

Syntactic Rules

```

class-dcl      = class identifier [ extends identifier ] is body end
body           = { var-dcl ; } constr-dcl { method-dcl }
var-dcl        = type identifier
constr-dcl     = create ( [ param-list ] ) is method-body ;
method-dcl     = [ type ] method identifier ( [ param-list ] ) is method-body ;
method-body    = { var-dcl ; } begin { statement } end
param-list     = var-dcl { , var-dcl }
type           = identifier [ [ ] ]
statement      = method-call-stmt
                assign-stmt
                if-stmt
                while-stmt
                for-stmt
                return-stmt

method-call-stmt = method-call ;
assign-stmt      = variable := expression ;
if-stmt          = if expression then { statement } [ else { statement } ] end ;
while-stmt       = while expression do { statement } end ;
for-stmt         = for identifier := expression to expression do { statement } end ;
return-stmt      = return expression ;
method-call      = name ( [ argument-list ] )
argument-list    = expression { , expression }
expression       = conjunction { | conjunction }
conjunction      = relation { & relation }
relation         = simple-expr [ relational-op simple-expr ]
simple-expr       = term { adding-op term }
term             = factor { multiplying-op factor }
factor           = [ unary-op ] value
value            = constant
                variable
                method-call
                creation
                ( expression )

constant         = int-const
                float-const
                char-const
                boolean-const
                string-const

variable         = name [ [ expression ] ]
creation         = new identifier ( [ argument-list ] )
                new identifier [ expression ]

name             = qualifier [ . identifier ]
qualifier        = super
                identifier

```

relational-op = < | <= | = | ~= | >= | >
adding-op = + | -
multiplying-op = * | /
unary-op = - | ~

Lexical Rules

Comment

a sequence of characters beginning with // and ending at the end of line

Reserved Words

begin	class	create	do	else	end
extends	false	for	if	is	method
new	return	super	then	to	true
while					

Identifiers

identifier sequence of one or more letters or digits beginning with a letter, not being a reserved word

Literals

bool-const **false** | **true**
char-const single graphic character enclosed in single quotes (')
float-const sequence of one or more digits including one decimal point and beginning with a digit
int-const sequence of one or more digits
string-const zero or more graphic characters (other than double quote) enclosed in double quotes (")

Semantic Rules

Value vs Reference Types

Variables of the primitive types (bool, char, float, int) are value variables, storing the primitive value.

Variables of class types (including String, InFile and OutFile) and array variables are reference types, storing a reference to the object or array.

Array types are considered to have one attribute (instance variable):

```
int length;
```

that is the number of elements in the array.

Predefined (Global) Identifiers

bool	boolean type
char	character type
float	float type
int	integer type
String	string class

InFile	input file class (cannot be instantiated) with methods: <pre> bool method readBool () char method readChar () float method readFloat () int method readInt () String method readString () </pre>
OutFile	output file class (cannot be instantiated) with methods: <pre> method writeBool (bool b) method writeChar (char c) method writeFloat (float d) method writeInt (int i) method writeString (String s) method writeEOL () </pre>
stdIn	instance of predefined class InFile - input text file
stdOut	instance of predefined class OutFile - output text file

Scope

Uses “Define before use” rule.

class name	The class name is visible from the end of the class header to the end of the class declaration. Once a class has been compiled, the class name is visible in every class in the same directory.
instance variable	An instance variable of a class is visible from the end of the variable declaration to the end of the class declaration, and in every class where the class name is visible.
method name	A method name of a class is visible from the end of the method header to the end of the class declaration, and in every class where the class name is visible.
formal parameter	A formal parameter is visible from the end of the variable declaration to the end of the method or constructor body.
local variable	A local variable is visible from the end of the variable declaration to the end of the method or constructor body.

Names in a more local scope hide (override) names from a more global scope. Names may not be overloaded within a single scope.

Constructor Chaining

Constructor chaining is not implicit. Within a constructor the superclass constructor can be called via:

```
super (...)
```

System

A system is a collection of classes within a single directory. The execution of a system involves the creation of a single instance of the class `Main` (i.e. the execution of the constructor of the class called `Main`). The constructor of the `Main` class must not take parameters. Each system must have exactly one class called `Main`.