = 0; i < 10; i + 1) {
    (hacerAlgo());
}
```

Sentencias return

Las sentencias *return* son utilizadas para terminar la ejecución de un método y retornar un valor. El valor retornado por el método es escrito luego de la palabra reservada *return*. Si el tipo de retorno del método no es *void*, debería utilizar la sentencia *return* para retornar algún valor. Sin embargo, esto no es

controlado por MiniJava.
Una sentencia *return* termina la ejecución de un método inmediatamente, por lo tanto todas las sentencias que se encuentren a continuación (dentro del mismo bloque) nunca serán ejecutadas.

```
dynamic int metodo1()
var int x, y, z;
{
	x = 1;
	y = 2;
	z = 3;
	return x + y + z;
}

dynamic void metodo2()
var int x;
{
	x = 1;
	if (x > 2) {
		return;
	} else {
		x = x + 1;
		return;
	}
}
```

Modificadores de métodos

Los modificadores son palabras reservadas usadas para modificar las declaración de los métodos. El modificador *static* (estático) es usado para especificar que un método no pertenece a una instancia específica de la clase en la que está contenida. Un método estático es accesible sin la creación de una instancia de la clase. Sin embargo, los métodos estáticos no pueden llamar a miembros no estáticos de la misma clase.
El modificador *dynamic* (dinámico) es usado para especificar que un método sólo puede ser accedido enviando un mensaje a una instancia de la clase donde está definido el método. Esto sólo es necesario si el método no está definido en la clase del objeto que se realiza la llamada o si la llamada se realiza desde un método estático.

```
static void metodoEstatico() {

}

dynamic void metodoDinamico() {

}
```

Instanciación de una clase

Una clase puede ser instanciada para crear un objeto. Esto se hace utilizando la palabra reservada *new* y llamando al constructor de la clase.

```
Foo foo = new Foo();
```

Llamando a los métodos

Tanto los métodos dinámicos y estáticos son llamados utilizando el operador . (punto).
Ambos pueden ser llamados a través del nombre de una instancia de la clase en la que está definido el método. Si el método que se desea invocar está definido dentro de la clase del objeto llamador y la llamada se encuentra en un contexto dinámico, se puede hacer referencia a la instancia actual utilizando la palabra reservada *this* o simplemente indicando el objeto receptor del mensaje.

```
class EstaClase {
	dynamic void foo() {
		((this).bar()); // bar está definido en EstaClase, el objeto receptor es sí
mismo.
		(bar()); // Equivalente a la llamada ((this).bar());
	}

	dynamic void bar()
	var OtraClase oc;
	{
		oc = new OtraClase();
		(oc.ping()); // ping está definido en OtraClase, se necesita un objeto
receptor.
	}
}

class OtraClase {
	dynamic void ping() {

	}
}
```

Los métodos estáticos también pueden ser accedidos usando el nombre de la clase en donde está definido el método. No se requiere la creación de una instancia de la clase.

```
class EstaClase {
	static void main() {
		(OtraClase.ping()); // No es necesario un objeto receptor. Se llama a la
clase.
	}
```

```
}

class OtraClase {
	static void ping() {

	}
}
```

Visibilidad de los miembros de una clase

Se dice que un miembro de una clase es *visible* si puede ser accedido desde cierto punto del programa.
Las variables de instancia sólo son visibles dentro del contexto de la clase a la que pertenecen. Las variables locales y parámetros sólo son visibles dentro del bloque de código del método en el que están declaradas. Los métodos de las clases son visibles desde cualquier punto del programa.

Métodos

Todas las sentencias en MiniJava residen dentro de los métodos. Los métodos son similares a las funciones excepto que pertenecen a clases. Un método tiene un modificador, un valor de retorno, un nombre y usualmente algunos parámetros que son inicializados con los argumentos de la llamada al método. El nombre de un método debe ser único dentro de su clase ya que MiniJava no soporta sobrecarga de métodos. Los métodos que no retornan nada tienen el tipo de retorno declarado como *void*.

```
class Sumador {
	dynamic int sumar(int x, int y) {
		return x + y:
	}
	dynamic int foo() {
		(sumar(10, 20));
	}
}
```

Constructores

Un constructor es un método especial llamado para crear un instancia de una clase u *objeto*. Su propósito principal, además de crear la instancia, es inicializar su estado interno (variables de instancia).
Las diferencias principales entre los constructores y los métodos ordinarios es que los constructores son sólo llamados cuando una instancia de una clase es creada y nunca retornan algo. Los constructores son declarados como métodos comunes, pero tienen el mismo nombre que la clase y no se especifica su tipo de retorno ni modificador.
Un constructor puede tener cualquier cantidad de argumentos, pero puede definirse uno sólo por clase.

En caso de no ser definido, se le asigna a la clase un constructor por defecto con cuerpo vacío.

```
class Punto {
	var int x, y, z;

	// Constructor de la clase Punto.
	Punto() {
		x = 1;
		y = 2;
		z = 3;
	}
}
```

Herencia

Las clases en MiniJava pueden extender a otra clase para heredar o copiar todos sus métodos. La herencia es declarada usando la palabra reservada *extends*. Si una clase no especifica una superclase, esta hereda implícitamente de la clase *Object*. Por lo tanto todas las clases en MiniJava son subclases de la clase *Object*.

Redefinición de métodos

Una clase puede redefinir o sobreescribir un método de su superclase definiéndolo con exactamente la misma signatura. Sin embargo en MiniJava no es posible sobrecargar métodos, por lo tanto no está permitido definir en una clase un método con el mismo nombre que un método de su superclase pero con diferente modificador, tipo de retorno o cantidad o tipo de parámetros.

```
class Superclase {
	dynamic void metodo(int a, int b) {
		return 0;
	}
}

class Subclase extends Superclase {

	// metodo de Subclase redefine a método de Superclase.
	dynamic void metodo(int a, int b) {
		return 1;
	}
}
```

Tipo estático y dinámico de una variable

Si una clase B extiende a una clase A, entonces una variable de tipo A también puede contener una

referencia a un objeto de una clase B.

```
var A foo;
foo = new B();
```

A es el tipo estático de la variable *foo* y *B* es su tipo dinámico. El tipo estático siempre se determina en tiempo de compilación mientras que el tipo dinámico en general sólo se puede conocer en tiempo de ejecución y puede cambiar.
MiniJava sólo permite invocar a los métodos que están definidos en el tipo estático de una variable. Del mismo modo, MiniJava sólo permite asignar una expresión a una variable de tipo *A* si el tipo de la expresión es *A* o una subclase de *A*.

Ligadura dinámica de código

Al enviar un mensaje a una variable objeto, se invoca el método definido para el tipo dinámico de la variable. Es decir la clase más específica a la cual pertenece el objeto referenciado por la variable.

```
class A {
	dynamic int foo() {
		return 0;
	}
}

class B extends A {
	// Redefinición del método foo de la clase A.
	dynamic int foo() {
		return 777;
	}
}

class Principal {
	static void main()
	var A a;
	var int x;
	{
		a = new B();
		x = (a.foo()); // El valor de x será 777 porque se ejecutará el método foo
de B.
	}
}
```

Métodos predefinidos por el lenguaje

MiniJava brinda una clase predefinida *System* que contiene varios métodos útiles para realizar la lectura y escritura por consola.
Estos métodos son:

Método	Descripción
System.read	Lee un valor entero por consola.
System.printI	Imprime en la consola un valor entero.
System.printC	Imprime en la consola un caracter.
System.printB	Imprime en la consola un valor booleano.
System.printS	Imprime en la consola un String.
System.println	Imprime en la consola el caracter de nueva línea.
System.printIln	Imprime en la consola un valor entero seguido del caracter de nueva línea.
System.printCln	Imprime en la consola un caracter seguido del caracter de nueva línea.
System.printBln	Imprime en la consola un valor booleano seguido del caracter de nueva línea.
System.printSln	Imprime en la consola un String seguido del caracter de nueva línea.

Modo de uso

Requerimientos del sistema

- Java Runtime Environment 7
- Ejecutable de la máquina virtual de Compiladores e Intérpretes (CeIVM) v1.1.0 (CeIVM-cei2011.jar)

Compilación

Para ejecutar un programa MiniJava se debe abrir una terminal e ingresar el siguiente comando:

```
java -jar Minijava.jar <ARCHIVO_ENTRADA> [<ARCHIVO_SALIDA>]
```

Donde,
<ARCHIVO_ENTRADA> es el nombre o ruta del archivo fuente de MiniJava a compilar. Si se ingresa sólo el nombre del archivo, éste debe estar ubicado dentro de la misma carpeta donde se encuentra el jar del compilador.

[<ARCHIVO_SALIDA>] es un parámetro opcional que representa el nombre o ruta del archivo de código intermedio generado por el compilador. Si se omite este parámetro, se generará la salida en un archivo con un nombre igual al del archivo de entrada seguido de ".CeIASM.txt". El archivo de salida será generado en la misma carpeta en que se encuentra el jar del compilador.

Se espera que las rutas indicadas sean rutas absolutas. De ingresarse un nombre o ruta de archivo inválida o inexistente, se notificará al usuario con un error al intentar abrir el archivo.
Si la compilación del programa es exitosa, el compilador mostrará un mensaje informando esta situación y generará el archivo de código intermedio. En cambio, si se produce algún error durante el proceso de compilación, la salida del compilador será un mensaje informando el error y el número de línea donde se ha producido.

Ejemplos de uso:

```
java -jar MiniJava.jar input.txt output.txt
java -jar MiniJava.jar /home/minijava /home/ceiasm
java -jar MiniJava.jar input.txt (en este caso el nombre del archivo de salida
será input.txt.CEIASM.txt).
```

Ejecución

Para ejecutar ejecutar el archivo de código intermedio CeIASM generado por el compilador en la máquina virtual CeIVM se debe ingresar en la terminal el siguiente comando:

```
java -jar CeIVM-cei2011.jar <ARCHIVO_ORIGEN> [-v [<ARCHIVO_LISTADO>]]
```

Donde,
<ARCHIVO_ORIGEN> es el nombre o ruta del archivo que contiene el código intermedio a ejecutar. Éste debe estar ubicado dentro de la misma carpeta donde se encuentra el jar de la máquina virtual. Este archivo de origen debería ser el archivo de salida de la compilación previamente hecha.

[-v] es un parámetro opcional que genera listados con información de ensamblado, linkeo y carga en un archivo con un nombre igual del archivo de origen seguido de ".list.txt".

[-v [<ARCHIVO_LISTADO>]] es un parámetro opcional que genera listados con información de ensamblado, linkeo y carga en un archivo con el nombre <ARCHIVO_LISTADO>.

Se espera que las rutas indicadas sean rutas absolutas. De ingresarse un nombre o ruta de archivo inválida o inexistente, se notificará al usuario con un error al intentar abrir el archivo.
Si la ejecución del programa es exitosa, se mostrarán por consola todas las salidas por pantalla del programa seguidas de un mensaje informando la ejecución exitosa. En cambio, si se produce algún error durante la ejecución, se mostrará por pantalla un mensaje informando el error.

Ejemplos de uso:

```
java -jar CeIVM-cei2011.jar input.CeIASM.txt
java -jar CeIVM-cei2011.jar input.CeIASM.txt -v
java -jar CeIVM-cei2011.jar input.CeIASM.txt -v listado.txt
```

Decisiones de diseño y limitaciones

Lenguaje

- MiniJava es sensible a las mayúsculas y minúsculas (*casesensitive*).

- No es posible declarar variables de instancia cuyo nombre sea igual al nombre de la clase en la que están definidas o igual al nombre de los métodos definidos dentro de la clase. Por ejemplo:

```
class Foo {
   var int Foo;

   static void main() {

   }
}
```

Este programa tendrá un error en compilación ya que la variable *Foo* tiene el mismo nombre que su clase.

```
class Bar {
   var int foo;

   static void foo {

   }

   static void main() {

   }
}
```

Este programa tendrá un error en compilación ya que la variable *foo* tiene el mismo nombre que el método *foo* que está definido dentro de su clase.

- Pueden existir varias clases con el método principal *main*. Por defecto se elegirá la primera aparición de *main* como punto de inicio de la ejecución. Por ejemplo:

```
class Foo {
    static void main() {

   }
}

class Bar {
```

```
    static void main() {

    }
}
```

En este caso, el método principal del programa será el método *main* de la clase *foo*.

- Dado que la clase *System* es una clase constituida únicamente de variables y métodos estáticos, no es posible:
  - Extender a la clase System.
  - Crear instancias de la clase System.
  - Declarar variables de tipo System.
  - Definir métodos con tipo de retorno System.

- MiniJava no soporta la sobrecarga de métodos dentro de una misma clase ni mediante herencia.

- Los métodos estáticos pueden ser invocados directamente si están definidos en la misma clase en la que se invocan, e.g. métodoEstaticoEnClase(), o a través del operador . (punto) teniendo como lado izquierdo:
  - El nombre de la clase en el que está definido el método estático. Ejemplo:
    ```
    (A.metodoEstaticoEnA())
    ```

  Si bien es desaconsejado debido a que se pierde la noción de métodos de clase, también puede tenerse como lado izquierdo:
  - Una instancia de la clase en la que está definido el método estático. Ejemplo:
    ```
    var A a1 = new A(); (a1.metodoEstaticoEnA());
    ```

  - Una expresión que retorne una instancia de la clase en la que está definido el método estático. Ejemplo:
    ```
    ((new A()).metodoEstaticoEnA()); ((this).metodoEstaticoEnA())
    ```

  Naturalmente también es posible realizar llamadas encadenadas con método estáticos y dinámicos. Esta decisión de diseño se basa en los lineamientos de Java SE 7.
  (ver http://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html).

Compilador

- El compilador no realiza ningún control sobre sentencias inalcanzables, caminos sin retorno o múltiples retornos. Por ejemplo:

```
static int sinRetorno() {

}
```

La definición de este método es válida y su invocación retorna 0.

```
static int sentenciaInalcanzable()
var x;
{
   return 1;
   x = x + 1;
}
```

La invocación a este método retorna 1 y el compilador no informará que la sentencia `x = x + 1` es una sentencia inalcanzable.

```
static int multiplesRetornos() {
   return 0;
   return 1;
}
```

```
La definición de este método es válida y retorna 0.
```