High-Level Language

Friday, March 25, 2022 9:58 PM

"High thoughts need a high language."
- Aristophanes (427-386 B.C.)

9.1 Examples

Examples will go through
- *Array types*
- *Object Types*
- *A linked list type*

Example 1: Hello World:

```
/** Printers "Hello World". File name Main.jack */

class Main {
    function void main() {
        do Output.printString("Hello World");
        do Output.println();  // New Line
        return;               // The return statement is mandatory
    }
}
```

Jack comes with a standard library also know as JackOS. It will contain files that implement mathematical functions, string processing, memory management, graphics, and I/O functions.

Example 2: Procedural programming and array handling:

Arrays are treated as instances of an array class. This will be a part of the OS It simplifies the construction of the compiler :)

```
/** Inputs a sequence of integers and computes their averages */

class Main {
    function void main() {
        var Array a;
        var int length;
        var int i, sum;
        let i = 0;
        let sum = 0;
        let length = Keyboard.readInt("How many numbers? ");
        let a = Array.new(length);  // constructs the array
        while ( I < length ) {
```

```
            let a[i] = Keyboard.readInt("Enter a number: ");
            let sum = sum + a[i];
            let i = i + 1;
        }
        do Output.printString("The average is: ");
        do Output.printInt(sum / length);
        do Output.println();
        return
    }
}
```

Example 3: Abstract Data Types:

Jack has three primitive types:

- *int*
- *char*
- *boolean*

Any other data type must be implemented as a class and packaged within the JackOS

If you wanted a double/float/fraction data type then you could implement it and add it to our JackOS library!

Using Classes

An abstraction of a Fraction data type class

```
/** Represents the Fraction type and related operations (class skeleton) */

class Fraction {
    /** Constructs a (reduced) fraction from x and y*/
    constructor Fraction new(int x, int y)

    /** Returns the numerator of this fraction */
    method int getNumerator()

    /** Returns the denominator of this fraction */
    method int getDenominator()

    /** Returns the sum of this fraction and the other one */
    method Fraction plus(Fraction other)

    /** Prints this fraction in the format x/y */
    method void print()

    /** Disposes this fraction */
    method void dispose()

    /** More fraction-related methods. */
```

```
    // minus, times, div, invert, etc.
}

class Main {
    function void main() {
        var Fraction a, b, c;
        let a = Fraction.new(4, 6); // a = 2/3
        let b = Fraction.new(1, 5); // b = 1/5
        // Adds up the two fractions, and print the result
        let c = a.plus(b);   // c = a + b = 13/15
        do c.print();        // prints "13/15"
        return;
    }
}
```

Implementing Classes

fields - specify object properties (member variables)
constructors - subroutines that create new objects
methods - subroutines that operate on the current object.
Functions - class-level subroutines (static methods) that operate on no particular object.

Statement types:

- *var*
- *let*
- *do*
- *while*
- *return*

Example 4: Linked list implementation

The data type list *is implemented recursively followed by a list.*

The value null *is also considered a list.*

The Operating System

Consists of eight classes and we will go over this more in chapter 12.

9.2 The Jack Language Specification

9.2.1 Syntactical Elements

A Jack program is a sequence of tokens, separated by an arbitrary amount of white space and commends. Tokens can be symbols, reserved words, constants, and identifiers

White space and	*Space characters, newline characters, and comments are ignored* *The following comment formats are supported:*

comments	// Comment to end of line /* Comment until closing */ /** Aimed at software tools that extract API Documentation */
Symbols	() used for grouping arithmetic expressions and for enclosing argument-lists (in subroutine calls) and parameter-lists (in subroutine declaration) [] used for array indexing {} used for grouping program units and statements , Variable list separator ; Statement terminator = Assignment and comparison operator . Class membership + - * / & | ~ < > Operators
Reserved words	class, constructor, method, function - Program components int, boolean, char, void - primitive types var, static, field - Variable declaration let, do, if, else, while, return - statements true, false, null - Constant values this - Object reference
Constants	• Integer constants are values in the range 0 to 32767. Negative integers are not constant but rather expressions consisting of a unary minus operator applied to an integer constant. The resulting valid range of values is -32768 to 32767 (the former can be obtained using the expression -32767-1) • String constants are enclosed within double quote (") characters and may contain any character except new line or double quote. These characters are supplied by the OS functions String.newLine() and String.doubleQuote() • Boolean constants are true or false • The null constant signifies a null reference.
Identifiers	Identifiers are composed from arbitrary long sequences of letters (A-Z, a-z), digits (0-9), and "_". The first character must be a letter or "_". The language is case sensitive: x and X are treated as different identifiers.

9.2.2 Program Structure

A combination of classes (files) within a folder. In which one file is named Main.jack and that file also contains a main function to run. All run by the Sysinit.jack file we used in the last chapter.

```
class Name {
   field variable declaration  // Must precede the subroutine declaration
   static variable declaration // Must precede the subroutine declaration
   subroutine declarations     // Constructor, method and function
                               // declaration, in that order
}
```

9.2.3 Data Types

Primitive data types:

- int

- *char*
- *boolean*

Arrays:

Declared using OS class Arrays
indexed from 0
The elements are not types (you can have an array with int's and char's)
Use of pointers to refer to the array (the variable only holds a pointer to the reference of the array object)

Object Types:

An instant of a class.

Strings:

Declared using OS class String
An array of Characters (think C)
Use of ASCII is necessary

Type Conversion:

Type casting is left up to the compiler
Basically we can implement it in chapter 10 & 11 if we would like.

9.2.4 Variable

Kind	*Description*	*Declared in*	*Scope*
Class	*static type varName1, varName2, …;* *One copy of each static variable exists, and this copy is shared by all the class subroutine* *(like private static variables in java)*	*Class Declaration*	*The class in which they are declared*
Field	*field type varName1, varName2, …;* *Every object (instance of a class) has a private copy of the field variables* *(like member variables in java)*	*Class Declaration*	*The class in which they are declared*
Local	*var type varName1, varName2, …;* *Created when the subroutine starts running and disposed when the subroutine ends*	*Subroutine Declaration*	*The subroutine in which they are declared*
Parameter	*Represents the arguments passed to the subroutine. Treated like local variables whose values are initialized by the subroutine caller*	*Subroutine Declaration*	*The subroutine in which they are declared*

Variable visibility

Static and field variables need getters/setters to be accessed or set.

9.2.5 Statements

Statemen	*Syntax*	*Description*

t		
let	let varName = expression; let varName[index] = expression	An assignment operation The variable kind may be static, local, field, or parameter

if	if (expression) {
	statements1;
	} else {
	statements2;
	}
		Typical if statement, with an optional else clause
		The curly brackets are mandatory, even if statements are a single line

while	while (expression) {
	statements;
	}
		Typical while statement.
		The curly brackets are mandatory, even if statements are a single line

do	do function-or-method call	Used to call a function/method for its effect, ignoring its return value.
return	return expression; return;	Used to return a value from a subroutine The second form must be used by void subroutines Constructors must return the value this.

9.2.6 Expressions

A Jack Expression:

- A constant.
- A variable name in scope (static, field, local, parameter).
- The this keyword (current object).
- An array element (arr[expression]).
- A subroutine call that returns a non-void type.
- An expression prefixed by one of the unary operations - or ~:
    - - expression: arithmetic negations
    - ~ expression: boolean negation (bit-wise for integers)
- expression op expression
    - op: {+, -, *, /, &, |, <, >, =}
- (expression)

Operator priority and order of evaluation:

- Priority is only given to parenthesis. Could implement PEMDAS in compiler though :)

9.2.7 Subroutine Calls

Syntax - subroutineName(exp1, exp2, …, expn)

Function calls / Constructor calls:

```
className.functionName(exp1, exp2, …, expn);
className.constructorName(exp1, exp2, …, expn);
```

Method calls:

```
varName.methodName(exp1, exp2, …, expn);
```

methodName(exp1, exp2, …, expn);

9.2.8 Object Construction and Disposal

Construction

First declare the variableName for the said object.
The use the object constructor to instantiate the object

Destruction

Memory.deAlloc(variableName);

To avoid memory leaks you must free all constructed objects that are no longer needed!!