# JavaCoop Documentation

By Terrance Kuo and Rudy Robles

## Target Language

We decided to use C Language as the language to make the compiler for. We are picking C because it is a low level language. Also With C we have the room to add object-oriented features found in higher level languages. The implementation language is in Java since both of us know Java and optimized performance won't be important here. Many aspects of C and Java also overlap, which will be able to save us much time to get things done.

## Features

The 1st feature we are having for our project is that objects and methods will have class-based inheritance.

An example of class based inheritance would be

```
class Bicycle
{
    // the Bicycle class has two fields
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }
    // the Bicycle class has three methods
```

```
    public void applyBrake(int decrement)
    {
        speed -= decrement;
    }
    public void speedUp(int increment)
    {
        speed += increment;
    }
}
// derived class
class MountainBike extends Bicycle
{
    // the MountainBike subclass adds one more field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear,int speed,
                        int startHeight)
    {
        // invoking base-class(Bicycle) constructor
        super(gear, speed);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one more method
    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }
```

```
}
```

Here we can see that the class MountainBike is able to take all of the methods from Bicycle since it is able to extend from Bicycle. The classes that inherit methods from the extended classes are also able to add additional methods.

Our 2nd feature we are adding is Subtyping. Subtyping is a key feature of object-oriented programming languages such as Java. In Java, Sis a subtype of T if S extends or implements T. Some examples of subtyping are:

- Integer is a subtype of Number
- ArrayList<E> is a subtype of Collection<E>
- String is a subtype of Object

Subtyping as a declaration would be

```
Number num = new Integer(2000);

Object obj = new String(“Hello World”);
```

And doing it as method invocations would be

```
JPanel panel = new JPanel();
panel.add(new JTextField(20));
panel.add(new JButton(“OK”));
```

Our custom component for this project is access modifier checking. Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members. Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components. For our program, until specified, everything is implicitly considered public.

# Planned Restrictions

We decided to not do any sort of code optimization. Code optimization is any method of code modification to improve code quality and efficiency. A program may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or

performs fewer input/output operations. However we decided to do this since it resembles the normal C compiler more as is. Having no optimization will mean that some errors may be ignored in return for a better speed execution in general like what c compiler does.
Another restriction that we decided to add on later on would be having the methods and classes only taking only 1 statement at a time. There are only 1 type of number types currently in our code. These limitations exist currently as a practical choice rather than an intentional one. To put it simply, we don't have the time or manpower to go more in depth as of now. These aspects of our code can be added in later if we were to put more work into this project in the future.

## Struggles during development

Throughout working on this project there were many different challenges that we had to work though. The most prevalent factor being the new work conditions because of the virus. The appearance of the virus and the subsequent quarantine have made communications more difficult between the two of us since face to face meeting is no longer possible and messaging between the two of us is time consuming as it takes time to read and send messages.

The fact that there were only two of us was a growing problem, particularly during the second half of the semester. This is due to the project getting more and more complex and difficult to implement starting when type checker came around. With only two of us, this meant that both of us had to do more per person compared to many other groups. Rudy also had troubles getting all the segments to sync up as one program.

## What we would do differently

Over the course of this class there are certain decisions that we should have done

differently that would have made our struggles with this project much more easier for the both of us.

The first of these decisions is asking the professor for more help than we did. Towards the end, we did ask for help and found that the feedback was very good and encouraging.

The second of our decisions was our attitude towards this work. Both of us had taken our time towards this project and not had done things as quickly as it could have been. This ties in with a similar point in that it was also not our biggest priority during the semester. Both of us were also graduating seniors, meaning we had also taken many classes this semester that also called for a great amount of attention, particularly our senior project. Still this doesn't excuse the fact that we had a tendency to procrastinate on this work.

Our last and biggest hitting decision was our team size. In the beginning of the semester we had a hard time finding anyone else to work with us and decided to try doing it with the two of us. However after doing typechecker of the project it was notable how hard it was to do this project with just the two of us. The addition of one person to our group would have greatly lightened the load between the two of us in addition to adding more brain power to make our program better.

# Abstract Syntax

var is a variable
class is the name of a class
function is the name of a method
str is a String
int is an integer
type ::= Int | Boolean | Void | Built-in types
    class class type; includes Object and String
    op ::= + | - | * | / Arithmetic operations
    exp ::= var | str | int | Variables, strings, and integers are expressions

this | Refers to my instance
printf(exp) | Prints something to the terminal
exp op exp | Arithmetic operations
exp.function(exp*) | Calls a method
new class(exp*) | Creates a new instance of a class
(type)exp Casts an expression as a type

vardec ::= type var Variable declaration
stmt ::= vardec; | Variable declaration
var = exp; | Assignment
while (exp) stmt | while loops
break; | break
{ stmt* } | block
if (exp) stmt else stmt | if/else
return exp; | return an expression
return; return Void
access ::= public | private | protected
methoddef ::= access type methodname(vardec*) stmt
vardecs are comma-separated
instancedec ::= access vardec; instance variable declaration
classdef ::= class classname extends classname {
instancedec* constructor(vardec*) stmt vardecs are comma-seperated
methoddef*
}
program ::= classdef* exp exp is entry point

void does not represent a sort of value and is only used for methods
Uninitialized variables will be counted as errors and must have a var before them that is not void