

Section 1: Program

The zip file is included in this repository.

Section 2: Teamwork

I was the Primary Engineer, and my partner was the Documentation and testing Engineer. As the Primary Engineer did the primary work on the parser and functionality of Catscript and my partner wrote three tests in my CatscriptPartnerTests.java file, as well as my technical writing.

My partner gave me three tests in the same format as the TDD tests as stated below. My partner's tests were very helpful as he had found several holes in my code that I had to make some changes to the execution of my Catscript code in order to satisfy the functionality of Catscript's grammar.

My partner also wrote my technical guide on Catscript. I believe it to explain the grammar of Catscript very thoroughly.

Section 3: Design pattern

Memoization was used from line 37 – 45 in CatscriptType.java. I used the pattern to cache the list type and save a little on computation.

```
private static final Map<CatscriptType, CatscriptType> LIST_TYPES = new
HashMap<>();
public static CatscriptType getListType(CatscriptType type) {
    CatscriptType listType = LIST_TYPES.get(type);
    if(listType == null) {
        listType = new ListType(type);
        LIST_TYPES.put(type, listType);
    }
    return listType;
}
```

Section 4: Technical writing.

Catscript Guide

Introduction

This guide provides an overview of the Catscript language, written by Cole Reimer for Gage Hilyard. We have completed the code for writing a compiler for the programming language “CatScript”. Catscript is a simple, statically typed programming language used to demonstrate the basics.

The Five Stages of Catscript are:

- *Tokenizing*
- *Parsing*
- *Evaluating*
- *Transpiling to Javascript*
- *Compiling to JVM Bytecode*

Program Structure:

A CatScript program is made up of a set of statements. Statements can be either program statements or function declarations. A program statement can be one of the following:

- *statement*
- *function_declaration*

A function declaration looks like this:

```
function function_name(parameter_list)[: return_type]{  
    function_body_statement_1  
}
```

Function parameters can be of a specific type, specified in the parameter list, and the function can return a specific type, specified in the return type.

Statements:

A statement is a command that performs an action. The different types of statements in CatScript include:

- *for_statement*
- *if_statement*
- *print_statement*
- *variable_statement*
- *assignment_statement*
- *function_call_statement*
- *return_statement*

For Statement:

A for statement is used to loop through a collection of items. The for statement takes an identifier that represents the current item in the collection, an expression that represents the collection itself, and a set of statements to execute for each item in the collection. The syntax for the for statement is:

```
for (identifier in expression) {  
    statement_1  
    statement_2  
}
```

If Statement:

An if statement is used to execute a set of statements if a condition is true. The if statement takes an expression that represents the condition and a set of statements to execute if the condition is true. The syntax for the if statement is:

```
if (expression) {  
    statement_1  
    statement_2  
}  
[ else if (expression_2) {  
    statement_3  
    statement_4  
}]  
[else {  
    statement_5  
    statement_6  
}]
```

Print Statement:

A print statement is used to output a value to the console. The print statement takes an expression that represents the value to output. The syntax for the print statement is:

```
print(expression)
```

Variable Statement:

A variable statement is used to declare a variable and optionally initialize it with a value. The variable statement takes an identifier that represents the variable name, an optional type expression that specifies the variable type, and an optional expression that initializes the variable with a value. The syntax for the variable statement is:

```
var identifier[: type_expression] = expression;
```

Assignment Statement:

An assignment statement is used to assign a value to a variable. The assignment statement takes an identifier that represents the variable name and an expression that represents the value to assign. The syntax for the assignment statement is:

```
identifier = expression;
```

Function Call Statement:

A function call statement is used to call a function. The function call statement takes an identifier that represents the function name and an argument list that contains the arguments to pass to the function. The syntax for the function call statement is:

```
function_name(argument_list);
```

Return Statement:

A return statement is used to end the execution of a function. The return statement returns control and the value to the calling function. The syntax for the return statement is:

```
return [expression]
```

Expressions:

An expression is a combination of values, variables, and operators that can be evaluated to a single value. There are different types of expressions in CatScript, including:

- *equality_expression*
- *comparison_expression*
- *additive_expression*
- *factor_expression*
- *unary_expression*
- *primary_expression*

Equality Expression:

An equality expression is used to compare two values for equality. The equality expression takes two comparison expressions and an operator that represents the equality operation. The syntax for the equality expression is:

```
comparison_expression_1 != comparison_expression_2  
comparison_expression_1 == comparison_expression_2
```

Comparison Expression:

This expression allows us to compare two expressions using comparison operators such as >, <, >=, <=. It consists of additive_expression optionally followed by one or more comparison operators followed by another additive_expression. The syntax for the comparison expression is:

```
additive_expression_1 >= additive_expression_2  
additive_expression_1 <= additive_expression_2  
additive_expression_1 > additive_expression_2  
additive_expression_1 < additive_expression_2
```

Additive Expression:

This expression allows us to add or subtract two expressions. It consists of factor_expression optionally followed by one or more addition or subtraction operators followed by another factor_expression. The syntax for the additive expression is:

```
factor_expression_1 + factor_expression_2  
factor_expression_1 - factor_expression_2
```

Factor Expression:

This expression allows us to multiply or divide two expressions. It consists of unary_expression optionally followed by one or more multiplication or division operators followed by another unary_expression. The syntax for the factor expression is:

```
unary_expression_1 / unary_expression_2  
unary_expression_1 * unary_expression_2
```

Unary Expression:

This expression allows us to apply unary operators such as negation or logical negation to an expression. It consists of a unary operator followed by another unary_expression, or a primary_expression. The syntax for the unary expression is:

```
not unary_expression_1  
- primary_expression
```

Primary Expression:

This expression represents the smallest unit of an expression. It can be a variable identifier, a literal value such as a string, integer, boolean or null, a list literal, a function call, or an expression wrapped in parentheses. The syntax for the primary expression is:

```
primary_expression = IDENTIFIER | STRING | INTEGER | "true" | "false" | "null" |  
                    list_literal | function_call | "(", expression, ")"
```

Catscript Typesystem

The types of all variables and functions/parameters are known at compile time. Catscript can make certain guarantees about the runtime types.

CatScript is statically typed, with a small type system as follows

CatScript Types:

- *int* - a 32 bit integer
- *string* - a java-style string
- *bool* - a boolean value
- *list* - a list of value with the type 'x'
- *null* - the null type
- *object* - any type of value

Catscript does have one complex type: list

You can declare a list with "list<T>"

Examples:

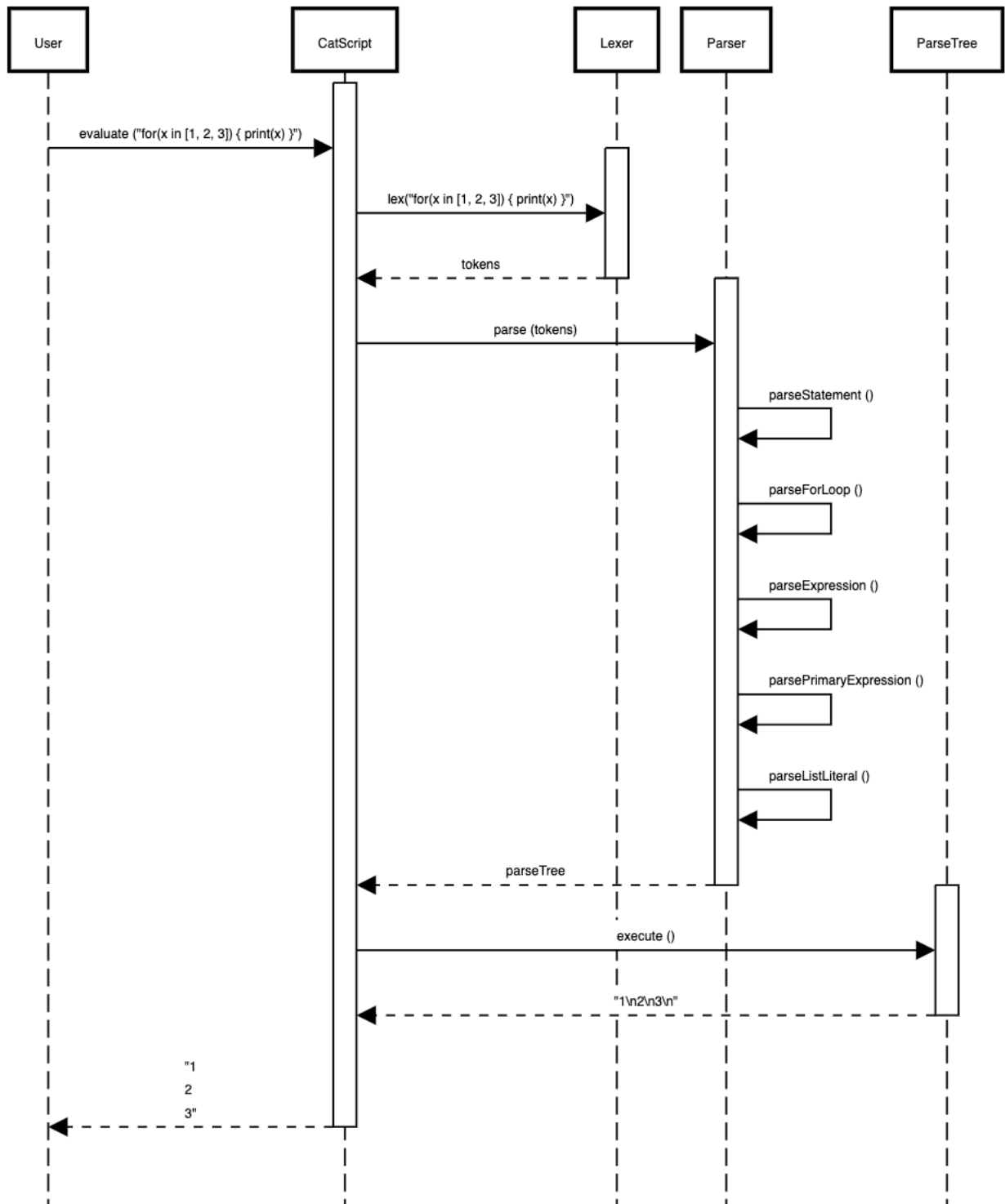
```
list<int> - list of integers  
list<object> - list of objects
```

What does assignability look like in Catscript?

- *Nothing is assignable from void*
- *Everything is assignable from null*
- *Otherwise we check the assignability of the backing java classes*

Section 5: UML.

Catscript For Loop Statement Diagram



Section 6: Design trade-offs

We decided to create our own Recursive-Descent Parser instead of using a parser generator like ANTLR.

The pros of creating our own parser were that we could easily debug the parser and add or remove what we wanted from it. A con for creating our own parser was that time would be spent creating the parser.

It was clear that the pros outweighed the cons in this scenario as the parser was easy to make and understand. The final code is very easy to read, can be modified easily to add any new keywords and operations. Which compared to the parser generators which are nearly impossible to read and understand at a glance.

Section 7: Software development life cycle model

We used Test Driven Development (TDD) for this project. TDD was very helpful as it made it quite clear what to implement next and if any changes we made had fundamentally broke any functionality of our code since we could easily run the tests again.

I do not think it hindered the development of Catscript in any sort of way since the tests were nontrivial and tested the code in ways that it would realistically be used in.

I personally find TDD to be very helpful because of how reassuring it is just to run the tests after any change to confirm whether any large modifications to the code changed its functionality.