Mozilla Rhino

General

- JavaScript interpreter
- Implemented fully in Java
- ECMA specification works as requirements

General structure

1. Token Stream
    - Turns text into “tokens”
2. Parser
    - Turns tokens into statements
3. Compiler
4. Interpreter
    - Executes statements

Token Stream

- Turns text into “tokens”
  - A token is a set of characters that are used as a single unit, e.g. “if”, “while”, “{“, “myvar”
  - Each token is identified by a type, e.g. “if” is IF, “{“ is LC (for left curly) and “myVar” is NAME (an identifier, such as variable or function name)
- Goes through input one character at a time and returns a list of tokens
- Basically groups character into meaningful units

Token Stream implementation

- Implemented in class org.mozilla.javascript.TokenStream
- Class org.mozilla.javascript.Token contains the token definitions
- Method org.mozilla.javascript.TokenStream#getToken is called every time the next token is requested by the parser. Contains big switch statement that determines which kind of token is returned.
    - This is a good place to set a breakpoint if you are looking for when exactly a certain token is produced
- Fields org.mozilla.javascript.TokenStream#string and org.mozilla.javascript.TokenStream#number hold the actual values parsed, e.g. for “2”, getToken will return NUMBER, but the value 2 will be stored in number field for it to be retrieved later

Parser

- Turns tokens into statements. This is a recursive descent parser
- Consumes tokens from token stream and builds statements according to the JavaScript grammar
  - For example, if first token is IF, the parser will expect to see a LP (left parenthesis) then an expression, then a RP, then a block, etc.
  - If the token found does not fit the grammar, an error is produced, e.g. “if 1” would produce error because parser was expecting to see a ‘(‘ and not a number

Parser implementation

- In class org.mozilla.javascript.Parser
- Method org.mozilla.javascript.Parser#parse() begins the parsing
- There is a method for each kind of expression defined in the grammar, e.g. statement, block, condition, eqExpression…
- Method org.mozilla.javascript.Parser#statementHelper is a good place to set a breakpoint, since most types of statements will be parsed here (in the big switch statement)

Compiler

- Turns parsed statements into an intermediate representation (IR)
- IR is a stack based language
- IR is a set of instructions, like push to stack, pop from stack, add numbers, call function...

Interpreter

- Executes the IR instructions from the compiler
- Creates objects, performs operations, etc.
- Instructions are not identical to parsed code, e.g. 2 + 2 gets turned into:
  - Push 2 to stack
  - Push 2 to stack
  - Result = Add top 2 numbers of stack and remove them
  - Push result to stack
  - Return top of stack

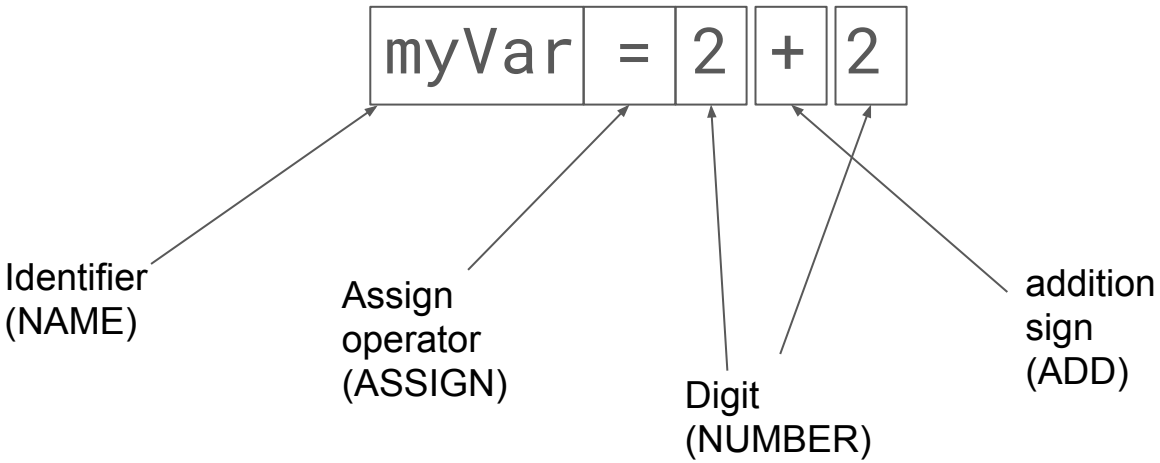
Interpreter implementation

- org.mozilla.javascript.Interpreter
- org.mozilla.javascript.Interpreter#interpretLoop executes the instructions. Contains big switch statement with a case for each type of instruction

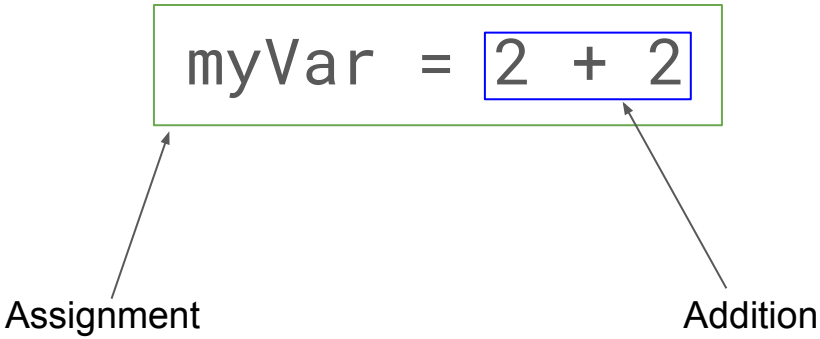
Example: code

```
myVar = 2 + 2
```

Example: lexer



Example: parser



Example: interpreter

```
myVar = 2 + 2
```

```
result = add(2, 2)
myVar = result
```

Set up and run Rhino

- Extract rhino1_6R5.zip
- In IntelliJ IDEA, New > Project from existing sources, select extracted folder
- In dialog, uncheck all source folders except for src and toolsrc
  - If you got no option in the dialog, right click src on left tree and select “Mark Directory as” > Sources root for src and toolsrc
- Right click toolsrc/org/mozilla/javascript/tools/debugger, Mark directory as > excluded
- Run main on org.mozilla.javascript.tools.shell.Main
- You should get an interpreter with the prompt “js>”
- You can debug this main method and enter statements in the interpreter to see how they are processed