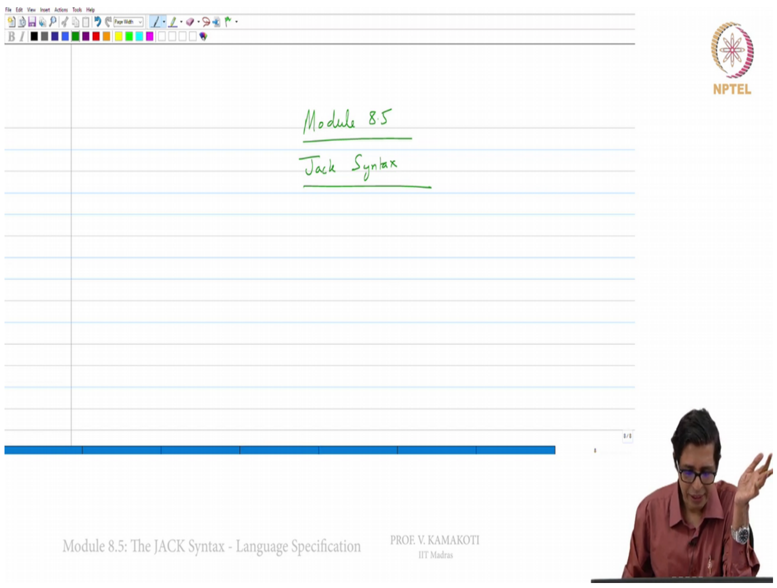
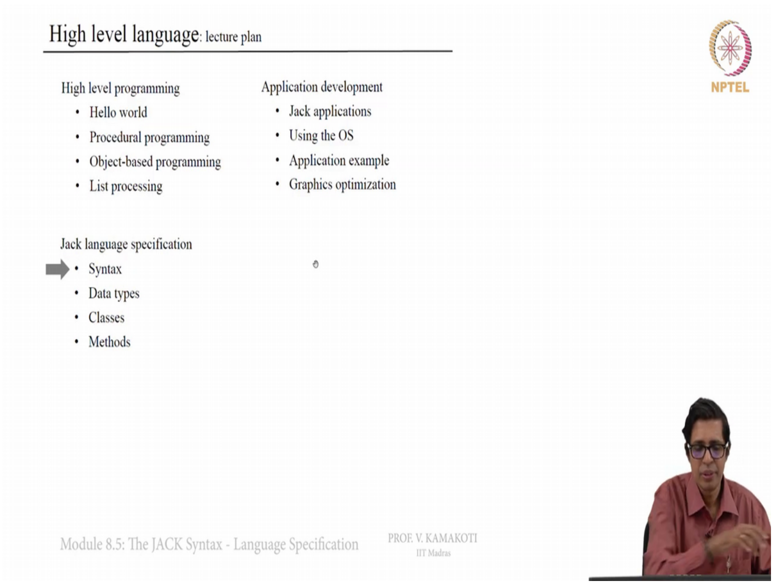
Foundations to Computer Systems Design
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Module 8.5
The Jack Syntax – Language Specification

(Refer Slide Time: 0:18)



So welcome to module 8.5 and in this we will be talking about the syntax or the grammar of the Jack language, we got some introduction of how Jack works right, some introduction of while loop, let, do, class, main we have seen so many things there and we have also executed a set of programs and understood what is happening there, now will now start looking at what we call as the Jack Syntax right, the syntax of jack and so that is what we will be covering as a part of this lecture.

(Refer Slide Time: 0:53)


High level language: lecture plan

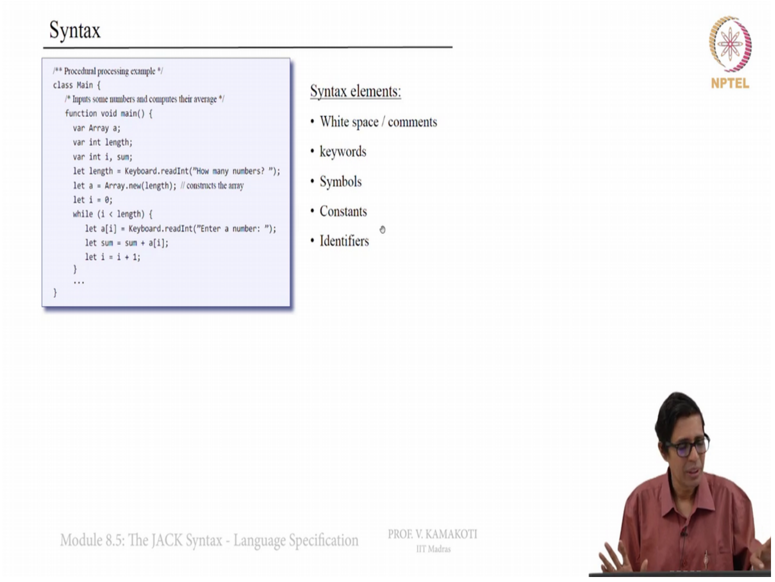
High level programming
- Hello world
- Procedural programming
- Object-based programming
- List processing

Application development
- Jack applications
- Using the OS
- Application example
- Graphics optimization

Jack language specification
- Syntax
- Data types
- Classes
- Methods

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras


Syntax

```
/** Procedural processing example */
class Main {
   /* Inputs some numbers and computes their average */
   function void main() {
      var Array a;
      var int length;
      var int i, sum;
      let length = Keyboard.readInt("How many numbers? ");
      let a = Array.new(length); // constructs the array
      let i = 0;
      while (i < length) {
         let a[i] = Keyboard.readInt("Enter a number: ");
         let sum = sum + a[i];
         let i = i + 1;
      }
      ...
}
```

Syntax elements:
- White space / comments
- keywords
- Symbols
- Constants
- Identifiers

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

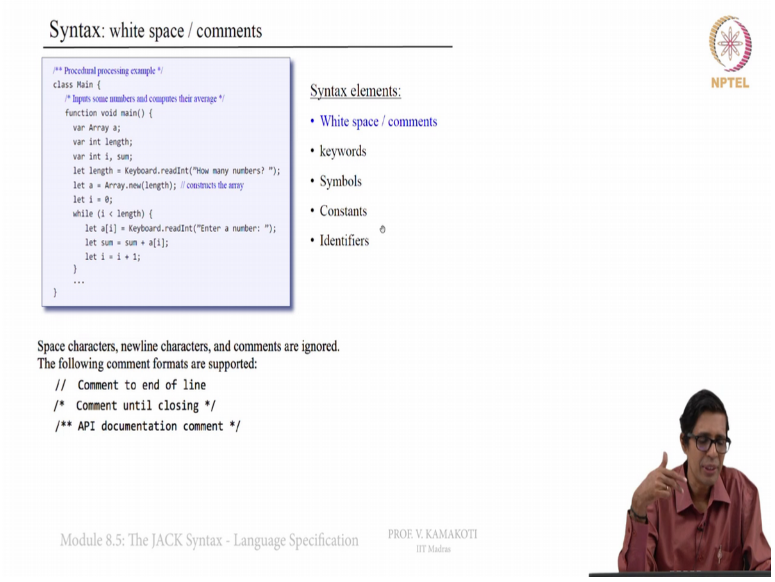
So the syntax elements are, what are the things that are present inside Jack? There are only six different things that you see, you will find some comments and whitespace, you will find keywords, what are the keywords? Where function, class, function, void, where, array, int, let okay, do while right and these are the things, these are the keywords that you will see, then you will see some symbols like you know these bracers, semi colon, dot right and then you will see some constants write like zero and then you will see identifiers like A, length, I, some right and this is syntax.

So this is a constant, this thing is how many numbers its string is a constant, but then you will see length, some, A of I etc okay, so there are only five different syntax elements in this entire Jack, see now this is the way you start understanding a programming language, you will be learning Python, you will be learning C++, C Sharp, A sharply, D sharp, E sharp etc, so many things you may be learning in life are some examples right.

The way you approach understanding the programming language in its full glory right, we need to understand it completely, it is not be halfway, now this is the way you start, so what would be this particular two models 8.5 and 8.6 slightly dry part of the course, but what you need to concentrate here and I will try to impress upon you is that the approach to understanding a programming language.

Now in this programming language I want the five different syntax elements nothing more, if I is something very easy for us to comprehend, now what are we, what is objective of this? I need ultimately to write a compiler, I have been using a compiler right, you have in project 09 we have used a compiler, now we have to basically construct a compiler of our own, to construct a compiler of our own we should know what is that, what is it that I will be exposed to, you will be exposed to only five elements in this, that is what this particular you know, discussion open for us, you will see some whitespace in comments, you will see some keywords, you will see some symbols, you will see some constant and identifiers that is all, nothing more nothing less.

(Refer Slide Time: 3:58)



Syntax: white space / comments

```
/** Procedural processing example */
class Main {
   /* Inputs some numbers and computes their average */
   function void main() {
     var Array a;
     var int length;
     var int i, sum;
     let length = Keyboard.readInt("How many numbers? ");
     let a = Array.new(length); // constructs the array
     let i = 0;
     while (i < length) {
        let a[i] = Keyboard.readInt("Enter a number: ");
        let sum = sum + a[i];
        let i = i + 1;
     }
     ...
}
```

Syntax elements:

- White space / comments
- keywords
- Symbols
- Constants
- Identifiers

Space characters, newline characters, and comments are ignored.
The following comment formats are supported:

```
//  Comment to end of line
/*  Comment until closing */
/** API documentation comment */
```

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

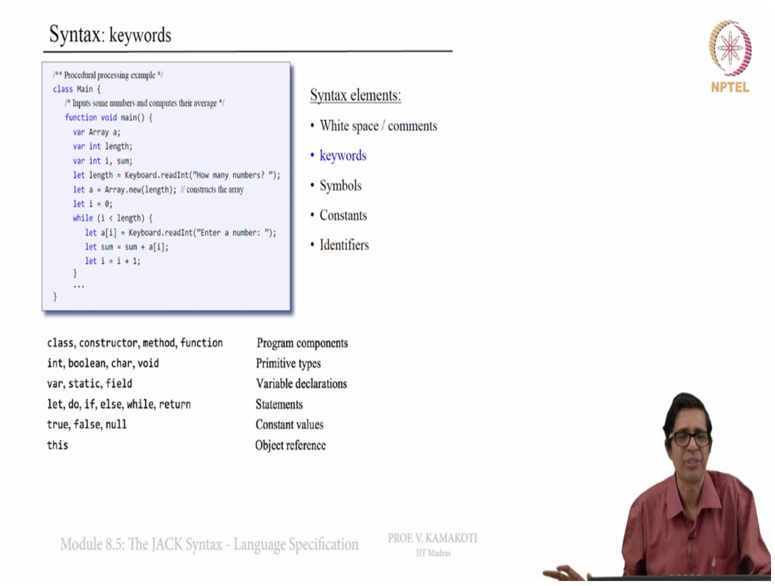
So when you are compiling all your space, characters, new line characters, comments, everythings are ignored, the following comments formats are supported I will have //, /* comment until closing and API documentation comment right, so these are the things, so in the whatever you see as blue are the comments and this will be ignored by your compiler, so when you start compiling first when you read the file, first go and remove of decomment it completely right.

So many codes this decommenting is not actually necessary because when we see the code that is return as a part of many of a even programming languages, we rarely comment right, that has been a cursive programming I should saying, when added by B. Tech I did not comment, maybe I did not do that much programming say for 30 years before, but you know even today, the amount of comment that is being returned is horrible right, we do not see single comment anywhere, so that is one thing that will,

So ultimately every academic institution in our country would have been graduating so many students, assume every student has written say some 1000 to 2000 lines of code, very optimistic for doing are very pessimistic maybe right, so imagine that total number of code that we should have generated over years right, now why are this code is not becoming public or open source today right, we are using a lot of open source by this entire course is open source right, I could not have run this code if Nand to Tetris website is not open source.

Now, why are we not able to contribute to such open source? The reason is we do not comment right, we do not write comments, we do not make effort to write comments, so one of the things that you all of you should start doing, if you want really contribute to the overall growth of computer science, in essentially open source software is start commenting right, so, but that comment is only for the human who is going to read your code and going to, you know enhance your code and for the compiler that comment is useless anyway right, but you write code for the other user not for the compiler right, so first thing in the compiler what you call as you know a syntactic analysis, I will look at the grammar of your, whatever you have written, first at the compiler will do is to decomment it.

(Refer Slide Time: 6:58)



Syntax: keywords

```
/** Procedural processing example */
class Main {
   /* Inputs some numbers and computes their average */
   function void main() {
      var Array a;
      var int length;
      var int i, sum;
      let length = Keyboard.readInt("How many numbers? ");
      let a = Array.new(length); // constructs the array
      let i = 0;
      while (i < length) {
         let a[i] = Keyboard.readInt("Enter a number: ");
         let sum = sum + a[i];
         let i = i + 1;
      }
      ...
}
```

Syntax elements:

- White space / comments
- keywords
- Symbols
- Constants
- Identifiers

class, constructor, method, function	Program components
int, boolean, char, void	Primitive types
var, static, field	Variable declarations
let, do, if, else, while, return	Statements
true, false, null	Constant values
this	Object reference

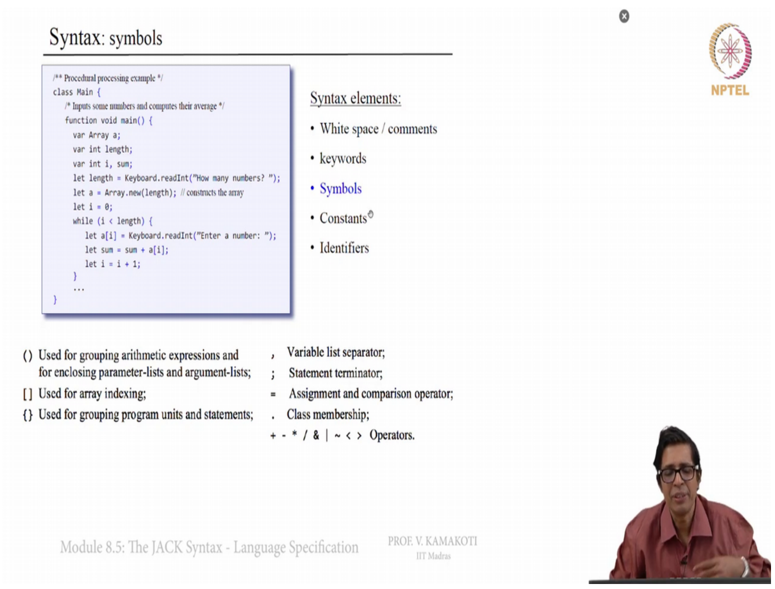
Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

Now what are the keywords here? Look at all these blue ones, class, function, void, where, int, let, while these are all keywords right, so what are the other keywords here, so there are keywords are of six types, one is program components which are class, constructor in this case we not seen a constructor here, method and function that are seen this things as a part of our prescribe.

Then the primitive types like int, boolean, char, void and then variable declarations like var, static, field, will talk about static bit later, but then field, we have seen this field, then all the statements let, do, if, else, while, return then all the constant values null, true, false, we have seen this null in the list case and true, false, and then object reference at is this right, I want to the same object this. so total number of keywords you count is 8+3 = 11+6 = 17+3 = 20, 21 keywords we have, that is all right, total keywords that is only 21,

(Refer Slide Time: 8:22)



Syntax: symbols

```
/** Procedural processing example */
class Main {
   /* Inputs some numbers and computes their average */
   function void main() {
     var Array a;
     var int length;
     var int i, sum;
     let length = Keyboard.readInt("How many numbers? ");
     let a = Array.new(length); // constructs the array
     let i = 0;
     while (i < length) {
        let a[i] = Keyboard.readInt("Enter a number: ");
        let sum = sum + a[i];
        let i = i + 1;
     }
     ...
}
```

Syntax elements:

- White space / comments
- keywords
- Symbols
- Constants
- Identifiers

()	Used for grouping arithmetic expressions and for enclosing parameter-lists and argument-lists;
[]	Used for array indexing;
{}	Used for grouping program units and statements;
,	Variable list separator;
;	Statement terminator;
=	Assignment and comparison operator;
.	Class membership;
+ - * / & | ~ < >	Operators.

Module 8.5: The JACK Syntax - Language Specification

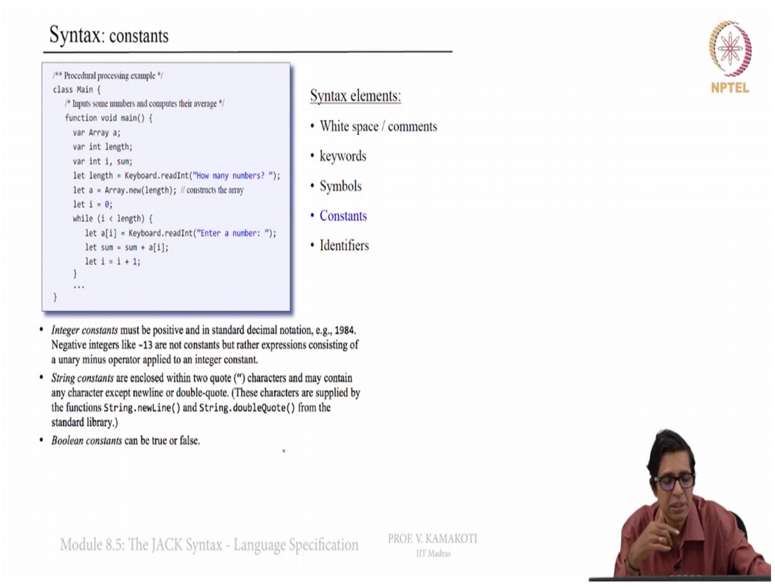
PROF. V. KAMAKOTI
IIT Madras

Then coming to symbols you have just eight classes of symbols, one which is used for grouping arithmetic expressions and for closing parameters list and argument list like you see here right, the closing and opening parenthesis as you see here, than there is a square bracket which is used for array indexing and then braces which is used for grouping program units and statements for while loop is exactly how C works, then comma for variable list separator, comma is used only for variable list separation nowhere else. I am using comma only in variable declaration can use comma.

Semi colon is for statement terminator, equal to is assignment and comparison operator both right, so I equal to length is a comparison, so that is why the I, what is the difference between this equal to, this equal to is an assignment statement, somewhere you can say, while I equal to length, that equal to is a comparison, so how do you distinguish for this assignment. I also have a let statement that is why I use let here, instead of in C we use double equal to for comparison and one equal to for assignment.

In this case I use equal to for both assignment and comparison but the way I distinguish between comparison and assignment this by using the keyword let okay, and this dot is for class membership array.new, keyboard.readint, read int right and then all the arithmetic operator which are any number +, -, *, /, &, |, ~, < and > for comparison, so totally if you say there are each symbol distinctly, there are sets of this, I am moving the mouse here, six of this, seven, eight, nine, totally nineteen symbols right, totally nineteen symbols, 21 keywords, 19 symbols.

(Refer Slide Time: 10:42)



Syntax: constants

```
/** Procedural processing example */
class Main {
   /* Inputs some numbers and computes their average */
   function void main() {
     var Array a;
     var int length;
     var int i, sum;
     let length = Keyboard.readInt("How many numbers? ");
     let a = Array.new(length); // constructs the array
     let i = 0;
     while (i < length) {
        let a[i] = Keyboard.readInt("Enter a number: ");
        let sum = sum + a[i];
        let i = i + 1;
     }
     ...
}
```

Syntax elements:

- White space / comments
- keywords
- Symbols
- Constants
- Identifiers

- *Integer constants* must be positive and in standard decimal notation, e.g., 1984. Negative integers like -13 are not constants but rather expressions consisting of a unary minus operator applied to an integer constant.
- *String constants* are enclosed within two quote (") characters and may contain any character except newline or double-quote. (These characters are supplied by the functions `String.newLine()` and `String.doubleQuote()` from the standard library.)
- *Boolean constants* can be true or false.

Module 8.5: The JACK Syntax - Language Specification

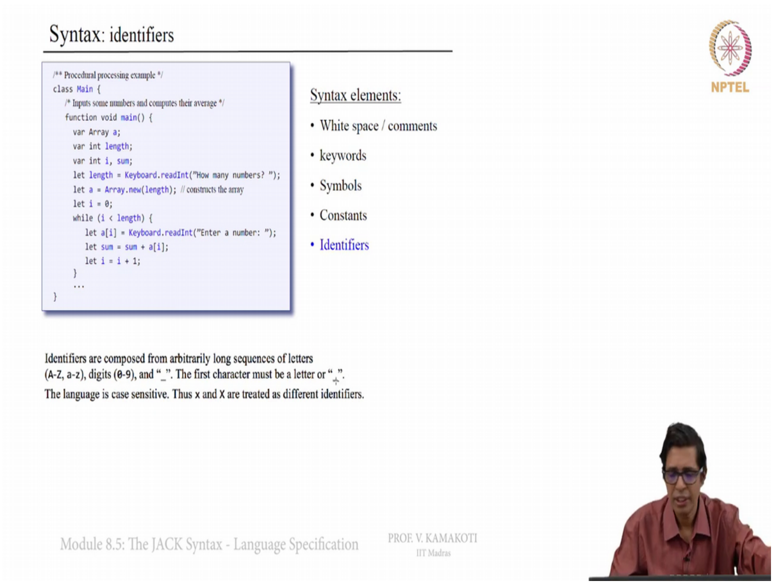
PROF. V. KAMAKOTI
IIT Madras

So the integer constants must be positive and in standard notation, negative integers like -13 are not constant, but rather expressions consisting of a unitary minus operator applied to an integer constant right, so please understand integer, so we are now talking about constants, there can be a 4 constants, the integer constant, integer constant is always positive right and negative integers like -13 or expressions, there is a unary minus, followed by 13.

String constants are in double quote and may contain any character except new line or double quote, so I cannot have a double quote inside double quote, these characters are supplied by the functions string.newline and string.doublequote from the standard library right, so if you want a new line for a string then you put string.newline and if you want a double quote string.doublequote okay.

Boolean constants can be true or false, small true or small false they can be and the null, NULL constant signifies a null reference okay, so these are all the four types of constants, please note constant is always positive constant, negative constant is an expression.

(Refer Slide Time: 12:07)



Syntax: identifiers

```
/** Procedural processing example */
class Main {
  /* Inputs some numbers and computes their average */
  function void main() {
    var Array a;
    var int length;
    var int i, sum;
    let length = Keyboard.readInt("How many numbers? ");
    let a = Array.new(length); // constructs the array
    let i = 0;
    while (i < length) {
      let a[i] = Keyboard.readInt("Enter a number: ");
      let sum = sum + a[i];
      let i = i + 1;
    }
    ...
}
```

Syntax elements:

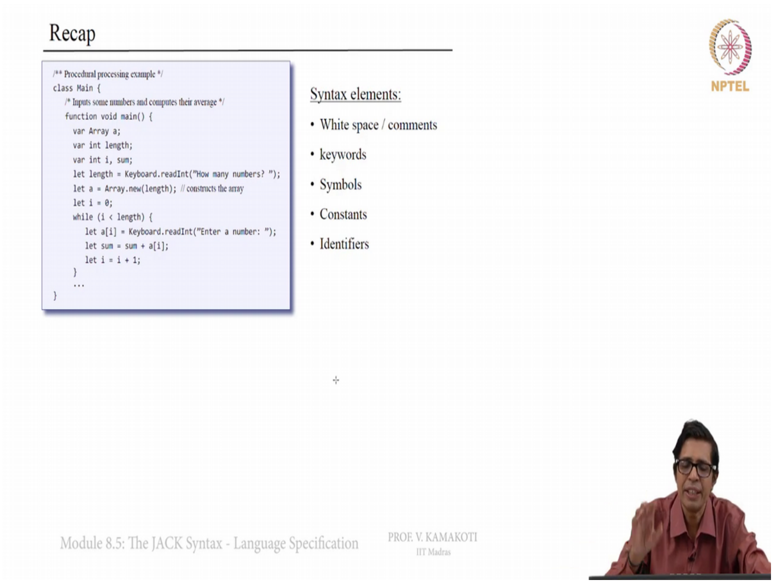
- White space / comments
- keywords
- Symbols
- Constants
- Identifiers

Identifiers are composed from arbitrarily 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. Thus x and X are treated as different identifiers.

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

Then identifiers let us be very clear here, are compose from arbitrary long sequence of letters like length, I, A, keyboard.int, so they start with A to Z or small a to z and they could have digits in between 0, 9 and they can have_, the first character must be a letter that A to Z or small a to z, A to Z or small a to z or it can be an_, the languages case-sensitive, so small I is different from capital I, the section X are treated as different identifiers, so this is about identifier.

(Refer Slide Time: 12:49)



Recap

```
/** Procedural processing example */
class Main {
  /* Inputs some numbers and computes their average */
  function void main() {
    var Array a;
    var int length;
    var int i, sum;
    let length = Keyboard.readInt("How many numbers? ");
    let a = Array.new(length); // constructs the array
    let i = 0;
    while (i < length) {
      let a[i] = Keyboard.readInt("Enter a number: ");
      let sum = sum + a[i];
      let i = i + 1;
    }
    ...
}
```

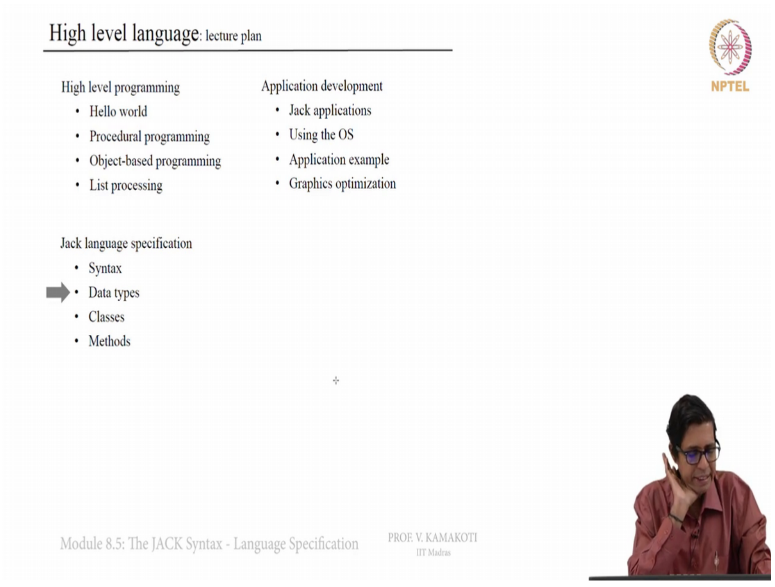
Syntax elements:

- White space / comments
- keywords
- Symbols
- Constants
- Identifiers

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

So essentially these are all the syntax elements, so keywords, symbols, constants and identifier.

(Refer Slide Time: 13:00)



High level language: lecture plan

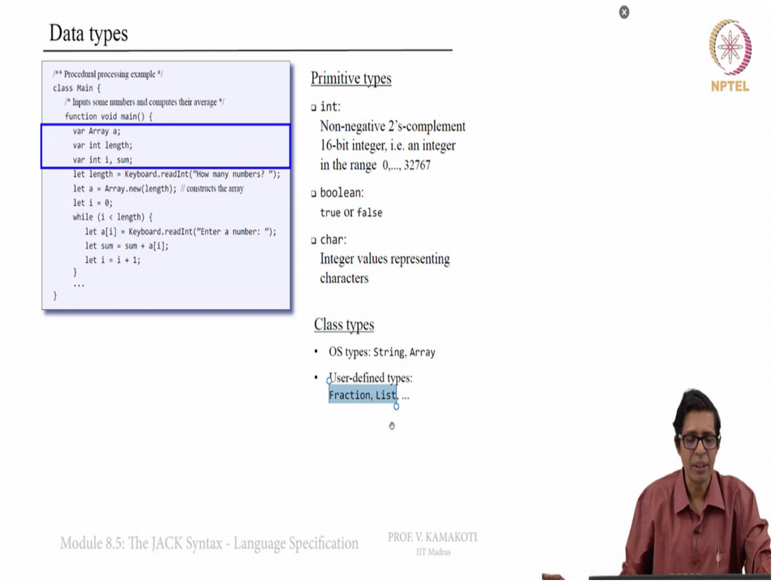
High level programming
- Hello world
- Procedural programming
- Object-based programming
- List processing

Application development
- Jack applications
- Using the OS
- Application example
- Graphics optimization

Jack language specification
- Syntax
- Data types
- Classes
- Methods

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras



Data types

```
/** Procedural processing example */
class Main {
   /* Inputs some numbers and computes their average */
   function void main() {
      var Array a;
      var int length;
      var int i, sum;
      let length = Keyboard.readInt("How many numbers? ");
      let a = Array.new(length); // constructs the array
      let i = 0;
      while (i < length) {
         let a[i] = Keyboard.readInt("Enter a number: ");
         let sum = sum + a[i];
         let i = i + 1;
      }
      ...
}
```

Primitive types

- int:
  Non-negative 2's-complement 16-bit integer, i.e. an integer in the range 0,..., 32767
- boolean:
  true or false
- char:
  Integer values representing characters

Class types

- OS types: String, Array
- User-defined types: Fraction, List, ...

Module 8.5: The JACK Syntax - Language Specification

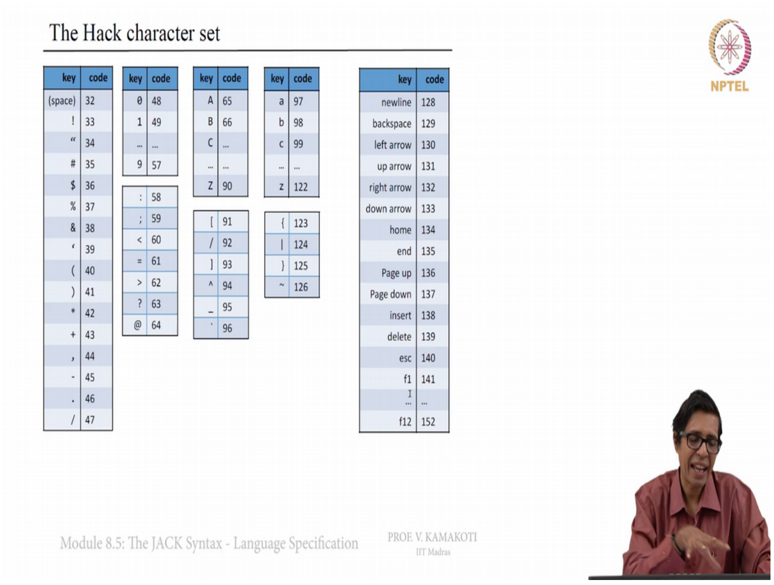
PROF. V. KAMAKOTI
IIT Madras

Now will see the different datatypes that are part of this, so I could have int which are nonnegative 2s complement 16-bit integer that is an integer in the range 0 to 32767, they are nonnegative 2s complement right, so 16-bit integer from 0 to 32767 right, so 2s complement essentially means it covers positive and negative integers right, so when I take it as 2s complement it will be, the value of that will be nonnegative for all arithmetic purposes.

The interpretation can be positive or negative, so we have, so I take you back to the module where I taught you about 2s complement arithmetic wherein a subtraction will be treated, everything is addition, subtraction, both addition and subtraction are treated as addition, so that is the interpretation of this.

So and you could have Boolean which is a primitive type instead of int here, I could have Boolean which is true or false, I could have char which is integer value representing characters again 16-bit values, now the class types are OS types which are string and array, which are provided by the operating system, so you see array here and then you can have user defined types like I have define fraction, I have define list etc, so we could have this user-defined class types right, so these are all the primitive types of data.

(Refer Slide Time: 14:37)



The Hack character set

key	code
(space)	32
!	33
"	34
#	35
$	36
%	37
&	38
'	39
(	40
)	41
*	42
+	43
,	44
-	45
.	46
/	47

key	code
0	48
1	49
...	...
9	57
:	58
;	59
<	60
=	61
>	62
?	63
@	64

key	code
A	65
B	66
C	...
...	...
Z	90
[	91
/	92
]	93
^	94
_	95
`	96

key	code
a	97
b	98
c	99
...	...
z	122
{	123
|	124
}	125
~	126

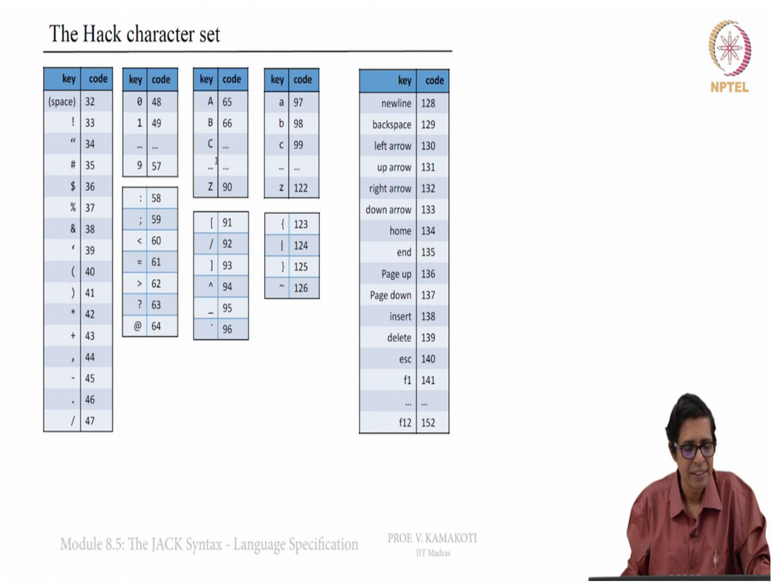
key	code
newline	128
backspace	129
left arrow	130
up arrow	131
right arrow	132
down arrow	133
home	134
end	135
Page up	136
Page down	137
insert	138
delete	139
esc	140
f1	141
...	...
f12	152

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

The hack character set is given here, so you need not know completely but, so this is how 32, 33, 34, 35, 47, 0 to 9 semi colon A to Z, so all the F1 to F12, so all the characters of 256 of ASCII may not be needed here, so the hack character set is defined for this particular system, these are all the characters that are recognized by okay, these are all the characters that are recognized by the hack computer right, so for a general system there is an characters set, this is ASCII, American standard code for information Interchange for the hack this is the character set.

(Refer Slide Time: 15:40)



The Hack character set

key	code
(space)	32
!	33
“	34
#	35
$	36
%	37
&	38
‘	39
(	40
)	41
*	42
+	43
,	44
-	45
.	46
/	47

key	code
0	48
1	49
...	...
9	57

:	58
;	59
<	60
=	61
>	62
?	63
@	64

key	code
A	65
B	66
C	...
...	...
Z	90

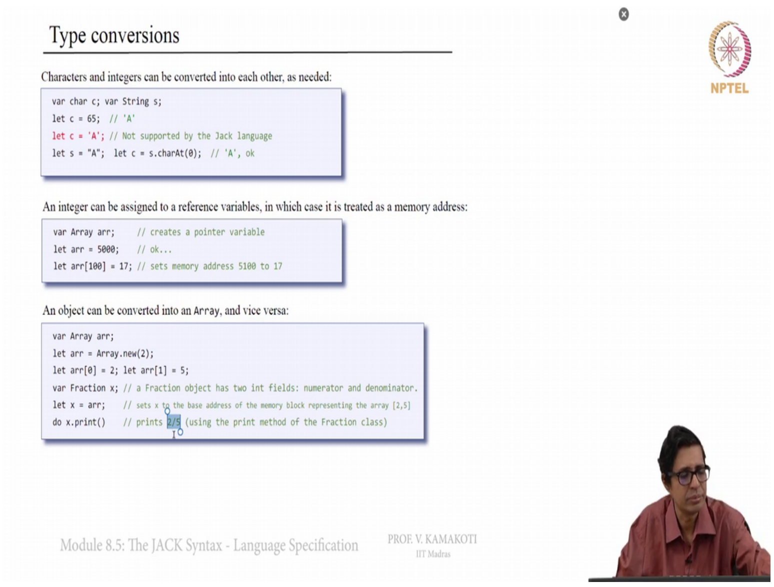
[	91
/	92
]	93
^	94
_	95
`	96

key	code
a	97
b	98
c	99
...	...
z	122

{	123
|	124
}	125
~	126

key	code
newline	128
backspace	129
left arrow	130
up arrow	131
right arrow	132
down arrow	133
home	134
end	135
Page up	136
Page down	137
insert	138
delete	139
esc	140
f1	141
...	...
f12	152

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras



Type conversions

Characters and integers can be converted into each other, as needed:

```
var char c; var String s;
let c = 65;  // 'A'
let c = 'A'; // Not supported by the Jack language
let s = "A";  let c = s.charAt(0);  // 'A', ok
```

An integer can be assigned to a reference variables, in which case it is treated as a memory address:

```
var Array arr;     // creates a pointer variable
let arr = 5000;    // ok...
let arr[100] = 17; // sets memory address 5100 to 17
```

An object can be converted into an Array, and vice versa:

```
var Array arr;
let arr = Array.new(2);
let arr[0] = 2; let arr[1] = 5;
var Fraction x; // a Fraction object has two int fields: numerator and denominator.
let x = arr;    // sets x to the base address of the memory block representing the array [2,5]
do x.print()    // prints 2/5 (using the print method of the Fraction class)
```

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

So the type conversion we have already seen right, if you assign a floating-point number to an, suppose I say, float I and int J and I equal to J so what will happen is J will be, J is an integer assigned to you are floating-point number, so J will be converted to your floating-point number and assigned to I, suppose I say J equal to I can in some compilers will make I which is an floating-point number it will make it as a integer and assign it to J, so normally and I am assigning one variable to another variable the one which is on the left inside, which is assigned some value, it will be of type A on right inside can be of type B, type B normally gets converted to type A or it gets casted to type A and assigned.

So this type of type conversion can also happen in the case of Jack, so here the characters and integers can be converted within each other right, where, char C, so this is how we define a character where string is, this is how we define a string, C equal to 65, let C equal to A both are same, let S equal to A or C equal to.char at 0, char at 0 is, 0 is A and if you go to the character set, if you go to the S.char at 0, the 0th location A is stored because S is a string, S of 0 will be A, S of 1 will be null right, S equal to A.

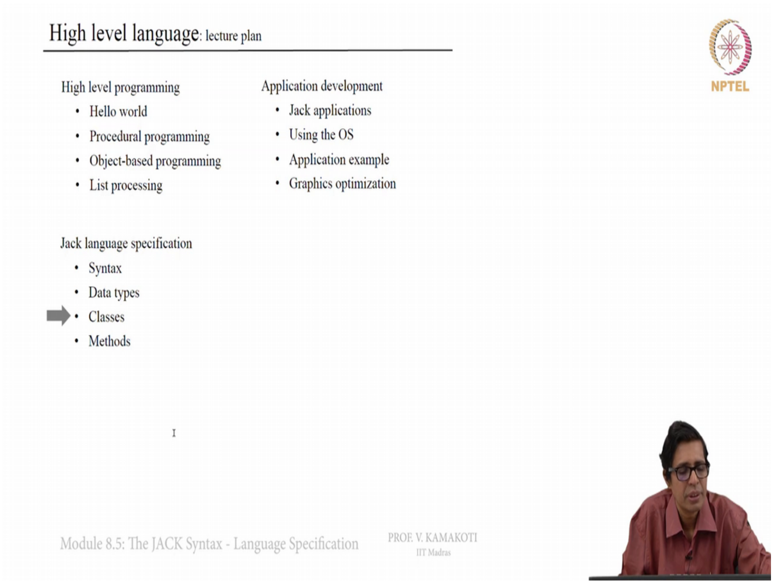
Now the char at 0 is A, so C equal to S essentially means this, so char at, suppose I define a string, string is an OS as we see here, string is an OS type, so the string has lot of things inside, we looked at that in great detail, but just to tell you here, one of the function, one of the method inside a string is char at location, so the char at location 0 is capital A and that is a character which I can assign, so this is the way I can play between strings and characters and integers interoperability between them and integer can be assigned to your reference variables in which case is treated as the memory address.

For example valid, array equal to ARR, ARR is equal to 5000 means ARR 0 is stored at 5000 address, so this is memory address, ARR of 100 equal to 17 that means there is this 17 stored? 5000 stores ARR of 0, so 5100 will store 17 right, so this is how I can assign, so if I say where array ARR just ARR this is also equivalent from C, we have borrow it from C only, just ARR is the starting address of the first element of the array, that is the 0th element of the array.

So when I say let ARR is equal to 5000, ARR 0 will be stored at 5000, ARR 1 will be 5001, so if I say ARR of 100 as you see C is equal to 17, so 5100 is stored the value 17, now another in the a type conversion is where, array ARR, let ARR is equal to array.new of 2 right, ARR 0 is equal to 2, let ARR 1 equal to 5 right, so this is how, so I have a declared an array ARR in which there are two elements and the 0th element I made it 2 and the first element I have made it 5.

Now I have where fraction X right, the moment I say fraction again that also requires two integers for numerator and denominator, I can say X is equal to ARR, X is equal to ARR means this is like pointer aliasing, this X also is a pointer to a location, as we saw in our implementation X is a pointer to a location from which there are two integers that are stored, now I say let X is equal to ARR, now I have alias this ARR and X right, I have aliases this ARR and X to point to the same thing, so this is something like you know, we can call it upper and over loading or casting of pointers both are same right, so this X is equal to ARR, so now I say X.print, it will print numerator as 2 and denominator as 5, it will print 2 x 5.

(Refer Slide Time: 20:58)



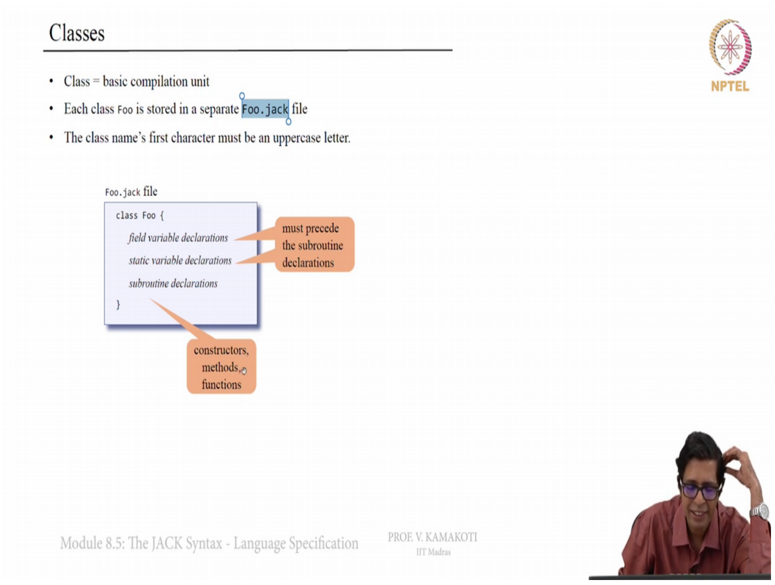
High level language: lecture plan

High level programming
- Hello world
- Procedural programming
- Object-based programming
- List processing

Application development
- Jack applications
- Using the OS
- Application example
- Graphics optimization

Jack language specification
- Syntax
- Data types
- Classes
- Methods

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras



Classes

- Class = basic compilation unit
- Each class Foo is stored in a separate Foo.jack file
- The class name's first character must be an uppercase letter.

Foo.jack file

```
class Foo {
   field variable declarations
   static variable declarations
   subroutine declarations
}
```

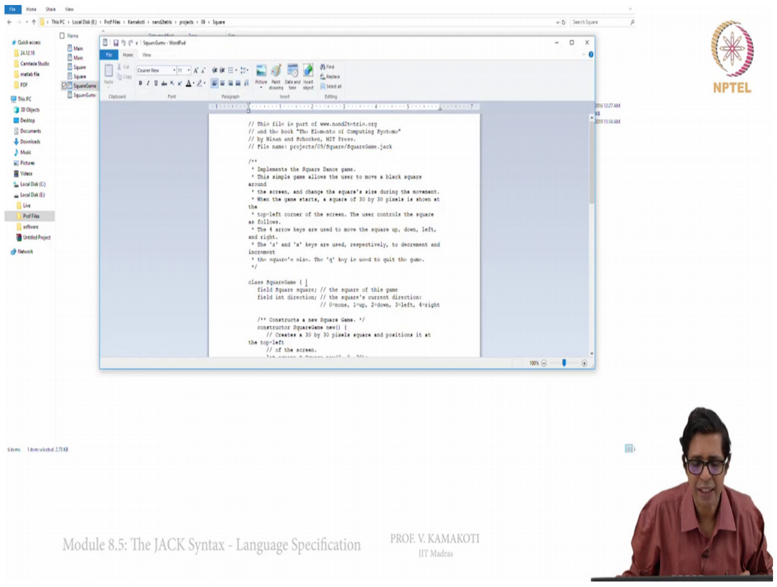
must precede the subroutine declarations
constructors, methods, functions

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

Now how does a class look? The class actually has the class foo, it has field variable declaration, it has static variable declarations and it has subroutine declarations, in the subroutine there are, so the field and static must proceed the subroutine declaration, what is static we will see slightly, so each class is stored in a separate Jack file, so for every class I have a Jack file and this is a naming conversion, in the class name is foo you call that file as foo.jack.

So if I main.jack the class name inside that will be main, we have Square.class the class name inside that if the square etc right, so and similarly subroutine declaration, so there are three types of subroutine, I could have constructor, I could have methods and functions, we have basically distinguished this three in our example.

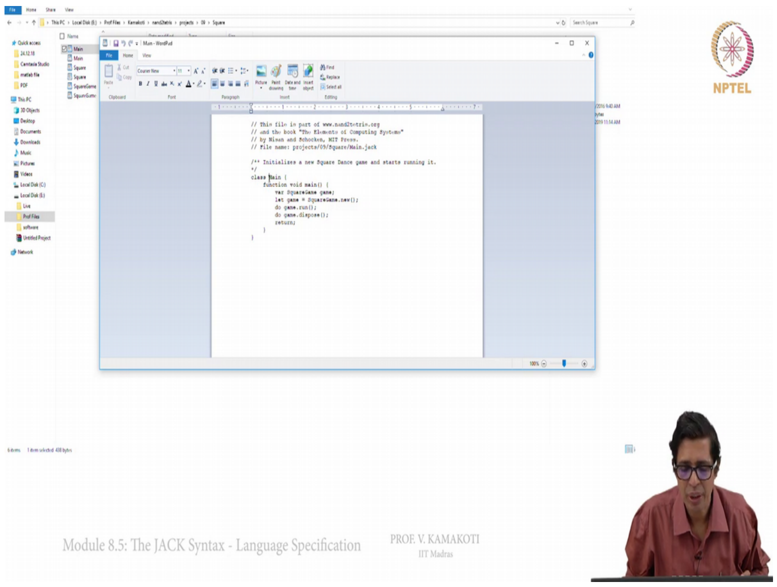
(Refer Slide Time: 22:10)



```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/Square/SquareGame.jack

/**
 * Implements the Square Dance game.
 * This simple game allows the user to move a black square
around
 * the screen, and change the square's size during the movement.
 * When the game starts, a square of 30 by 30 pixels is shown at
the
 * top-left corner of the screen. The user controls the square
as follows.
 * The 4 arrow keys are used to move the square up, down, left,
and right.
 * The 'z' and 'x' keys are used, respectively, to decrement and
increment
 * the square's size. The 'q' key is used to quit the game.
 */

class SquareGame {
   field Square square; // the square of this game
   field int direction; // the square's current direction:
                        // 0=none, 1=up, 2=down, 3=left, 4=right

   /** Constructs a new Square Game. */
   constructor SquareGame new() {
      // Creates a 30 by 30 pixels square and positions it at
the top-left
      // of the screen.
```

Module 8.5: The JACK Syntax - Language Specification

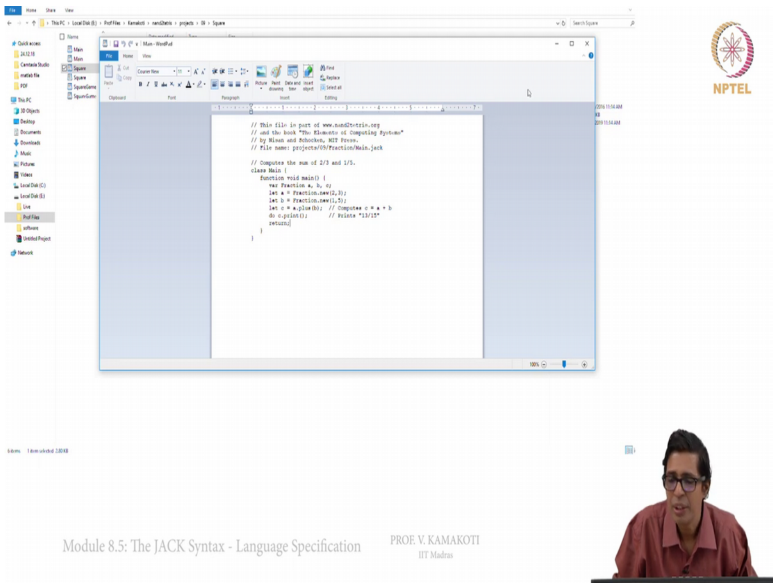
PROF. V. KAMAKOTI
IIT Madras



```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/Square/Main.jack

/** Initializes a new Square Dance game and starts running it.
*/
class Main {
    function void main() {
        var SquareGame game;
        let game = SquareGame.new();
        do game.run();
        do game.dispose();
        return;
    }
}
```

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras



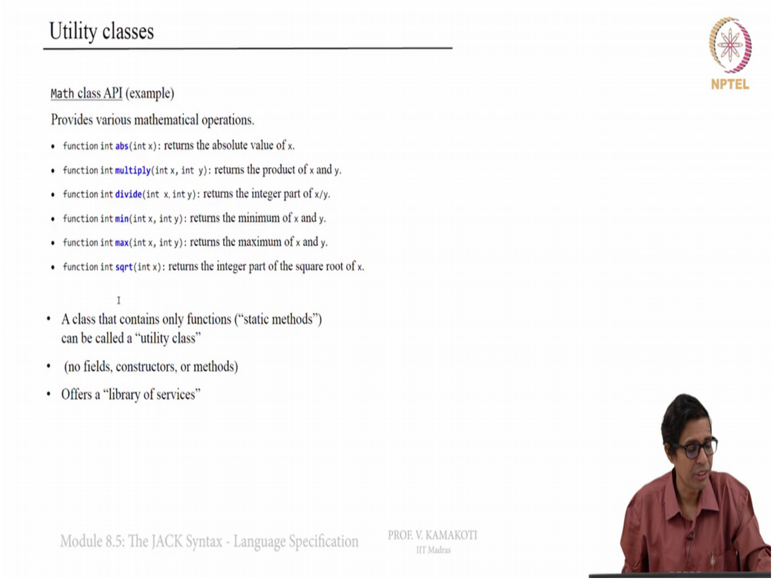
```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/Fraction/Main.jack

// Computes the sum of 2/3 and 1/5.
class Main {
   function void main() {
      var Fraction a, b, c;
      let a = Fraction.new(2,3);
      let b = Fraction.new(1,5);
      let c = a.plus(b);  // Computes c = a + b
      do c.print();       // Prints "13/15"
      return;
   }
}
```

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

So just to take you back and show you one example here, suppose I go to the projects and let us take the square, so there are three jack files, the squaregame.jack, now you have the class call square game, inside squaregame.jack, similarly, if you say main.jack this main.jack you have the class call main, similarly you have, you do not have anything else, if you have the square.jack, you have a class call square inside and nothing more, so this naming convention we need to follow as a part of, it will be followed as a part of every jack program.

(Refer Slide Time: 23:04)



Utility classes

Math class API (example)

Provides various mathematical operations.

- `function int abs(int x)`: returns the absolute value of x.
- `function int multiply(int x, int y)`: returns the product of x and y.
- `function int divide(int x, int y)`: returns the integer part of x/y.
- `function int min(int x, int y)`: returns the minimum of x and y.
- `function int max(int x, int y)`: returns the maximum of x and y.
- `function int sqrt(int x)`: returns the integer part of the square root of x.

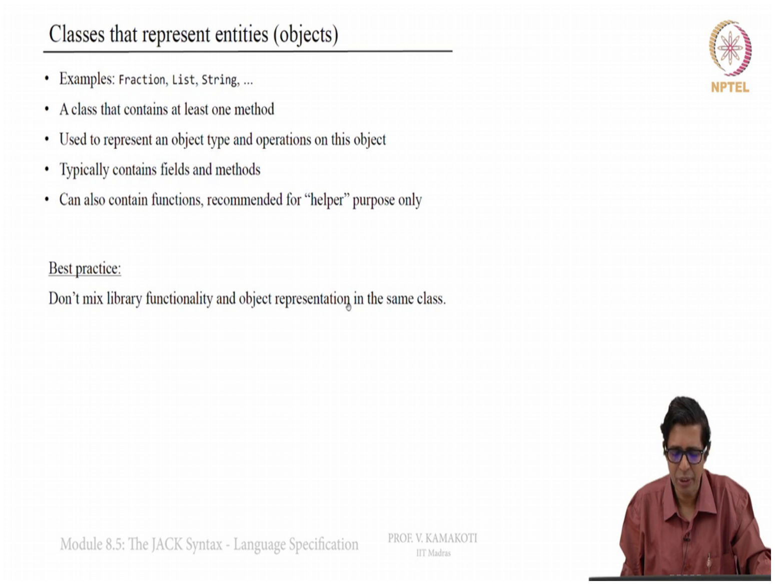
- A class that contains only functions ("static methods") can be called a "utility class"
- (no fields, constructors, or methods)
- Offers a "library of services"

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

So these are all the APIs that are already available as a part of the operating system is giving you this libraries, the math.h you are using right in your C, similarly what will be our math.h, so I will have abs absolute value multiplied, divided, main, max, square root these are all their already, so this is call utility class and there will be no constructor or methods for this, they are basically static methods, they are just methods, they are function int and so we can use them, So this is a library of services like how you are math.h we can use this.

(Refer Slide Time: 23:47)

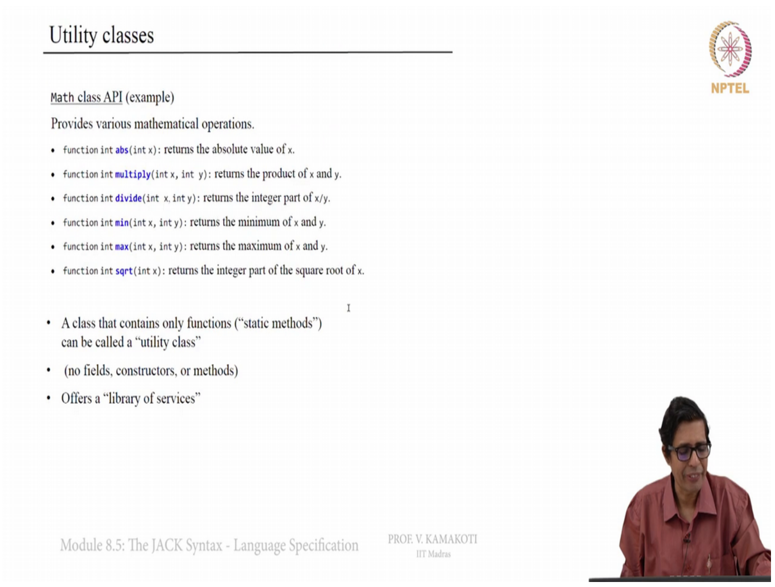


Classes that represent entities (objects)

- Examples: `Fraction`, `List`, `String`, ...
- A class that contains at least one method
- Used to represent an object type and operations on this object
- Typically contains fields and methods
- Can also contain functions, recommended for "helper" purpose only

Best practice:
Don't mix library functionality and object representation in the same class.

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras



Utility classes

`Math` class API (example)
Provides various mathematical operations.

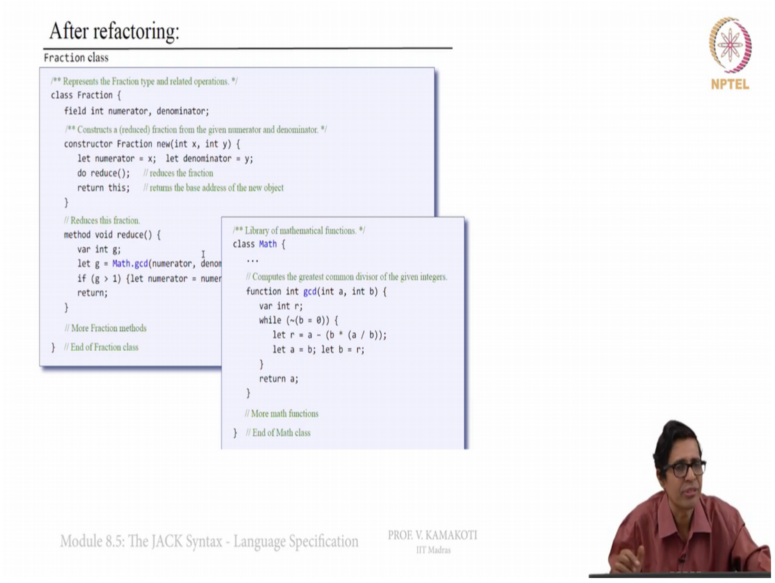
- `function int abs(int x)`: returns the absolute value of x.
- `function int multiply(int x, int y)`: returns the product of x and y.
- `function int divide(int x, int y)`: returns the integer part of x/y.
- `function int min(int x, int y)`: returns the minimum of x and y.
- `function int max(int x, int y)`: returns the maximum of x and y.
- `function int sqrt(int x)`: returns the integer part of the square root of x.

- A class that contains only functions ("static methods") can be called a "utility class"
- (no fields, constructors, or methods)
- Offers a "library of services"

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

Then you have fraction, list, string these are all classes that will have at least one method and used to represent an object type and operations can also contain functions recommended for help purpose only, so all these things, so he have seen this fraction, list, string etc, so we should not mix this library functionality and object representation in the same class, okay this is different and this is different, this is object representation, okay.

(Refer Slide Time: 24:20)



After refactoring:

Fraction class

```
/** Represents the Fraction type and related operations. */
class Fraction {
   field int numerator, denominator;
   /** Constructs a (reduced) fraction from the given numerator and denominator. */
   constructor Fraction new(int x, int y) {
      let numerator = x;  let denominator = y;
      do reduce();     // reduces the fraction
      return this;     // returns the base address of the new object
   }
   // Reduces this fraction.
   method void reduce() {
      var int g;
      let g = Math.gcd(numerator, denom
      if (g > 1) {let numerator = numer
      return;
   }
   // More Fraction methods
}  // End of Fraction class
```

```
/** Library of mathematical functions. */
class Math {
   ...
   // Computes the greatest common divisor of the given integers.
   function int gcd(int a, int b) {
      var int r;
      while (~(b = 0)) {
         let r = a - (b * (a / b));
         let a = b; let b = r;
      }
      return a;
   }
   // More math functions
}  // End of Math class
```

Module 8.5: The JACK Syntax - Language Specification

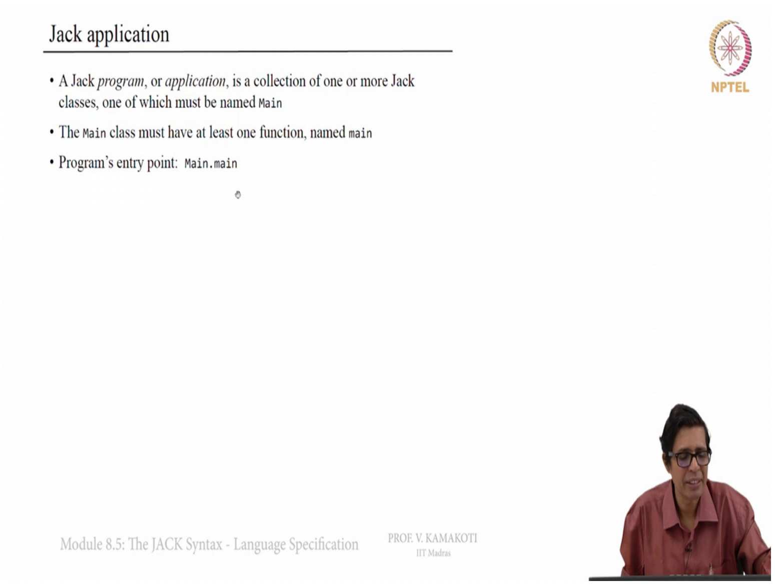
PROF. V. KAMAKOTI
IIT Madras

So this is a very bad example, in our fraction itself we had a function right, which is an utility function right, instead of doing it, so it was as the part of the fraction, so somebody has to instantiate a fraction to start using it and nobody else can use it from outside, only the this fraction.gcd can use it, rather than I could have an another class math right, which has this function in gcd and I can say math.gcd right, so because gcd is a function across many languages, across many objects right, I am not using gcd just for reducing fractions, I can use gcd for say some cryptography or some number theory computations later right.

So this is a very bad design, so I have put this gcd inside a fraction and I am using it, rather than I could put gcd as part of the math, library of mathematical function and just say math.gcd, then this should have been very useful for us. Okay, so in some essence there was one small correction to what I told in the earlier things, a function can also be used by an external user, it is not that I do not only methods can used, but a function can also be used by the external users.

So this is a math class, so this concept of trying to move out unnecessary things or generic things into a common library that concept is called refractory right, so refractory is a way by which the size of the classes gets reduced and irrelevant or more generic things actually move out into a better class, into a class of functions, so that is something that we need to keep in mind.

(Refer Slide Time: 26:33)



Jack application

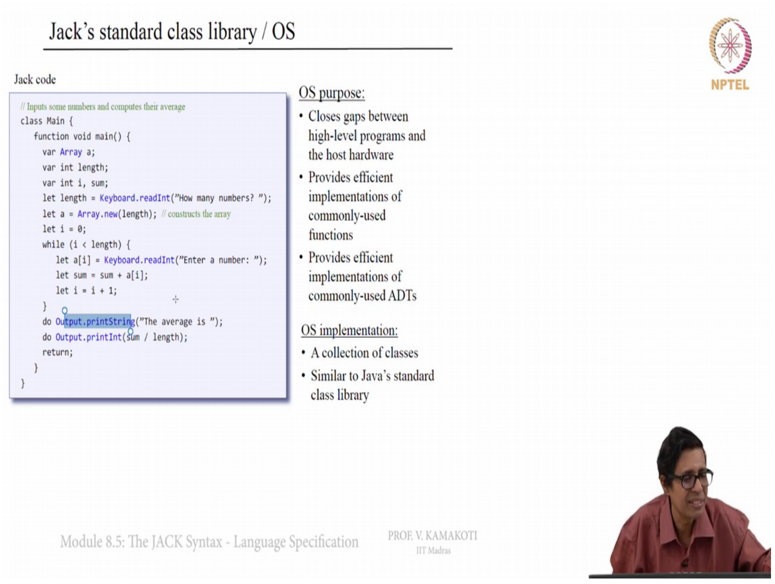
- A Jack *program*, or *application*, is a collection of one or more Jack classes, one of which must be named `Main`
- The `Main` class must have at least one function, named `main`
- Program's entry point: `Main.main`

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

Right, so finally we come to the jack application, a jack program or application is a collection of one or more jack classes, one of which must be named main, the main class must have at least one function named main and the programs entry point is main.main.

(Refer Slide Time: 26:50)



Jack's standard class library / OS

Jack code

```
// Inputs some numbers and computes their average
class Main {
   function void main() {
     var Array a;
     var int length;
     var int i, sum;
     let length = Keyboard.readInt("How many numbers? ");
     let a = Array.new(length); // constructs the array
     let i = 0;
     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);
     return;
   }
}
```

OS purpose:

- Closes gaps between high-level programs and the host hardware
- Provides efficient implementations of commonly-used functions
- Provides efficient implementations of commonly-used ADTs

OS implementation:

- A collection of classes
- Similar to Java's standard class library

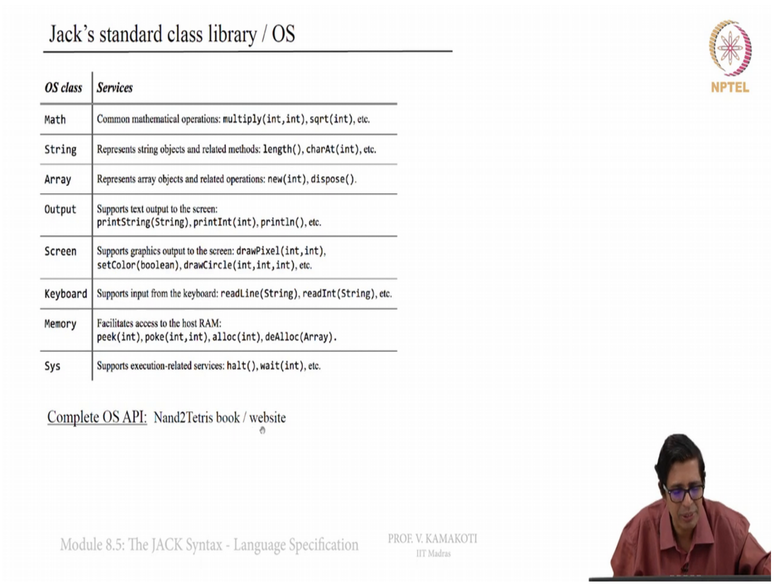
Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

So what is the OS do hear? The OS gives you lot of things like for example array, it creates an space for array.new create space for you, keyboard.readint, it allows you, it interfaces the keyboard with the program, it interfaces output.printstring, it interfaces the screen with the program, so what you see here is the purpose of having an OS, it closes the gaps between high-level programs and the host hardware, it provides efficient implementation of commonly used to functions like you know your math function can be part of OS and also all your allocation of memory and etc, provides efficient implementation of commonly used abstract datatypes like your int etc right, and the OS in turn, OS itself is a software and how do you implement the OS? It will be a collection of classes similar to a Java standard class library.

Like keyboard is a class which has read int, read string etc, array is a class where it has new, dispose etc, output is a class which has print string, print int, print LN new line etc right, so why itself will be a collection of classes, we see in project 12 some very good implementation of OS reading.

(Refer Slide Time: 28:19)



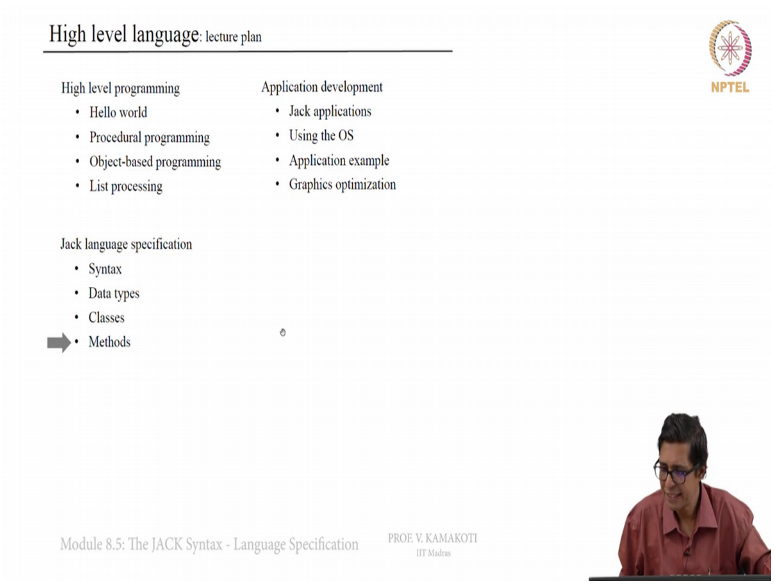
Jack's standard class library / OS

OS class	*Services*
`Math`	Common mathematical operations: `multiply(int,int)`, `sqrt(int)`, etc.
`String`	Represents string objects and related methods: `length()`, `charAt(int)`, etc.
`Array`	Represents array objects and related operations: `new(int)`, `dispose()`.
`Output`	Supports text output to the screen: `printString(String)`, `printInt(int)`, `println()`, etc.
`Screen`	Supports graphics output to the screen: `drawPixel(int,int)`, `setColor(boolean)`, `drawCircle(int,int,int)`, etc.
`Keyboard`	Supports input from the keyboard: `readLine(String)`, `readInt(String)`, etc.
`Memory`	Facilitates access to the host RAM: `peek(int)`, `poke(int,int)`, `alloc(int)`, `deAlloc(Array)`.
`Sys`	Supports execution-related services: `halt()`, `wait(int)`, etc.

Complete OS API: Nand2Tetris book / website

So the jack standard class library which is provided by the OS like what you see your stdio.h, math.h, string.h all those things that you see that is a part of C language, so OS also, the jack OS also provides math, string, array, output, screen, keyboard, memory, sys etc, so we will see some of this the complete OS API is available as a part of your project, we will show you very shortly on this.

(Refer Slide Time: 28:52)



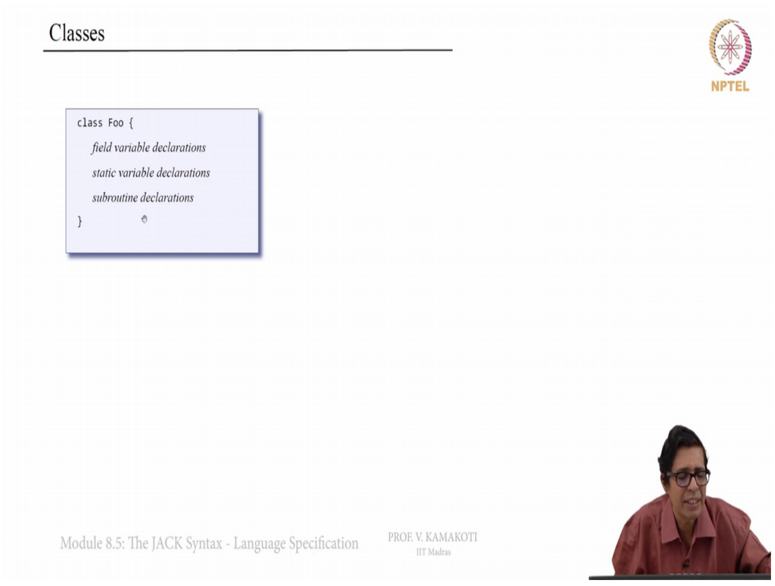
High level language: lecture plan

High level programming
- Hello world
- Procedural programming
- Object-based programming
- List processing

Application development
- Jack applications
- Using the OS
- Application example
- Graphics optimization

Jack language specification
- Syntax
- Data types
- Classes
- Methods

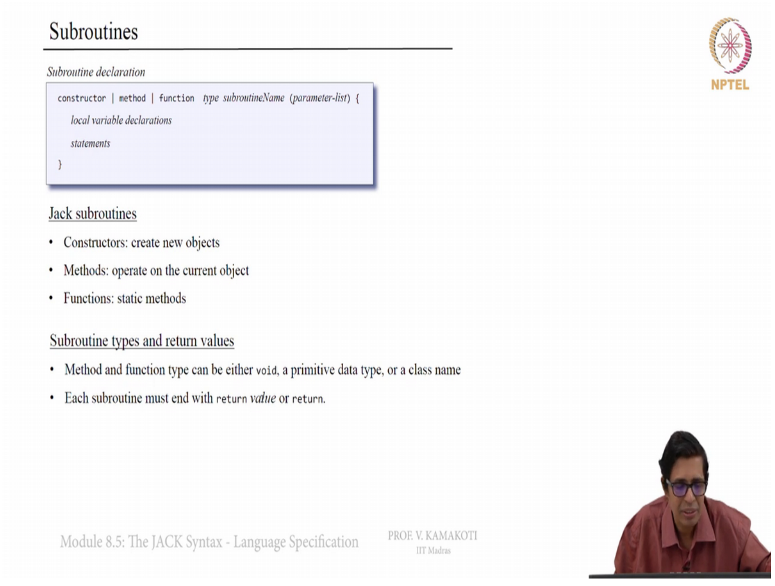
Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras



Classes

```
class Foo {
    field variable declarations
    static variable declarations
    subroutine declarations
}
```

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras



Subroutines

Subroutine declaration

```
constructor | method | function  type subroutineName (parameter-list) {
    local variable declarations
    statements
}
```

Jack subroutines
- Constructors: create new objects
- Methods: operate on the current object
- Functions: static methods

Subroutine types and return values
- Method and function type can be either `void`, a primitive data type, or a class name
- Each subroutine must end with `return` *value* or `return`.

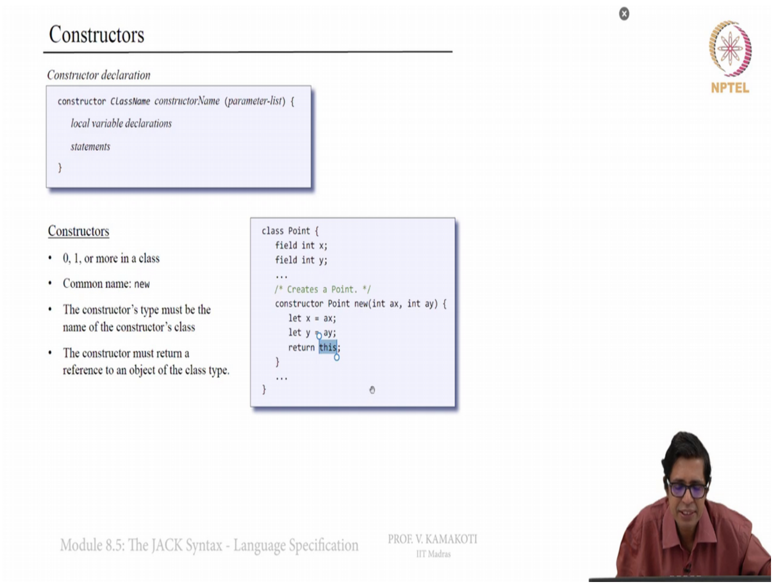
Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

The last thing is about methods, so class foo some named foo, so we are field variable, static variable and subroutine declarations, now we are talking about subroutine declarations, so what are the methods here? There are three types of methods one can be a constructor, a method or a function, the subroutines can be of three types constructor, method or functions and then you have a subroutine name forward by the parameter list, then inside you will have local variable declarations, statement, etc right.

So what others jack subroutines? Constructor which will create new objects, methods, which will operate on the current object, functions are static methods which can be used also, which will not change the content of the current objects, so the static method will not change, it will basically, it will take some values and give you a new value and the methods will use its function to change the context of the current object right.

So the static methods means that the function will take some value, compute a new value that, this function will not change their content of the different local variables inside the, content of the different fields inside the class, that is why we call it as static methods, so the subroutine types and return values you could have method and function type can be either void or a primitive datatype or a class name right, so and each subroutine must end with return some value or just return, so when it is void just a return, if it is some primitive data type is a return value or its a class name also you return some value, which is the same class.

(Refer Slide Time: 30:47)



Constructors

Constructor declaration

```
constructor ClassName constructorName (parameter-list) {
    local variable declarations
    statements
}
```

Constructors

- 0, 1, or more in a class
- Common name: new
- The constructor's type must be the name of the constructor's class
- The constructor must return a reference to an object of the class type.

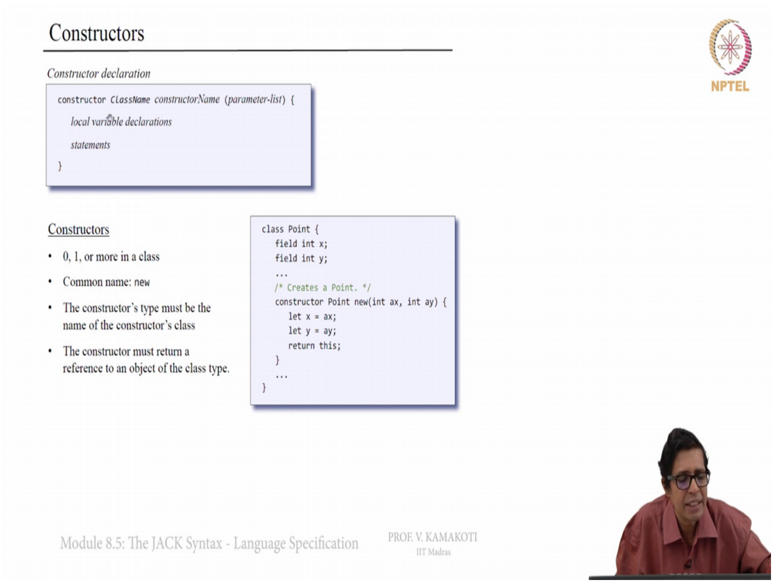
```
class Point {
   field int x;
   field int y;
   ...
   /* Creates a Point. */
   constructor Point new(int ax, int ay) {
      let x = ax;
      let y = ay;
      return this;
   }
   ...
}
```

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

Right, now the constructors basically looks like this, constructor, class name, construct our name followed by parameter list, so this is how class point, so I want to declare a point it has a field int X, int Y, constructor point new int ax, int ay, so whenever I call, when I declare okay some point A, point P, when I say p.new of 2,3 X coordinate of P becomes 2 and Y coordinate becomes 3 right and every time this returns this object, so if you create a.new will return that object A itself. That is why we say return this. Okay.

(Refer Slide Time: 31:42)



Constructors

Constructor declaration

```
constructor ClassName constructorName (parameter-list) {
    local variable declarations
    statements
}
```

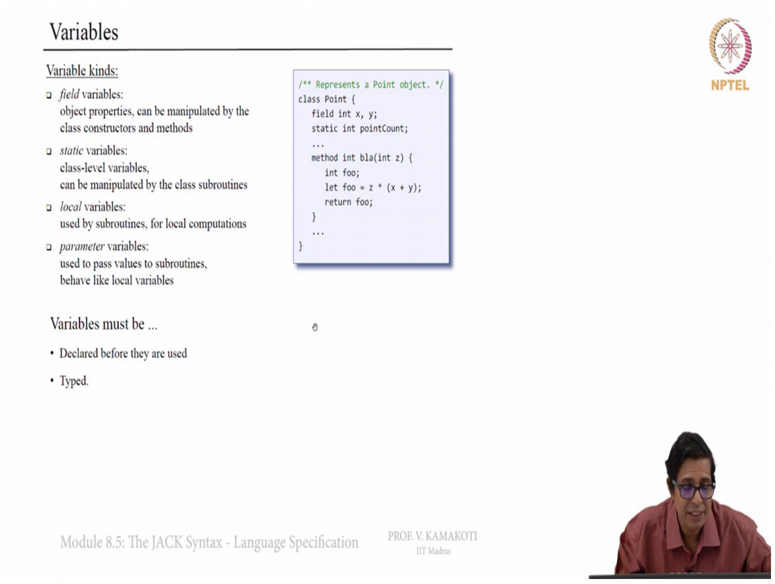
Constructors

- 0, 1, or more in a class
- Common name: new
- The constructor's type must be the name of the constructor's class
- The constructor must return a reference to an object of the class type.

```
class Point {
   field int x;
   field int y;
   ...
   /* Creates a Point. */
   constructor Point new(int ax, int ay) {
      let x = ax;
      let y = ay;
      return this;
   }
   ...
}
```

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras



Variables

Variable kinds:

- *field* variables:
  object properties, can be manipulated by the class constructors and methods
- *static* variables:
  class-level variables,
  can be manipulated by the class subroutines
- *local* variables:
  used by subroutines, for local computations
- *parameter* variables:
  used to pass values to subroutines,
  behave like local variables

```
/** Represents a Point object. */
class Point {
   field int x, y;
   static int pointCount;
   ...
   method int bla(int z) {
      int foo;
      let foo = z * (x + y);
      return foo;
   }
   ...
}
```

Variables must be ...

- Declared before they are used
- Typed.

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

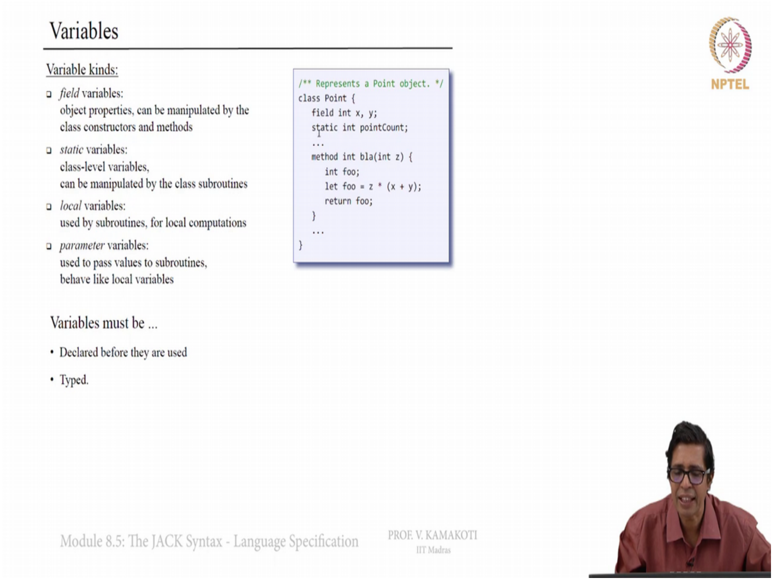
Now we have the variables again inside this, the local variables can be of, the total number of variables inside a class can be of four types, there can be V field, variables which are belonging to the class, then there can be a local variables that belong to the function right, this int block, so this foo will be belonging to this function, it will just do after this method goes it will go off right and then there are parameter list like int inside, etc, which will be supplied to this function.

Then there are static variables which are like point count, suppose I am class point basically is going to declare some set of points right, I am going to instantiate number of points, at some point I want to find out how many points I have instantiated right, so that is how I have this static int point count, so I have a class point and I want to create say 100 points, 1000 points, 10,000 points many number of points, at any stage I want to find out how many points I have created, so I had this static int point count,

Whenever I am constructing a new point right, I say point count equal to point count +1 and that will be remembered right, this X and Y are new for every time I am creating a point this X and Y will be taking the value of that new point but this point count will be remembered, so every time I create a new function I can say point count equal to point count +1, at any point, anywhere in the program. I can ask this point.point count and that will tell me what is the number of points that are basically instantiated is that point.

So what we mean by a static point count, static int point count right, so we have been talking about static variables, where those variables originate, these are the ways by which they originate right, so there are these four types of variables field, static, local and parameter, the variable must be declared before they are used, so I were to declare int X, Y, I cannot use a just X, Y and then later declare and the variables was strongly typed right.

(Refer Slide Time: 34:12)



Variables

Variable kinds:

- *field* variables:
  object properties, can be manipulated by the class constructors and methods
- *static* variables:
  class-level variables,
  can be manipulated by the class subroutines
- *local* variables:
  used by subroutines, for local computations
- *parameter* variables:
  used to pass values to subroutines,
  behave like local variables

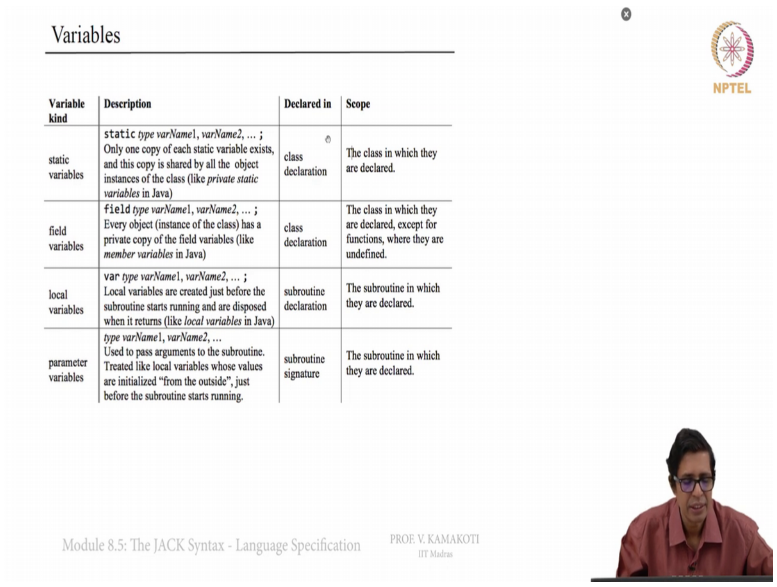
```
/** Represents a Point object. */
class Point {
   field int x, y;
   static int pointCount;
   ...
   method int bla(int z) {
      int foo;
      let foo = z * (x + y);
      return foo;
   }
   ...
}
```

Variables must be ...

- Declared before they are used
- Typed.

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

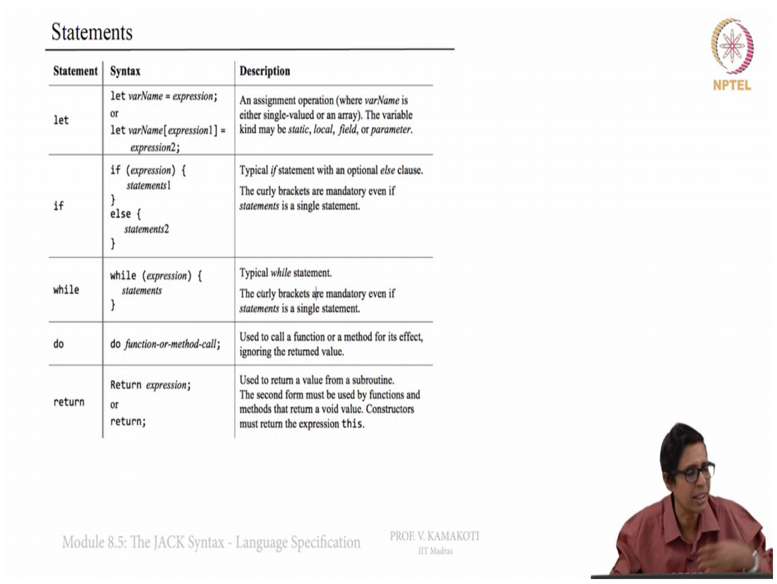

Variables

Variable kind	Description	Declared in	Scope
static variables	`static` *type varName*1, *varName2*, … ; Only one copy of each static variable exists, and this copy is shared by all the object instances of the class (like *private static variables* in Java)	class declaration	The class in which they are declared.
field variables	`field` *type varName*1, *varName2*, … ; Every object (instance of the class) has a private copy of the field variables (like *member variables* in Java)	class declaration	The class in which they are declared, except for functions, where they are undefined.
local variables	`var` *type varName*1, *varName2*, … ; Local variables are created just before the subroutine starts running and are disposed when it returns (like *local variables* in Java)	subroutine declaration	The subroutine in which they are declared.
parameter variables	*type varName*1, *varName2*, … Used to pass arguments to the subroutine. Treated like local variables whose values are initialized "from the outside", just before the subroutine starts running.	subroutine signature	The subroutine in which they are declared.

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

So the static variables are declared in the class declaration, this is the class within the class right, so static variables are declared in the class declaration, field variables are also declared in the class declaration, local variables are declared within a subroutine right, so here, so these are int foo is a local variable, while the parameter variables are in the subroutines signature for example bla, int, inside, so this is where the parameter variables are declared right.

(Refer Slide Time: 34:50)


Statements

Statement	Syntax	Description
`let`	`let` *varName* = *expression*; or `let` *varName*[*expression*1] = *expression*2;	An assignment operation (where *varName* is either single-valued or an array). The variable kind may be *static*, *local*, *field*, or *parameter*.
`if`	`if (`*expression*`) {` *statements*1 `}` `else {` *statements*2 `}`	Typical *if* statement with an optional *else* clause. The curly brackets are mandatory even if *statements* is a single statement.
`while`	`while (`*expression*`) {` *statements* `}`	Typical *while* statement. The curly brackets are mandatory even if *statements* is a single statement.
`do`	`do` *function-or-method-call*;	Used to call a function or a method for its effect, ignoring the returned value.
`return`	`Return` *expression*; or `return;`	Used to return a value from a subroutine. The second form must be used by functions and methods that return a void value. Constructors must return the expression `this`.

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

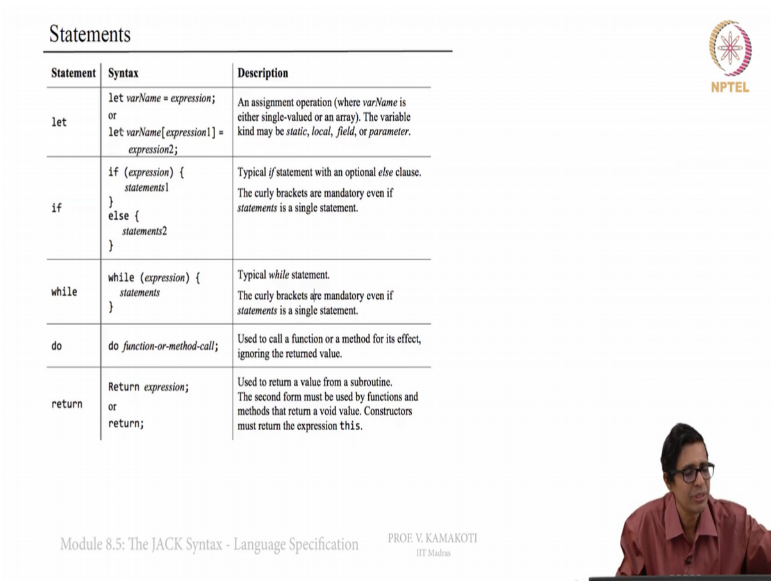
So these are the ways, what are the difference statements that we see? We have seen 5 different types of statements, let is a statement, so let var name equal to expression or let var name of expression 1 equal to expression 2 right, where the var name can be a variable which can be a static, local, field or parameter variable okay and similarly so if statement, if expression curly brackets statements 1 is curly bracket statement 2, where statement can be a single statement or a collection of statements.

So while there is an expression then it can be a statements and do a function or method call and return, returned an expression or return itself right, now so this is a very good introduction to what we term as how we formally define certain values, see I have define a statement using statements right, for example if expression can have if inside this statements 1 and else statements 2, so if statement can have if inside this because, so if is a statement which will contain in turn more statements which could be if or let or while or do.

So inside an if I could have while right, so this is a recursive definition of statements and this is beauty, so this input comes from what we call as theory of computing or automata theory, so there will be a course that many of you will study as automata theory, when you are studying this automata theory course you will be subjected to something like what is a regular expression what will be, so you will be require context free grammar, context and seteo grammar etc right.

When you start studying this things you will be exposed to this recursive definition and one of the foundations steps for understanding, where will you use this recursive definition one question will be asking, why should I study this at all, this is exactly where you will be using such type of recursive definitions where a statement, what is a recursive function? A function calling itself is a recursive function, similarly a statement that calls itself, the definition of a statement, includes statements right and so that is why this is called a recursive definition because a definition of statement includes a definition right, so I could have multiples statements which could be anyone of this inside and if, so that is how we are using it, so now we have define what are statements.

(Refer Slide Time: 37:57)

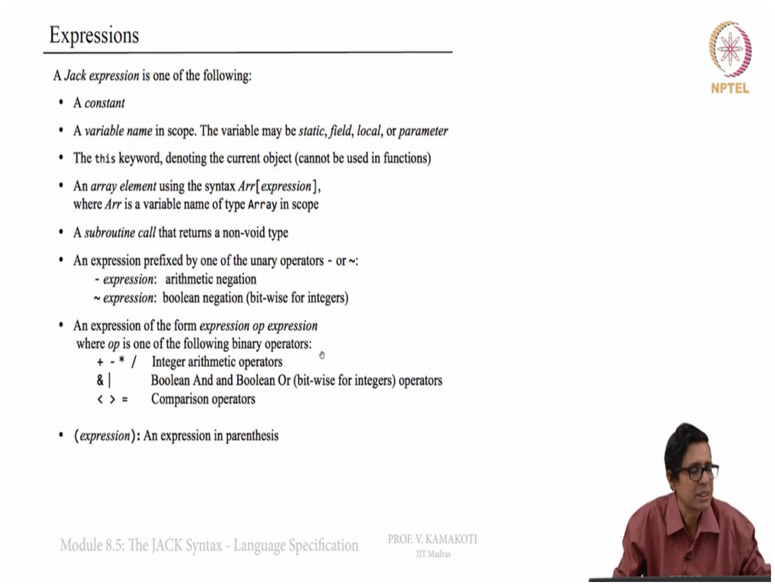


Statements

Statement	Syntax	Description
`let`	`let` *varName* = *expression*; or `let` *varName*[*expression1*] = *expression2*;	An assignment operation (where *varName* is either single-valued or an array). 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* is a single statement.
`while`	`while (`*expression*`) {` *statements* `}`	Typical *while* statement. The curly brackets are mandatory even if *statements* is a single statement.
`do`	`do` *function-or-method-call*;	Used to call a function or a method for its effect, ignoring the returned value.
`return`	`Return` *expression*; or `return;`	Used to return a value from a subroutine. The second form must be used by functions and methods that return a void value. Constructors must return the expression `this`.

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras



Expressions

A *Jack expression* is one of the following:

- A *constant*
- A *variable name* in scope. The variable may be *static*, *field*, *local*, or *parameter*
- The `this` keyword, denoting the current object (cannot be used in functions)
- An *array element* using the syntax *Arr*[*expression*], where *Arr* is a variable name of type `Array` in scope
- A *subroutine call* that returns a non-void type
- An expression prefixed by one of the unary operators - or ~:
  - - *expression*: arithmetic negation
  - ~ *expression*: boolean negation (bit-wise for integers)
- An expression of the form *expression op expression* where *op* is one of the following binary operators:

`+ - * /`	Integer arithmetic operators
`& |`	Boolean And and Boolean Or (bit-wise for integers) operators
`< > =`	Comparison operators

- (*expression*): An expression in parenthesis

Module 8.5: The JACK Syntax - Language Specification

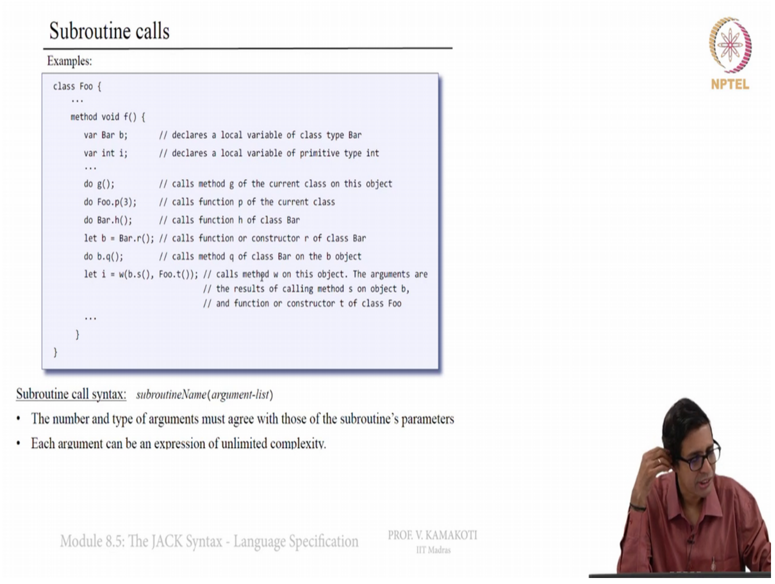
PROF. V. KAMAKOTI
IIT Madras

Now we have to go and define what are expressions? Expressions is 2+3 into 5 is expression, 2+3 x 5 is expression, just A equals to 2, just a 2 itself is an expression, we have seen here, let var name is equal to 5, 5 is a expression, so I constant itself is expression, a variable name within that scope static, field, local or parameter that is also an expression, this keyword keynoting the current object cannot be used in function, but it can be use in methods right, this is a expression, an array element using array expression that ARR is a variable name of types array in scope, it should be in the scope, you know scope of the variable I hope all of you know what is scope of variable is right.

So it is within the definition, so yes, this is an expression, a subroutine call that returns a non-void type because is also an expression, so I could have, suppose this is an integer let I equal to gcd of a, b that is separate int call right, so these are all some basic entities of expression, till now we have not seen the word expression itself, but see here, you are seeing this right, so the expression is an array in which I could have ARR of 2 into J plus K, 2 into J plus K is an expression right, so an expression can have an array inside that main expression, so the recursiveness is starts here.

Now an expression prefixed by unary operator is an expression, that is why we said -13 is an expression, 13 is an constant that is an expression, prefixed it by minus, so -13 becomes an expression or ~expression is an expression, so these are all unary operators, now I look at binary operators like plus, minus, star, star Dave, so an expression op expression, so what you are saying here, expression op expression is another depression, so that op can be plus, minus, star, division, it can be integer or it will be Boolean and Bitwise and Bitwise are, it can be less then, greater than or equal to comparison, an expression inside a parenthesis is an expression.

(Refer Slide Time: 40:36)



Subroutine calls

Examples:

```
class Foo {
    ...
    method void f() {
       var Bar b;       // declares a local variable of class type Bar
       var int i;       // declares a local variable of primitive type int
       ...
       do g();          // calls method g of the current class on this object
       do Foo.p(3);     // calls function p of the current class
       do Bar.h();      // calls function h of class Bar
       let b = Bar.r(); // calls function or constructor r of class Bar
       do b.q();        // calls method q of class Bar on the b object
       let i = w(b.s(), Foo.t()); // calls method w on this object. The arguments are
                                  // the results of calling method s on object b,
                                  // and function or constructor t of class Foo
       ...
     }
}
```

Subroutine call syntax: *subroutineName*(*argument-list*)

- The number and type of arguments must agree with those of the subroutine's parameters
- Each argument can be an expression of unlimited complexity.

Module 8.5: The JACK Syntax - Language Specification

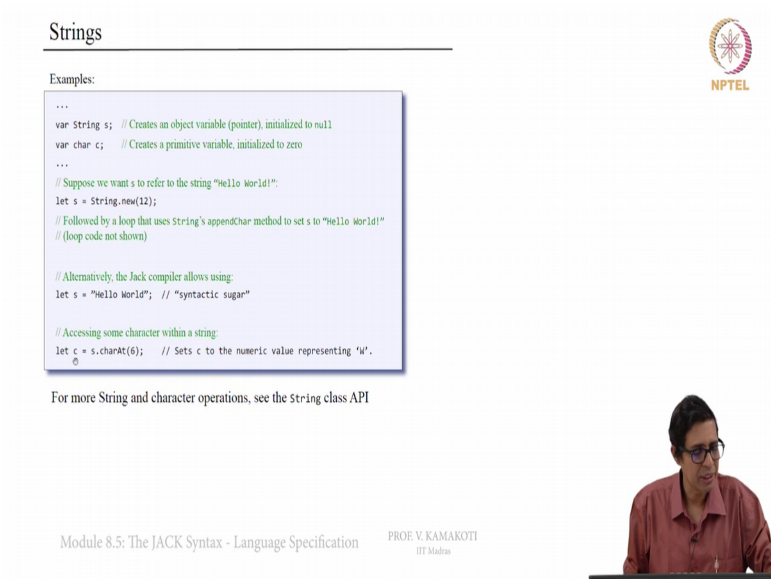
PROF. V. KAMAKOTI
IIT Madras

Right, now lasting that we will be seeing is the subroutine calls, so I have methods, void F, I can say, so that declares a local variable bar, another local variable primitive type int, do G this call methods G of the current class on the subject, but if I say to foo.p this calls a function P of the current class, do bar.h calls function at of some other class bar, B equal to bar.r calls function or constructor R of class bar or do pb.q calls method Q of class bar on the B object, let I equal to W of b.s, foo.t calls method W on this object.

The arguments or the results of calling method S on object B and function or constructor T of class foo okay, so this is how we see this, so when I do a method I say do that matter, but when I call a function I just call it as a function, function will be a expression, method cannot be a expression, so what you need to understand? You need to understand this particular slide very carefully, wherein you need to make it distinction between method and function within the same class, method and function out of the in some other class and var do I use, let var do I use do, do is basically used for methods, while and you know the functions are basically used in expression and you could also do a function of a particular thing.

So you have to be extremely careful in understanding how these things work right, so one of the things will make a distinction between methods and functions in terms of their usage right, a better understanding.

(Refer Slide Time: 43:09)



Strings

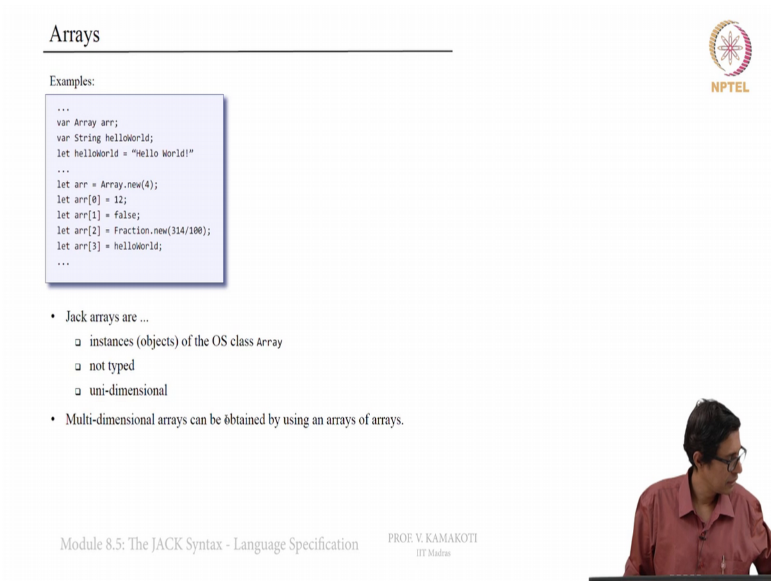
Examples:

```
...
var String s;  // Creates an object variable (pointer), initialized to null
var char c;    // Creates a primitive variable, initialized to zero
...
// Suppose we want s to refer to the string "Hello World!":
let s = String.new(12);
// Followed by a loop that uses String's appendChar method to set s to "Hello World!"
// (loop code not shown)

// Alternatively, the Jack compiler allows using:
let s = "Hello World";  // "syntactic sugar"

// Accessing some character within a string:
let c = s.charAt(6);    // Sets c to the numeric value representing 'W'.
```

For more String and character operations, see the `String` class API

Now string is a class given by the operating system, so a string S, char C, so I can say string.new 12, so this will give you a string of size 12. Okay, this is like malloc for a string, then say let S is equal to hello world, so this is say something like, I can put this entire thing, so both of thing are same, so this will automatically put some 12 characters, now I can access some character within a string, let C equals to s.char at 6, so this, the sixth character 0, 1, 2, 3, 4, 5, 6 would be W right, so it will, C will be assigned the values W, C is a character.

(Refer Slide Time: 44:02)



Arrays

Examples:

```
...
var Array arr;
var String helloWorld;
let helloWorld = "Hello World!"
...
let arr = Array.new(4);
let arr[0] = 12;
let arr[1] = false;
let arr[2] = Fraction.new(314/100);
let arr[3] = helloWorld;
...
```

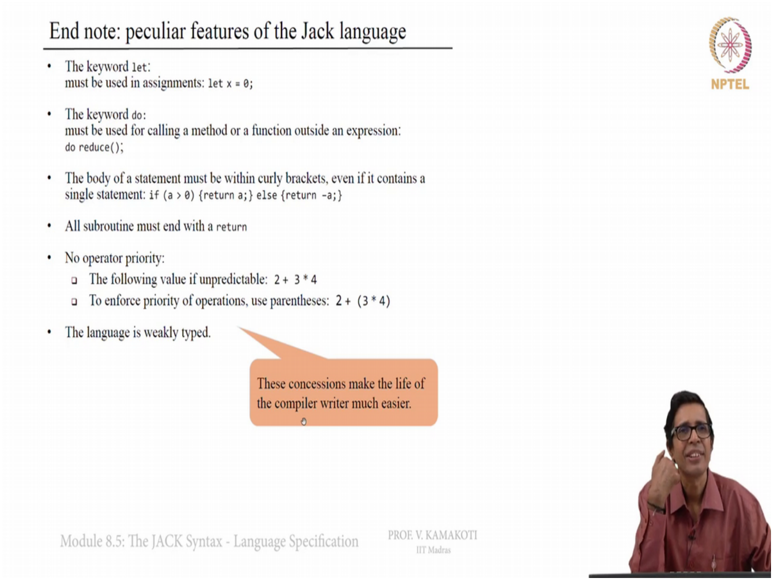
- Jack arrays are ...
  - instances (objects) of the OS class `Array`
  - not typed
  - uni-dimensional
- Multi-dimensional arrays can be obtained by using an arrays of arrays.

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

So array again, this is array.new 4, array 0 is 12, array one is false, I can put anything inside the array and array 2 is equal to fraction.new of 314/100 right and let R 3 is equal to hello world, so I can do all these stuff here. Okay, multidimensional, so the Jack array, so the instances object of the OS class array and is not typed, so I can store different things in different arrays and they are unidimensional, multidimensional arrays can be obtained by using an array of arrays okay.

(Refer Slide Time: 44:55)



End note: peculiar features of the Jack language

- The keyword `let`:
  must be used in assignments: `let x = 0;`
- The keyword `do`:
  must be used for calling a method or a function outside an expression:
  `do reduce();`
- The body of a statement must be within curly brackets, even if it contains a single statement: `if (a > 0) {return a;} else {return -a;}`
- All subroutine must end with a `return`
- No operator priority:
  - The following value if unpredictable: `2 + 3 * 4`
  - To enforce priority of operations, use parentheses: `2 + (3 * 4)`
- The language is weakly typed.

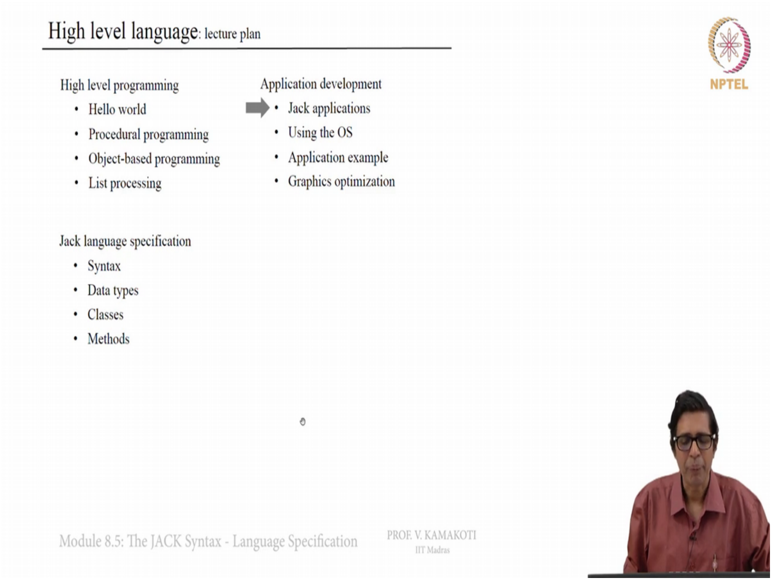
These concessions make the life of the compiler writer much easier.

Module 8.5: The JACK Syntax - Language Specification

PROF. V. KAMAKOTI
IIT Madras

So keyword let must be used in assignments, the keyword do must be used for calling a method or function outside an expression. Okay, the body of a statement must be within curly brackets, even if it contains a single statement, in C if you have a single statement you cannot and all subroutine must end with a return, no operator priority, so the following value if unpredictable, for example 2+3 into 4, the enforced parody of operations use parenthesis right, the language is weakly typed in that sense and these concessions make the life of the compiler writer much easier, so we have to write a compiler finally, so the objectives of this entire exercise and this makes it very good right.

(Refer Slide Time: 45:44)



High level language: lecture plan

High level programming
- Hello world
- Procedural programming
- Object-based programming
- List processing

Application development
- Jack applications
- Using the OS
- Application example
- Graphics optimization

Jack language specification
- Syntax
- Data types
- Classes
- Methods

Module 8.5: The JACK Syntax - Language Specification PROF. V. KAMAKOTI IIT Madras

So this basically concludes the syntax of a Jack language very quickly and now we will look at some small notes on the application development in the next model. Thank you.