Java^{2}Script

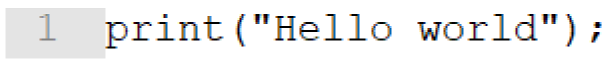
By Jesus Banuelos, Jim Inong, and Justin Kingston

Why this language, and why this language design?

- Our group chose Java as the compiler language as it was the language all team members were most familiar with. Javascript was chosen as our target language due to it being a high-level language, making the project not impossible considering our group's familiarity with both languages, while still ensuring the project would be non-trivial due to the differences between Object-oriented and scripting languages.
- Our language design was chosen bearing in mind the core mechanics of Java. Although we removed some features Java has, we focused on implementing features such as Access Modifiers, Subtyping, and Class-Based Inheritance (while ignoring Javascript's built-in classes, since using Javascript's classes would make this feature trivial).

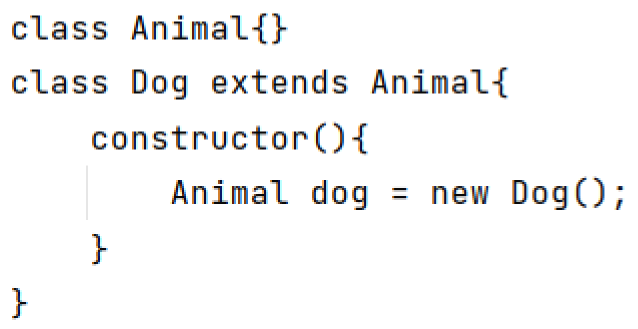
Code snippets

- Hello World in Java^{2}Script



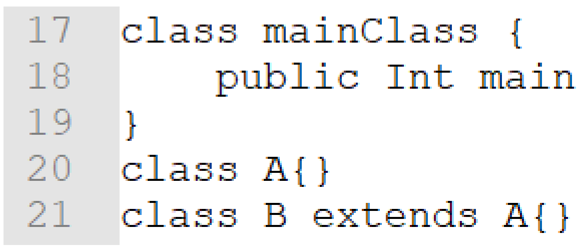
```
print("Hello world");
```

- Example class definition in Java^{2}Script. Notice how constructors are created with the keyword 'constructor'. We also allow for inheritance via the 'extends' keyword.



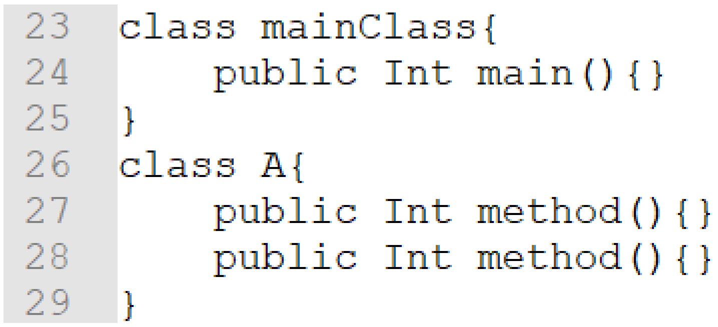
```
class Animal{}
class Dog extends Animal{
    constructor(){
        Animal dog = new Dog();
    }
}
```

- Java^{2}Script also features class-based inheritance.



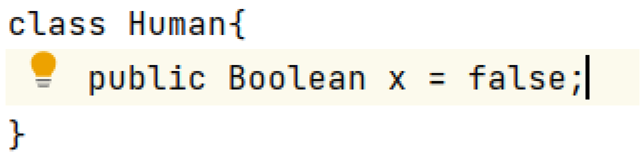
```
class mainClass {
    public Int main(){}
}
class A{}
class B extends A{}
```

- Different methods in Java[2]Script must have different names. The following code snippet is invalid and will result in an error due to the method() method being initialized twice.



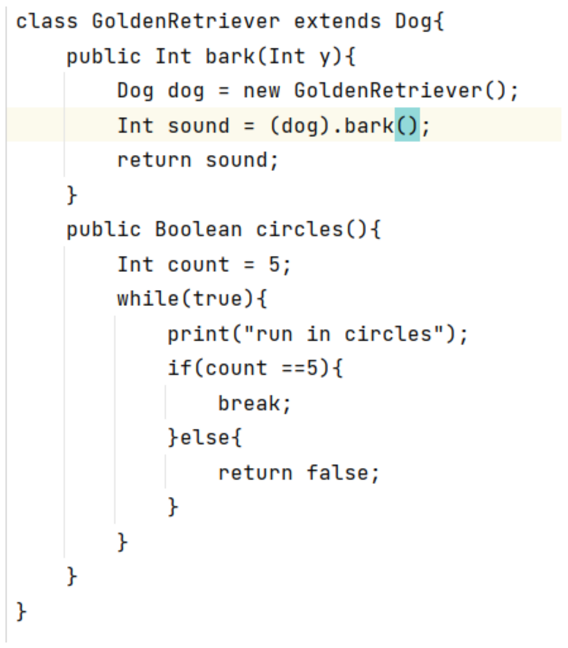
```
class mainClass{
    public Int main(){}
}
class A{
    public Int method(){}
    public Int method(){}
}
```

- Cannot initialize outside of methods.



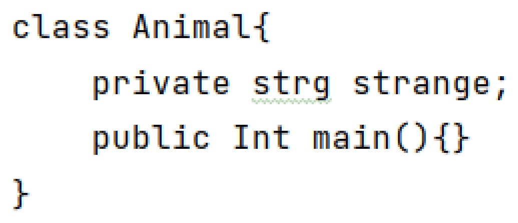
```
class Human{
    public Boolean x = false;
}
```

- When calling a method make sure to surround the variable you are using to call the method with parenthesis. Also methods cannot be called without them being assigned to a variable since every method has to return something.



```
class GoldenRetriever extends Dog{
    public Int bark(Int y){
        Dog dog = new GoldenRetriever();
        Int sound = (dog).bark();
        return sound;
    }
    public Boolean circles(){
        Int count = 5;
        while(true){
            print("run in circles");
            if(count ==5){
                break;
            }else{
                return false;
            }
        }
    }
}
```

- When you declare instance variables they cannot be changed later on. It is a strange behavior and useless but you can still declare them.



```
class Animal{
    private strg strange;
    public Int main(){}
}
```

Known limitations

- Java^{2}Script currently has a barebones code generator.
- Although Java^{2}Script features methods, methods may not have a return type of “void.”
- Java^{2}Script features extending classes to create subclasses, but does not feature abstract classes, which would normally allow the implementation of methods in subclasses. As such, no “implements” keyword exists either.
- Variables cannot have their initial values reassigned.

- Instance variables are not functional (as they can't be reassigned or initialized) but they will compile properly.

What would you do differently?

- If we could do it all again, we would likely try to write tests before writing our code. This would allow us to better pinpoint the kind of code we should be writing for the compiler.
- (Only for Jesus) Started off in Visual Studio Code and was having trouble with the version control and towards the end of the semester switched over to IntelliJ and that made things a lot easier. Thus, if I were to do this project again I would begin on IntelliJ rather than on VSCode.

How do I compile your compiler?

- Compile in Intellij with Java JDK 8 using Maven with the following commands:
  - mvn compile

How do I run your compiler?

- To compile example code from .j^2s to .js:
  - mvn exec:java -Dexec.args="tests/inheritance.j^2s output_files/inheritance_output.js"
  - mvn exec:java -Dexec.args="tests/recursion.j^2s output_files/recursion_output.js"

Abstract Syntax:

var is a variable

classname is the name of a class

methodname is the name of a method

object is a var, classname, String, this

strg is a string

```
intg is an integer

bool is a boolean

type ::= Int | Boolean | classname | String

primary_exp ::= var | strg | intg | `(` exp `)` | bool | exp`.`methodname(exp*)  |

new classname(exp*) | this

multiplicative_op ::= `*` | `/`

multiplicative_exp ::= period_exp( multiplicative_op period_exp)*

additive_op ::=  `+` | `-`

additive_exp ::= multiplicative_exp( additive_op multiplicative_exp)*

rel_op ::= `<` | `>`

rel_exp ::= additive_exp (rel_op additive_exp) //has to be 0 or 1 call

boolean_op ::=  `==` | `!=`

boolean_exp ::= rel_exp (boolean_op rel_exp) // has to be 0 or 1 call

exp ::= boolean_exp

varDec ::= type var

stmnt ::= varDec `;` |

        while  (exp) stmnt |

        break`;` |

        { stmnt* } |

        if (exp) stmnt else stmnt |

        return exp `;` |

        print(exp) `;` | varDec `=`exp`;`
```

```
accessMod :: = public | private | protected

methodDef ::= accessMod type methodname(varDec*) stmnt

instanceDec ::= accessMod varDec`;`

mainMethod ::= public Int main() stmt

classDef ::= class classname extends classname {

        mainMethod

        instanceDec*

        constructor(varDec*) stmnt

        methodDef*

        }

programName ::= classDef*
```

A method named main is the entry point