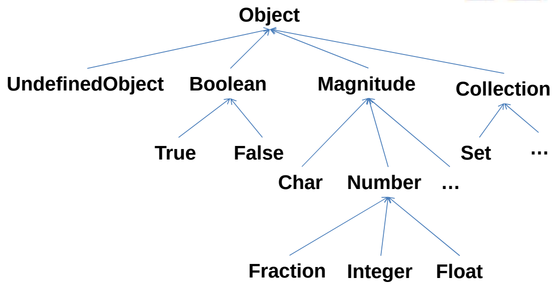
CSE 3302
Programming Languages

Smalltalk

Chengkai Li
Spring 2008

Everything is object. Objects communicate by messages.

Object Hierarchy

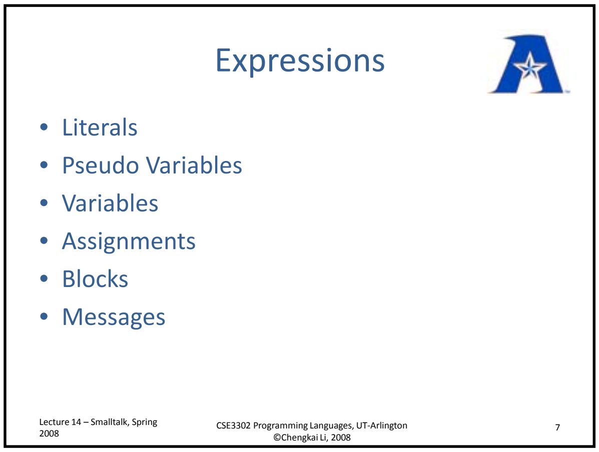


No Data Type.
There is only Class.

Smalltalk Syntax is Simple.

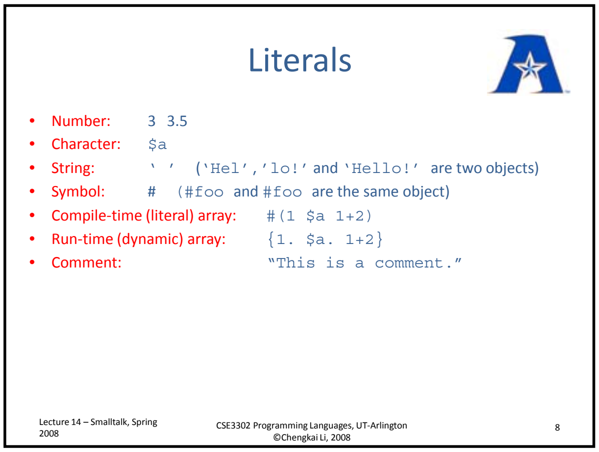
Syntax

- Smalltalk is really “small”
  - Only 6 keywords (pseudo variables)
  - Class, object, variable, method names are self explanatory
  - Only syntax for calling method (messages) and defining method.
    - No syntax for control structure
    - No syntax for creating class



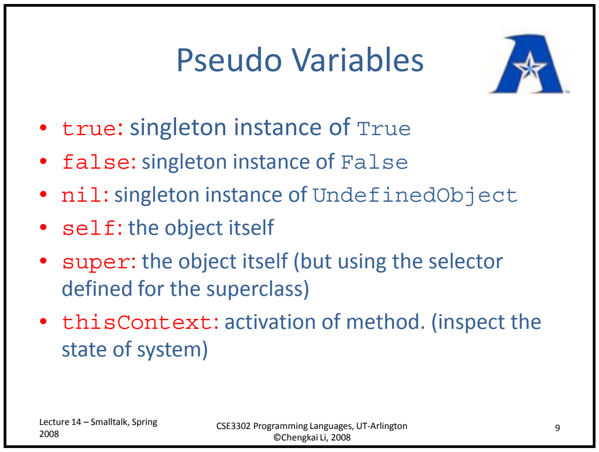
Expressions

- Literals
- Pseudo Variables
- Variables
- Assignments
- Blocks
- Messages



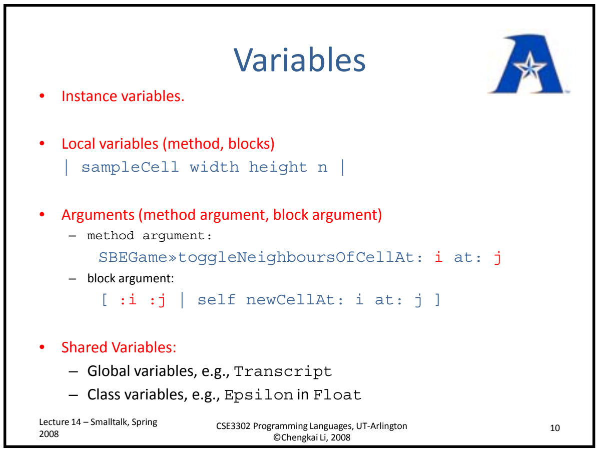
Literals

- Number: 3 3.5
- Character: `$a`
- String: `‘ ’` (`‘Hel’,’lo!’` and `‘Hello!’` are two objects)
- Symbol: # (`#foo` and `#foo` are the same object)
- Compile-time (literal) array: `#(1 $a 1+2)`
- Run-time (dynamic) array: `{1. $a. 1+2}`
- Comment: `“This is a comment.”`



Pseudo Variables

- `true`: singleton instance of `True`
- `false`: singleton instance of `False`
- `nil`: singleton instance of `UndefinedObject`
- `self`: the object itself
- `super`: the object itself (but using the selector defined for the superclass)
- `thisContext`: activation of method. (inspect the state of system)



Variables

- Instance variables.

- Local variables (method, blocks)

```
| sampleCell width height n |
```

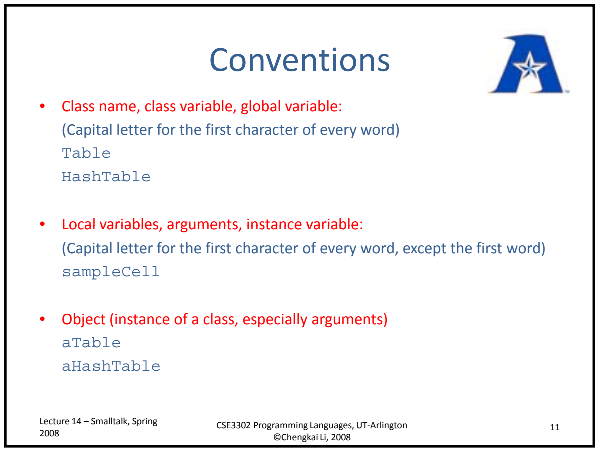
- Arguments (method argument, block argument)
  - method argument:

```
SBEGame»toggleNeighboursOfCellAt: i at: j
```

  - block argument:

```
[ :i :j | self newCellAt: i at: j ]
```

- Shared Variables:
  - Global variables, e.g., `Transcript`
  - Class variables, e.g., `Epsilon` in `Float`

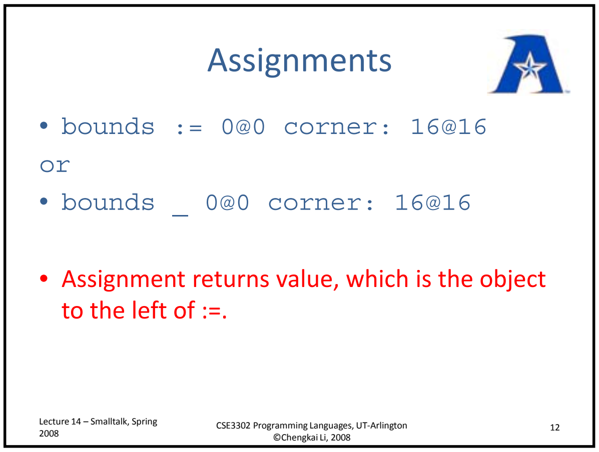


Conventions

- Class name, class variable, global variable:
  (Capital letter for the first character of every word)
  `Table`
  `HashTable`

- Local variables, arguments, instance variable:
  (Capital letter for the first character of every word, except the first word)
  `sampleCell`

- Object (instance of a class, especially arguments)
  `aTable`
  `aHashTable`



Assignments

- `bounds := 0@0 corner: 16@16`

or

- `bounds _ 0@0 corner: 16@16`

- Assignment returns value, which is the object to the left of :=.

Defining a Method

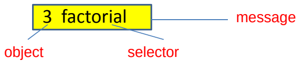
selector (method name)
  | local variable |
  statement (expression).   (. is used to end a statement)
  statement(expression).
  ^ return-value   (^ returns value from a method)

Example of a method

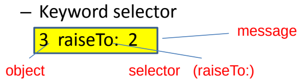
- FloatArray>>= aFloatArray

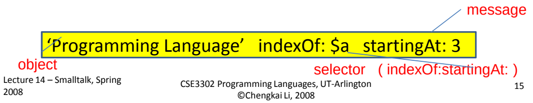
Methods and Messages

- Method Name: Selector
- Method Invocation: Message
  – Unary selector



  – Keyword selector




message
'Programming Language' indexOf: $a startingAt: 3
object
selector (indexOf:startingAt:)

Keyword Selector: more readable

- `table insert: anItem at: anIndex`

```
table insert: 3 at: 5
```

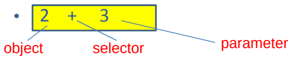
vs.

- `table.insert(anItem, anIndex)`

```
table.insert(3,5)
```

Binary selector

-



- 2 + 3 + 4 ?

- aTable / 3 (what it means depends on the class)
- 1+2*3 (* does not have higher precedence than -, because they are messages that can be sent to any object. No mathematical meaning is assumed.)

- Examples:
  – Integer>>#+
  – Complex>>#+
  – Fraction>>#+

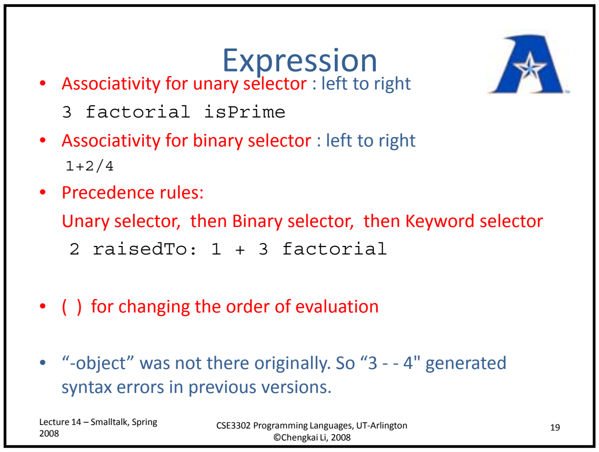
```
3/5
(1/3) + (1/2)
```

Binary selector

- + - * /
- = (equality) ~= >= <= > <
- == (identity, the two objects are the same object), ~~
- & | Boolean
- , (string concatenation)

'Hel','lo' = 'Hello'
'Hel','lo' == 'Hello'
#Hello == #Hello

- Assignment := is not a method



Expression

- Associativity for unary selector : left to right

```
3 factorial isPrime
```

- Associativity for binary selector : left to right

```
1+2/4
```

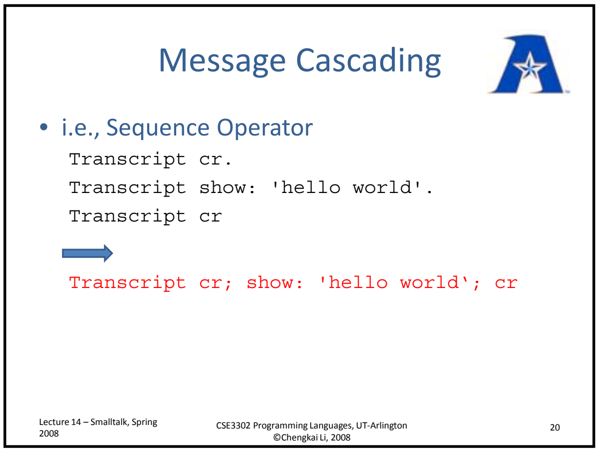
- Precedence rules:

  Unary selector, then Binary selector, then Keyword selector

```
 2 raisedTo: 1 + 3 factorial
```

- ( ) for changing the order of evaluation

- “-object” was not there originally. So “3 - - 4" generated syntax errors in previous versions.



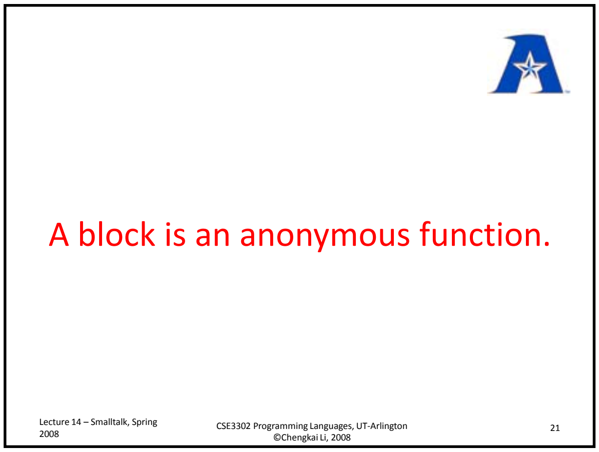
Message Cascading

- i.e., Sequence Operator

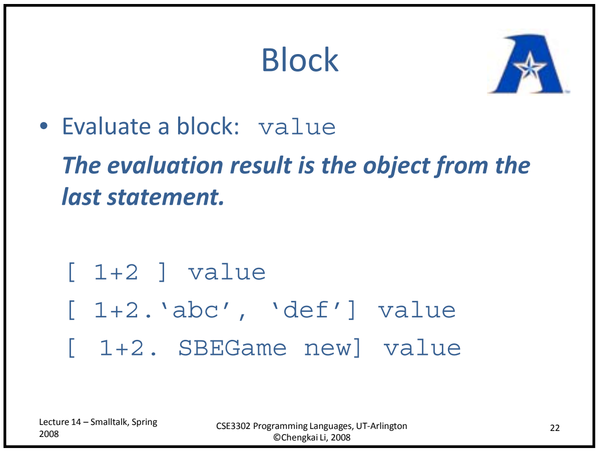
```
Transcript cr.
Transcript show: 'hello world'.
Transcript cr
```

⇒

```
Transcript cr; show: 'hello world'; cr
```



A block is an anonymous function.

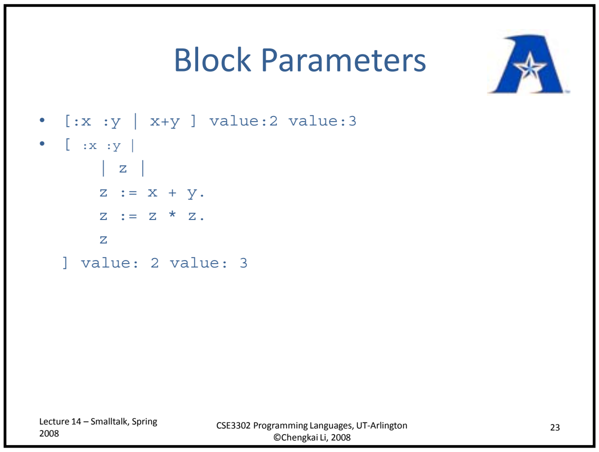


Block

- Evaluate a block: `value`

  The evaluation result is the object from the last statement.

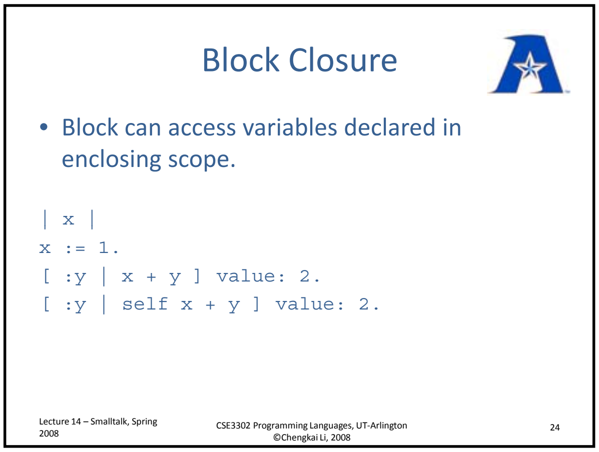
```
[ 1+2 ] value
[ 1+2.‘abc’, ‘def’] value
[  1+2. SBEGame new] value
```



Block Parameters

- `[:x :y | x+y ] value:2 value:3`

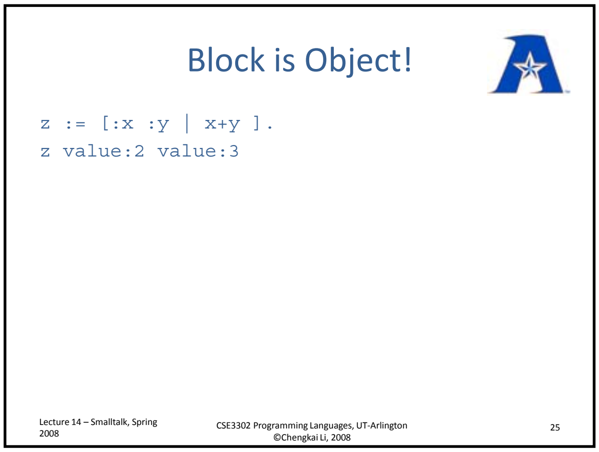
-
  ```
  [ :x :y |
      | z |
      z := x + y.
      z := z * z.
      z
  ] value: 2 value: 3
  ```



Block Closure

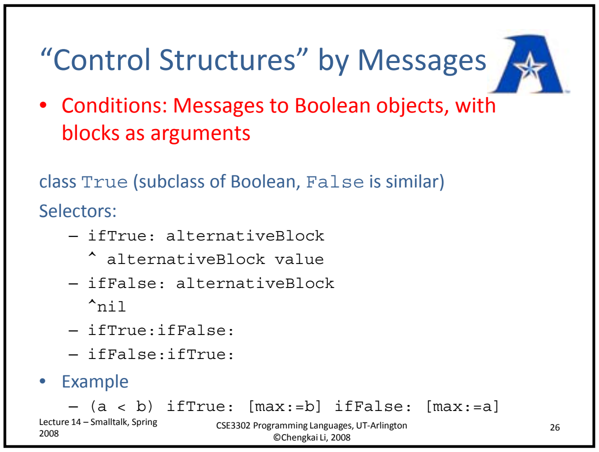
- Block can access variables declared in enclosing scope.

```
| x |
x := 1.
[ :y | x + y ] value: 2.
[ :y | self x + y ] value: 2.
```



Block is Object!

```
z := [:x :y | x+y ].
z value:2 value:3
```



“Control Structures” by Messages

- Conditions: Messages to Boolean objects, with blocks as arguments

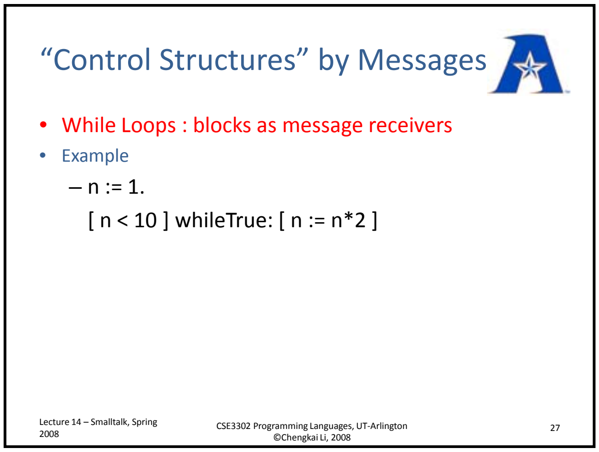
class `True` (subclass of Boolean, `False` is similar)

Selectors:

```
– ifTrue: alternativeBlock
    ^ alternativeBlock value
– ifFalse: alternativeBlock
    ^nil
– ifTrue:ifFalse:
– ifFalse:ifTrue:
```

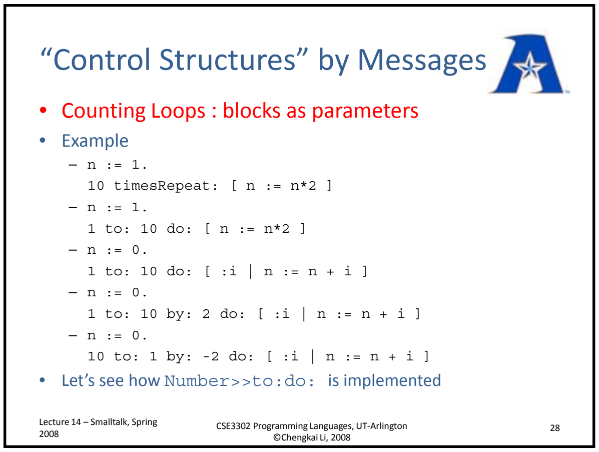
- Example

```
– (a < b) ifTrue: [max:=b] ifFalse: [max:=a]
```



“Control Structures” by Messages

- While Loops : blocks as message receivers
- Example
  - n := 1.
    [n < 10] whileTrue: [n := n*2]



“Control Structures” by Messages

- Counting Loops : blocks as parameters
- Example

```
– n := 1.
  10 timesRepeat: [ n := n*2 ]
– n := 1.
  1 to: 10 do: [ n := n*2 ]
– n := 0.
  1 to: 10 do: [ :i | n := n + i ]
– n := 0.
  1 to: 10 by: 2 do: [ :i | n := n + i ]
– n := 0.
  10 to: 1 by: -2 do: [ :i | n := n + i ]
```

- Let’s see how `Number>>to:do:` is implemented