**Language Design Proposal: pOOP Base**

**Student Name(s):** Kyle Dewey

**Language Name: pOOP Base**

**Target Language:** C

**Language Description:** (Pathetic) object-oriented programming. The goal is for me to better understand how object-oriented programming languages work. I want to implement a Java-like language with classes and subclasses. I'm intentionally picking C because it is pretty low-level, but it's not so low-level that it will require me to spend a lot of time understanding the target language.

**Key Features**: Objects + methods with class-based inheritance, subtyping, checking if a variable is initialized before use, checking that a function returning non-void always returns.

**Planned Restrictions:** there is no way to reclaim allocated memory (either automatically or manually), and no optimizations.

**Suggested Scoring and Justification:**

* **Lexer**: 10%. Only support for reserved words, identifiers, and integers. No comments.
* **Parser**: 10%. Uses S-expressions.
* **Typechecker:** 40%. Handles subtyping and method overloading, checking if a variable is initialized before use, checking that a function returning non-void always returns.
* **Code Generator:** 40%. Has to handle inheritance and virtual tables (for method calls).

**Syntax:**

var is a variable

classname is the name of a class

methodname is the name of a method

str is a string

i is an integer

type ::= `Int` | `Boolean` | `Void`| **Built-in types**

classname **class type; includes Object and String**

**Arithmetic and relational operators**

op ::= `+` | `-` | `\*` | `/` | `<` | `==`

exp ::= var | str | i | **Variables, strings, and integers are**

 **expressions**

 **`**this` | **Refers to my instance**

`true` | `false` | **Booleans**

`(` `println` exp `)` | **Prints something to the terminal**

`(` op exp exp `)`| **Arithmetic operations**

`(` `call` exp methodname exp\* `)`| **Calls a method**

`(` `new` classname exp\* `)`  **Creates a new object**

vardec ::= `(` `vardec` type var `)` **Variable declaration**

stmt ::= vardec | **Variable declaration**

`(` `=`var exp `)` | **Assignment**

`(` **`**while` exp stmt\* `)` | **while loops**

 **`**break` | **break**

`(` `if` exp stmt [stmt] `)` | **if with optional else**

 `(`return [exp] `)` | **return, possibly void**

methoddef ::= `(` `method` methodname

 `(` vardec\* `)` type stmt\* `)`

constructor ::= `(` `init` `(` vardec\* `)`

 [`(` `super` exp\* `)`]

 stmt\* `)`

classdef ::= `(` `class` classname [classname]

`(`vardec\* `)`

 constructor

 methoddef\* `)`

program ::= classdef\* stmt+ **stmt+ is the entry point**

**Example (animals with a speak method):**

(class Animal

 ()

 (init ())

 (method speak () Void

 (return (println 0))))

(class Cat Animal

 (init ()

 (super))

 (method speak () Void

 (return (println 1))))

(class Dog Animal

 (init ()

 (super))

 (method speak () Void

 (return (println 2))))

(vardec Animal cat)

(vardec Animal dog)

(= cat (new Cat))

(= dog (new Dog))

(call cat speak)

(call dog speak)