Jzero

a programming language

CSE 423:Compiler Writing

Marisa Loraas

January 25, 2022

Contents

1 Introduction

Jzero is a subset of Java. It is intended to do simple and somewhat trivial computations using Java syntax. Sadly it will not include any object oriented capabilities, but will be able to mirror computations from NMT's CSE 113 course assignments in Java syntax.

Jzero will be legal .java files, and be consistent with the textbook "Build Your Own Programming Language" by Dr. Clinton Jeffery.
A Hello world file named "HelloWorld.java" in Jzero would look something like this:

```
public class HelloWorld {
    public static void main (String args[]){
        System.out.println("Hello World");
    }
 }
```

Java supports many different types, Jzero will include:

- int
- double
- char
- String
- boolean
- arrays (of all types)

In Jzero, int and char both refer to 64 bit values
Arrays can be of any type but can only be 1 dimensional. They also require the "new" tag to initialize.
Jzero has while and for loops as control structures for the language. Both will require curly braces. for loops cannot have initialization within the statement, all 3 expressions must not be empty, and multiple variables cannot be updated at the same time. Jzero will also include if statements and switch statements structured like so:

```
if (statement){
    ...
}

if (statement){
    ...
} else {
    ....
}

if (statement){
    ...
} else if (statement){
    ...
} else {
    ...
}
switch (statement){
    case identifier:
        ...
        break;
    ...
    default:
        ...
        break;
}
```

When in doubt, Jzero will include all of the specifications from the textbook mentioned at the beginning of this introduction. This language is meant to help compiler students in understanding compilers and building programming languages.

2 Lexical Rules

The Lexical rules for Jzero begin with the lexical rules required for Java; however, Jzero may reduce and/or simplify these rules.

2.1 Whitespace and Comments

Jzero will include the whitespace characters tab(\t), space, carriage return(\r), formfeed(\f), and newline(\n) from Java.

Jzero will support both types of comment styles that are supported in Java. Comment text can either be placed between the /∗ and ∗\delimeters for multi-line use or any text followed by // for single line use.
Here is an example:

```
//This is a single line comment
variable = 35;
/*This is
a multi-line
comment */
```

2.2 Reserved words

Java Contains about 50 reserved words for use as keywords, meaning they cannot be used as identifiers. Below is a table of these reserved words, those in bold will be used as reserved words in Jzero, and those not should result in a fatal error.

abstract	continue	**for**	**new**	**switch**
assert	**default**	**if**	package	synchronized
bool	do	goto	private	this
break	**double**	implements	protected	throw
byte	**else**	import	**public**	throws
case	enum	instanceof	**return**	**true**
catch	extends	**int**	short	try
char	final	interface	**static**	**void**
class	**false**	long	**string**	volatile
const	float	**null**	super	**while**

2.3 Operators

Java contains about 50 different operator character tokens. Jzero will reduce this to support about 18 tokens, shown below.

=	assignment
+ - * /	binary arithmetic, int and double
%	binary arithmetic, int
+= -=	unary increment and decrement, int only
== != <= >= <>	binary comparison
&& || !	logical AND, OR, and NOT
: ?	conditional

An error is reported for:

- bitwise or shift operators
- augmented assignment operators
- ternary ?: operator
- Any kind of object oriented operation

In Jzero, a comma is valid punctuation but not an operator.

2.4 Literals

1. Integers
   - Only decimal integer literals are allowed in Jzero
   - Hexadecimal, octal, and binary are recognized and reported as lexical errors
   - Underscores with integers will also throw lexical errors
2. Reals
   - Simple real numbers where one or more integers on the left and/or right side of the decimal place will be allowed in Jzero

- Jzero will report scientific/exponent as a lexical error
- Hexideciaml, binary, and signed real numbers will also be reported as lexical errors

3. Characters
    - Only simple single characters literals consisting of a single character surrounded by and apostrophe will be allowed in Jzero. This can be an escape character or a printable keyboard character.
    - Jzero will report error with character literals consisting of more than one character
4. Strings
    - Jzero will have simple literal strings with 0 or more characters enclosed in quotation marks.
    - Characters in strings can either be an escape character or a printable keyboard character.
5. Booleans
    - The boolean type has two values, represented by the boolean literals ”true” and ”false”
6. Null Literal
    - The null type has one value, the null reference, represented by the null datatype literal ”null”
7. Escape Sequences (Character and String Literal)
   This table was adapted from this Tutorials Point page:

   `https://www.tutorialspoint.com/escape-sequences-in-java`

Escape Sequence	Hex value in ASCII	Character Represented
\t	09	Horizontal Tab
\n	0A	Newline
\r	0D	Carriage Return
\f	0C	Form Feed
\b	08	Backspace
\’	27	Apostrophe or single quotation mark
\”	22	Double quotation mark
\\	5C	Backslash

2.5 Punctuation

Punctuation characters are lexemes that are supported in Jzero that are not part of operators, identifiers, or literals.
Punctuation character in Jzero:
( ) , ; { } [ ] .
All other punctuation letters should be reported as lexical errors.

2.6 Identifiers

Jzero will follow the same rules of identifiers that Java has: a letter followed by 0 or more letters and/or digits and identifiers are case sensitive.

3 Syntax

3.1 Function Syntax

Functions in Jzero for simplicity sake will always have to be static, any other function types will throw an error. Multiple parameters of different data types will also be permitted. All functions must be within a class, which should be the same as the file name.
The function syntax for Jzero will look like the following:

```
public class identifier{
    public static return_type identifier(parameter_list){
        function body
    }
}
```

3.2 Control Structures

```
if (statement) {body}
if (statement) {body} else {body}
if (statement) {body} else if (statement) {body} else {body}
while (statement) {body}
for (init; condition; increment) {body}
switch (statement){
    case identifier:
        body
        break;
    ...
    default:
        body
        break;
}
```

3.3 Declaration Syntax

Jzero will only allow simple initializers like int, double, array, char, and String.
Global variables outside of functions but within a class will have a declaration syntax like this:

```
public datatype variableName;
public datatype variableName[] = new datatype[];
public datatype variableName = literal;
```

Within functions, declarations will look like this:

```
    datatype variableName;
    datatype variableName[] = new datatype[];
    datatype variableName = literal;
```

A slightly simpler syntax is allowed for parameter lists, which do not allow initializers.

4 Data Types and Semantics

4.1 Numbers

Jzero will only allow two different number data types for simplicity sake: integers (int) and doubles (double). Type casting will not be allowed, unlike in Java.

4.2 Booleans

Like in Java, booleans will have their own data type labeled ”boolean” in which there are two values it can have: true or false.

4.3 Characters

Characters will be there own separate data type from strings, where a single legal character is included within two apostrophe's/single quote marks. They will be marked by the ”char” tag.

4.4 Strings

Like in Java, strings will be there own data type, where a string can either be null (its own type) or 1 or more characters surrounded by quotation marks. They will be marked by the ”String” tag (case sensitive).

4.5 Arrays

Arrays will be initialized in the same way they are in Java, where the new tag is used. Arrays will also be allowed to be any of the data types listed previously.

5 Summary

Jzero is intended to be a very small subset of the actual Java coding language, designed so that compiler students can learn the necessary steps needed in creating a programming language and compiler. Even with this, the notation of the language should be simple and effective enough to perform simple programming tasks.

6 References

The structure for this paper and some if its contents was referenced from Dr.Jeffery's website ***linked here***. As well as the material in his book (our class textbook) ”Build Your Own Programming

Language"
The references to material about the java language came mostly from oracles website ***linked here***.