SHANGHAI JIAOTONG UNIVERSITY SCHOOL OF SOFTWARE

LITTLEJAVA LANGUAGE REFERENCE MANUAL

Compiler Principle (Fall 2012)

施佳鑫 梁凉 袁理

2012/10/8

```
This manual generally introduces a computer language named LittleJava.
```

Initial Language Summary (Compare and Contrast with Standard Java)

LittleJava is a subset of Java, thus a pure **Object-Oriented Programming Language.**

LittleJava is designed for new-comers in programming.

It **implements basic functions** of Java, but **leaves out** some vague language features.

Supported

1. Self-defined class: **class**
2. Static members of classes: **static**
3. Recursive call of functions
4. Basic Flow Control Statements: **if else(else if), while, for**
5. Comment: style identical with Java
6. Expression: priority same with Java

Unsupported

1. Interface and Inheritance: **interface, extend**
2. Access modifiers: **private, public, protected**
3. **Bitwise** operation
4. **Post-/Pre-**increment/decrement
5. Polymorphism
6. Package: can only import a file within the current location or PATH location.

Remark: This manual defines the minimal ability of LittleJava. Language features of LittleJava will be added as needed. And the content of manual will be enriched with the developing of LittleJava.

Keyword

if、else、while、for、int、string、float、void、return、true、false、new、static、null

Reserved Word

boolean、break、byte、case、catch、char、class、const、continue、default、do、double、else、extends、final、finally、float、for、goto、if、implements、import、instanceof、int、interface、long、native、new、package、private、protected、public、return、short、static、super、 switch、 synchronized、 this、 throw、 throws、transient、 try、 void、 volatile、 while

LEX

```
identifier -> [_a-zA-Z]([_a-zA-Z0-9])*

char -> Character whose ASCII value ranges from 0 to 255

string -> char*

figure -> [0-9]([0-9])*

        -> [0-9]([0-9])*.[0-9]([0-9])*
```

GRAMMAR

```
Program -> ImportSection* ClassDecl*


ImportSection -> import string


ClassDecl -> class identifier { VarDecl* MethodDecl* }

VarDecl -> static ? Type identifier ;

        -> static ? Type identifier = Expr ;

MethodDecl -> static ? Type identifier ( ParameterList ? ){ Statement* }

Type -> int

         -> char

         -> float

         -> bool

         -> void

         -> Type[]

         -> identifier
```

```
Statement -> VarDecl

          -> { Statement* }

          -> if (Expr) Statement

          -> if (Expr) Statement else Statement

          -> while (Expr) Statement

          -> for ( ForInit ? ; Expr? ; Assignment? ) Statement

          -> return Expr ;

          -> break ;

          -> continue ;

          -> Assignment ;


Assignment -> identifier = Expr

           -> identifier[Expr] = Expr


ForInit -> Type identifier

        -> Type identifier = Expr


Expr -> OrExpr

OrExpr -> OrExpr || AndExpr

       -> AndExpr

AndExpr -> AndExpr && EquExpr

        -> EquExpr

EquExpr -> EquExpr == RelExpr

        -> EquExpr != RelExpr

        -> RelExpr
```

```
RelExpr -> RelExpr < AddExpr

        -> RelExpr <= AddExpr

        -> RelExpr > AddExpr

        -> RelExpr >= AddExpr

        -> AddExpr

AddExpr -> AddExpr + MulExpr

        -> AddExpr – MulExpr

        -> MulExpr

MulExpr -> MulExpr ‘*’ UnaryExpr

        -> MulExpr / UnaryExpr

        -> MulExpr % UnaryExpr

        -> UnaryExpr

UnaryExpr -> new Type [ Expr ]

          -> new Type ( ArgumentList? )

          -> ! UnaryExpr

          -> - UnaryExpr

          -> ( Expr )

          -> VarExpr

VarExpr -> VarExpr[ Expr ]

        -> VarExpr.identifer

        -> VarExpr.identifier( ArgumentList ? )

        -> identifier::identifier( ArgumentList ? )

        -> identifier

        -> true

        -> false
```

```
         -> null
         -> this
         -> figure
         -> “string”
         -> ‘character’

ArgumentList -> ArgumentStart* identifier
ArgumentStart -> identifier ,

ParameterList -> ParameterStart* Type identifier
ParameterStart -> Type identifier ,
```

COMMENT

A comment may appear in 2 forms.

One is embedded in a pair of **/*** and ***/**, and cannot be nested.

Sample:

```
/* The quick brown fox jumps over
the lazy dog. */
```

The other is started with **//** and goes to the end of the line.

```
// The quick brown fox jumps over the lazy dog.
```

SAMPLE PROGRAM SNIPPET