jLite

Lite version of Java

Language Reference Manual

Sudhakar Perumal
skp2111@columbia.edu

2 Reference Manual

This section of the document describes the subset of java language features supported by jLite programming language. Antlr like syntax is also provided whereever possible.

2.1 Lexical Structure

This section specifies the lexical structure of jLite programming language.

2.1.1 Line Terminators

Lines are terminated by the ASCII characters CR, or LF, or CR LF. The two characters CR immediately followed by LF are counted as one line terminator, not two.

2.1.2 Tokens

Tokens can be found in the form of identifiers, keywords, literals, separators and operators. These tokens will be separated by whitespace and line terminators.

2.1.3 White space

White space is defined as the ASCII space, horizontal tab, and form feed characters, as well as line terminators.

2.1.4 Comments

There are two kinds of comments:

/* text */ A traditional comment: all the text from the ASCII characters /* to the ASCII characters */ is ignored (as in C and C++).

// text A end-of-line comment: all the text from the ASCII characters // to the end of the line is ignored (as in C++).

2.1.5 Identifiers

An identifier is an unlimited-length sequence of letters and digits, the first of which must be a letter. An identifier cannot have the same spelling as a keyword, boolean literal or the null literal.

2.1.6 Keywords

The following character sequences, formed from ASCII letters, are reserved for use as keywords and cannot be used as identifiers

boolean	else	import	super
break	extends	int	this
class	for	new	try
catch	finally	package	
double	if	return	

2.1.7 Literals

A literal is the source code representation of a value of a primitive type, the String type, or the null type. Following types of primitive literals are supported by this language

Integer literals	e.g. 0, 10
Floating Point literals	e.g. 3.14, 1-e9
Boolean literals	e.g. true, false

2.1.8 Separators

The following nine ASCII characters are the separators

```
(   )   {   }   [   ]   ;   ,   .
```

2.1.9 Operators

The following 26 ASCII characters are the operators.

```
=     >     <     !     ?     :
==    <=    >=    !=    &&    ||    ++    --
+     -     *     /     &     |
+=    -=    *=    /=    &=    |=
```

2.2 Types

The jLite programming languages supports two categories of types: primitive types and reference types. Primitive types are Boolean types and numeric types. Reference types are class types, interface types and array types. The values of a reference type are references to objects. String literals are represented by String objects.

2.2.1 Primitive types and values

A primitive type is predefined by the jLite programming language and named by its reserved keyword. The numeric types are the integer types and the floating-point types. Boolean type has exactly two values: true and false. Comparison operators (==, >= and <=) and numerical operations (+, -, *, /) are allowed against integer and floating point types. Boolean types allows relational (== and !=), logical (!, &) and conditional operations (? :).

Examples of primitive types

```
int i = 10;
double d = 10.0;
boolean b = true;
```

2.2.2 Reference types and values

A reference type can be of class types, interface types or array types. jLite supports Objects. An Object is a class instance or an array. The reference values (often just references) are pointers to these objects, and a special null reference, which refers to no object. A new class instance is implicitly created when the string concatenation operator + is used in a non-constant expression, resulting in a new object of type String. A new array object is implicitly created when an array initializer expression is evaluated. New objects of the types Boolean, Integer and Double may be implicitly created by boxing conversion.

Following operations are allowed on references to objects

- Field access
- Method invocation
- Cast operator
- In case of string objects, concatenation

Example of reference type and values

```
String s = “example”;
java.util.HashMap map = new java.util.HashMap();
java.io.StringReader sr = null;
int i[] = new int[10];
Object obj = “example”;
```

Grammar

```
type
        :       Identifier ('.' Identifier)*
        ;
```

2.3 Package

A script file may begin with package declaration. Package declaration takes the same form as java programming language. The package declaration must be the first declaration in the file.

Example

package com.jlite;

Grammar

```
packageDeclaration
        :       'package' qualifiedName ';'
        ;

qualifiedName
        :       Identifier ('.' Identifier)*
        ;
```

2.4 Import Declarations

Import declarations also takes the same form as java programming language. An import declaration makes types available by their simple names. If the referenced package is not available, compilation will result in error.

Example

import java.util.*;

Grammar

```
importDeclaration
        :       'import' Identifier ('.' Identifier)* ('.' '*')? ';'
        ;
```

2.5 Classes

Class declaration defines new reference types and describe how they are implemented. Sub classes and nested classes (declaring many classes in single file) are allowed. However, abstract classes, inner classes and interfaces are not allowed. Access modifiers on the class level are not allowed, in other words, all classes will be treated as public classes.

Example.

class test   is equivalent to   public class test

Grammar

```
classDeclaration
        :       'class' Identifier
     ('extends' type)?
     classBody
        ;

type
        :       Identifier ('.' Identifier)*
        ;
```

2.6 Exceptions

Exception handling is done using try catch block. When an exception is thrown, control is transferred from the code that caused the exception to the nearest dynamically-enclosing catch clause of a try statement that handles the exception. Since jLite converts the source into bytecode, the internal mechanism by which the exceptions are handled is not affected.