**Language Design Proposal: ODANG**

**Student Names**: Charles Dang, Giovanni Orozco, Marius Larnoey

**Language Name**: ODANG

**Compiler Implementation Language and Reasoning**: Java. We already know how to code in Java.

**Target Language**: Javascript

**Language Description**: Object-oriented programming. We’re going to create a Java-like language that is more optimized and useful.

**Planned Restrictions**: No garbage collection.

**Abstract Syntax**:

var is a variable

i is an integer

str is a string

classname is the name of the class

methodname is the name of the method

type ::= Int | String| Char | Boolean | Void

op ::= \* | / | + | - | = | % | > | < | && | ! | || | ++ | --

expr ::= var | str | i |

this | // reference to current object

println(expr\*) | //prints text on console

expr op expr | //arithmetic operations

expr.methodname(expr\*) | //call a method with any number of expressions as parameters

new classname(expr\*) /\*create a new class with any number of expressions as parameters\*/

(type) expr // cast the expression to a type, runtime error if cast to something it isn’t

vardec ::= type var // declare a variable

booleanstmt ::= var < int || var > int || var <= int || var >= int || var != int || var == int /\* boolean statement that compares values \*/

stmt := vardec = expr; | //simple variable assignment

var = expr; | assign an expression to a variable

while (expr) stmt | // while loop

for (var = expr; booleanstmt; var++ || var--) stmt | // for loop

break; | // break

{ stmt\* } | // block of statements

if (expr) stmt [else stmt] | // if statement

return expr; | // return expression

return; // return Void

methoddef ::= methodname(vardec\*) stmt /\* a method definition that contains any possible number of variable declarations separated by commas as parameters\*/

classdef ::= class classname extends classname //implies subtyping

vardec\*

classname (vardec\*) stmt /\* a constructor that has any number of variables that are separated by commas\*/

[classname operator op (classname &var) stmt] // implement operator overloading

methoddef\*

}

program ::= classdef\* expr

**Computation Abstraction Non-Trivial Feature**: Object-oriented classes

**Non-Trivial Feature #2**: Subtyping

**Non-Trivial Feature #3**: Operator Overloading

**Work Planned for Custom Component**: Subtyping will be left for the end since we need object-oriented classes implemented first