Language Design Proposal: pOOP

Student Name(s): Kyle Dewey
Language Name: pOOP

Compiler Implementation Language and Reasoning: Java. I'm already familiar with it, and I'm not planning to get into optimizations. Learning a new language is an unnecessary risk.

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, access modifier checking, runtime checking for casts, checking if a variable is initialized before use, checking if void is used as a value, 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**: 2%. Only support for reserved words, identifiers, and integers. No comments.
- **Parser**: 5%. Uses S-expressions.
- **Typechecker:** 33%. Handles subtyping, access modifiers, and method overloading, checking if a variable is initialized before use, checking if void is used as a value, checking that a function returning non-void always returns.
- **Code Generator:** 40%. Has to handle inheritance, runtime casts, 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
```

```
op ::= `+` | `-` | `*` | `/` Arithmetic operations
exp ::= var | str | i | Variables, strings, and integers are
                        expressions
        `this` | Refers to my instance
        `(` `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
        `(` `cast` type exp `)` Casts an expression as a type
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
access ::= public | private | protected
methoddef ::= `(` `method` access type methodname
                  `(` vardec* `)` stmt* `)`
instancedec ::= `(` vardec access type var `)`
constructor ::= `(` `init` `(` vardec* `)`
                    [`(` `super` exp* `)`]
                    stmt* `)`
classdef ::= `(` `class` classname [classname]
               `(` instancedec* `)`
              constructor
              methoddef* `)`
program ::= classdef* stmt+  stmt+ is the entry point
```