Language Design Proposal

Student Name(s): Miguel Cruz, Jacob Poersch, Nicholas Araklisianos

Language Name: CoopJa

Compiler Implementation Language and Reasoning: Java; Most group members are familiar with the language.

Target Language: C

Language Description: “C’s Cooperative Object Oriented Programming from Java” -- We plan to use Java’s Object Oriented nature and bring this to the C language. With this, we will be including class-based inheritance in our language. We were thinking of a few different target languages, but ultimately settled with C since we felt its differences with Java were significant enough that we could make some meaningful additions.

Planned Restrictions: We will not be featuring any memory deallocation in our language, nor any garbage collection. We also will not be featuring Java’s Generics in our language.

Syntax (Subject to Change):
var is a variable
objectname is the name of a class
methodname is the name of a method
str is a string
i is an integer

```
type ::= int | double | char | boolean | string | auto |   [Built in types of variables]
         objectname                                        [Objects are also types]
op ::=   + | - | * | / |                                   [Arithmetic operations]
         > | < | >= | <= | == | != | ==| |                 [Comparison Operations]
         | | & | ^ | >> | << | ~                           [Bitwise Operators]
         var++; |                                          [Increments a variable]
         var--; |                                          [Decrements a variable]
vardec ::= type var                                        [Variable declarations]

exp ::= var | str | i |                                    [Basic expressions]
        Exp op exp|                                        [Arithmetic expression]
        println(exp)|                                      [prints to the terminal]
        This                                               [Refers to this instance]
        objectname.Method(Var*)                            [Call Method]
        new objectName(exp*)                               [Declare a new instance of an object]
```

```
access ::= Public | Private | Protected              [access type for a method or var]
stmt  ::=       vardec; |                             [Variable Declarations]
                var = exp; |                          [assignment to variable]
                If (exp) stmt else stmt |             [standard if/else statement]
                while (exp) stmt |                    [loop statement with restriction]
                for (vardec; exp; exp) stmt |         [for loop statement]
                break; |                              [escape loop statement]
                {stmt*}                               [block]
                return exp|                           [return an expression]
                return; |                             [Empty return]

instancedec ::= access vardec;
result_type ::= type | void                           [Return types]
methodef::=     Access result_type methodname (vardec) stmt     [Method declarations]

objectdefheader ::= access class objectname | access class objectname extends objectname
objectdef::=    objectdefheader {
                        vardec*                       [Variable declarations]
                        public objectname {smt*}      [Constructor]
                        methodef*
                }

program ::= objectdef* exp                            [exp is an entry point]
```

Computation Abstraction Non-Trivial Feature: Objects and methods with class based inheritance.

Non-Trivial Feature #2: Access Modifiers (such as public, private, or protected). Refers to both variables and methods.

Non-Trivial Feature #3: Type Inference, allowing for an “auto” type. The compiler will determine what the “auto” type actually is.

Work Planned for Custom Milestone: Access Modifiers. Until it is implemented, everything is treated as public.