Administrative Issues

- HW3 (due at March 18th)
- Essay (due at May 1st)
  - ABET requirement
  - you must get a passing score (37.5), otherwise you will receive Incomplete (I) for this course
- MP2 (due at April 3rd)
  - released at March 20th

CSE 3302 Programming Languages

Smalltalk (cont.)

Chengkai Li
Spring 2008

About method arguments

A>>f: x
  x := …

Assignment to argument is not allowed.

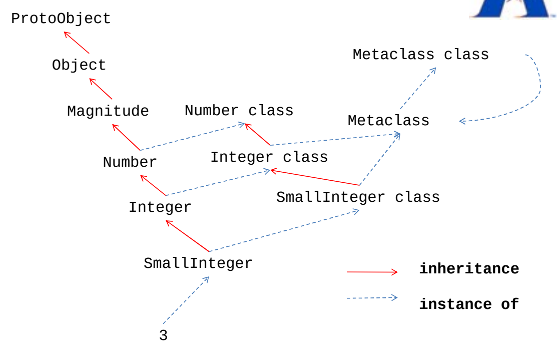
Why?

- If you want to change x, send a message to x
- If you just want to refer to another object, why don't use a different time?

OO in Smalltalk

- Everything is an Object (Classes are objects too)
- Every object is an instance of a class (a class is an instance of its metaclass)
- Every class has a superclass
- Everything happens by messages.
- Method lookup follows the inheritance chain.

Object(Instance), Class, Inheritance

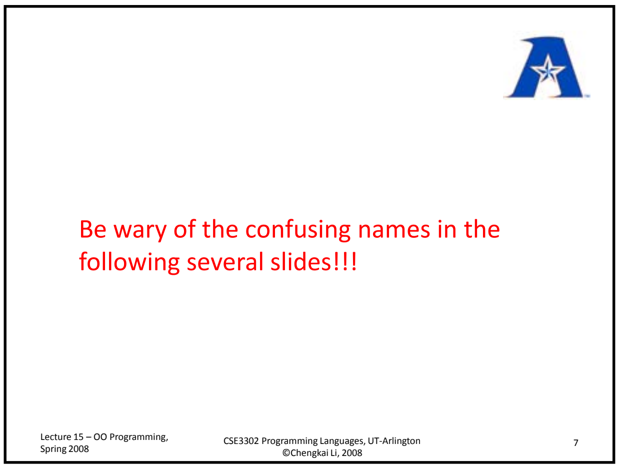


Examples

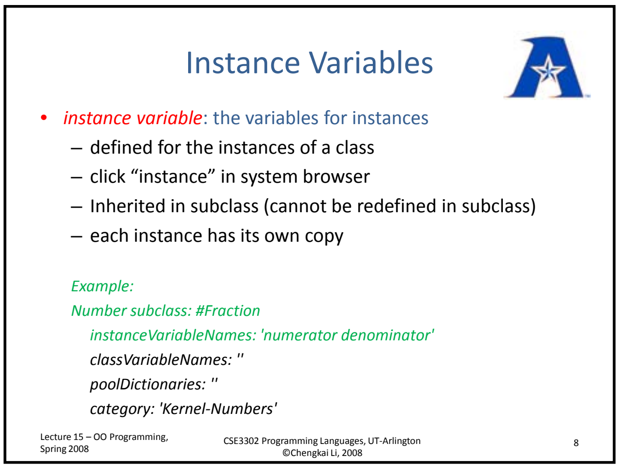
"print it" and "inspect it"

- `3`
- `3 class`
- `3 class class`
- `3 class class class`
- `3 class class class class`
- `3 class class class class class`

- `SmallInteger superclass`



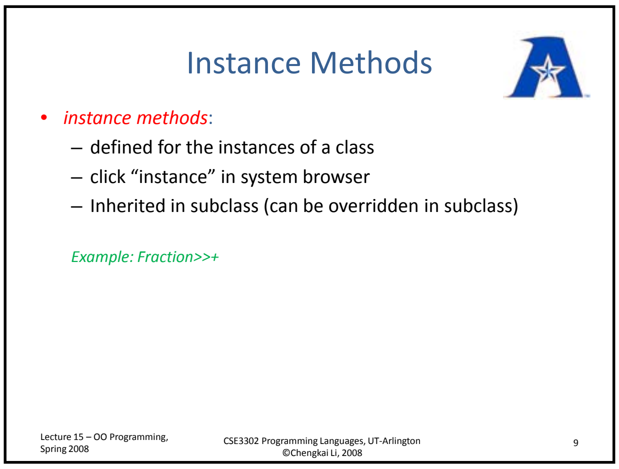
Be wary of the confusing names in the following several slides!!!



Instance Variables

- *instance variable*: the variables for instances
  - defined for the instances of a class
  - click “instance” in system browser
  - Inherited in subclass (cannot be redefined in subclass)
  - each instance has its own copy

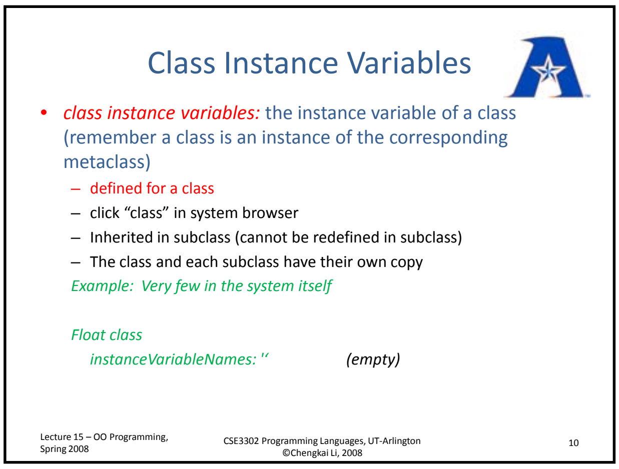
Example:
Number subclass: #Fraction
    instanceVariableNames: 'numerator denominator'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Kernel-Numbers'



Instance Methods

- *instance methods*:
  - defined for the instances of a class
  - click “instance” in system browser
  - Inherited in subclass (can be overridden in subclass)

Example: Fraction>>+

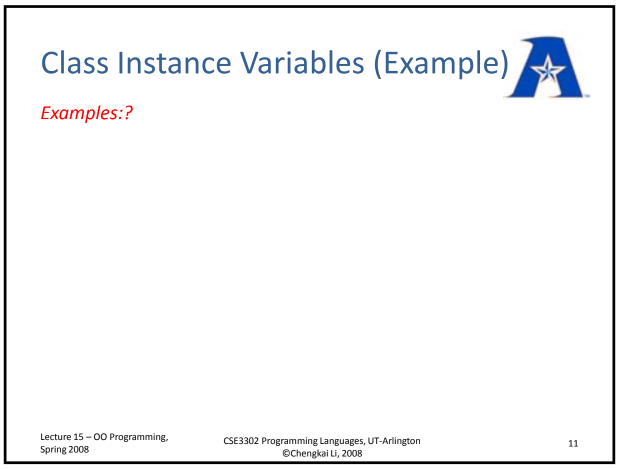


Class Instance Variables

- *class instance variables:* the instance variable of a class (remember a class is an instance of the corresponding metaclass)
  - defined for a class
  - click “class” in system browser
  - Inherited in subclass (cannot be redefined in subclass)
  - The class and each subclass have their own copy

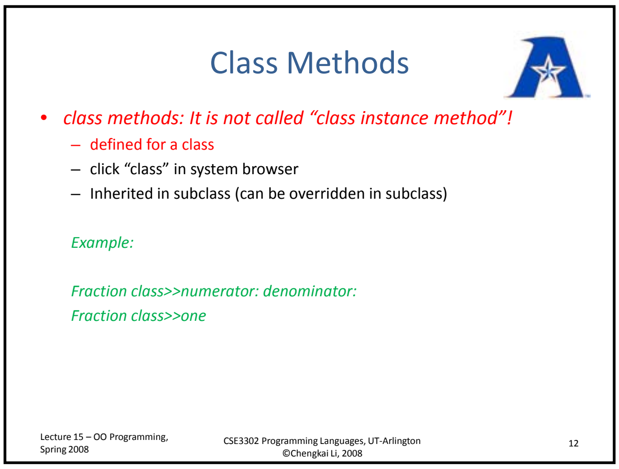
Example: Very few in the system itself

Float class
    instanceVariableNames: ‘‘   *(empty)*



Class Instance Variables (Example)

Examples:?



Class Methods

- *class methods: It is not called “class instance method”!*
  - defined for a class
  - click “class” in system browser
  - Inherited in subclass (can be overridden in subclass)

Example:

Fraction class>>numerator: denominator:
Fraction class>>one

Class Variables

- Class Variable: it doesn't pair with "class method", which pairs with "class instance variable"
  - defined for both a class and its instances
  - click "instance" in system browser
  - Inherited in subclass (cannot be redefined in subclass)
  - There is only one copy! (shared by the class itself, all the subclasses, and all the instances of the class/subclasses)

Example:

```
Number variableWordSubclass: #Float
    instanceVariableNames: ''
    classVariableNames: 'E Epsilon Halfpi Infinity Ln10 Ln2 MaxVal MaxValLn MinValLogBase2
    NaN NegativeInfinity NegativeZero Pi RadiansPerDegree Sqrt2 ThreePi Twopi'
    poolDictionaries: ''
    category: 'Kernel-Numbers'
```

Access

- All data members are private
  - Instance variables: only directly accessible to the instance itself (inside instance methods)
    - Different instances have different copies of the instance variables
  - Class instance variables: only directly accessible to the class itself (inside class methods)
    - Each subclass has its own copies of the class instance variables
  - Class variables: directly accessible to all the instances of the class and the class itself. (inside class methods and instance methods)
    - The same copy shared by all instances and subclasses
- All methods are public (any object can send any message to any other object)
  - The private instance variables are accessible to outside through the methods.

For Accessing Private Data: Setter and Getter

Example:
class `Complex`
instance variable `real, imaginary`

- getter
```
Complex>>real
    ^real
```
- setter
```
Complex>>real: aNumber
    real _ aNumber
```
(remember _ means :=)

Class Methods for Constructing New Instances

- Example:

class `Complex`
class methods:

```
– Complex class>>real: aNumber1 imaginary:
  aNumber2
     | newComplex |
     newComplex _ super new.
     newComplex
          real: aNumber1;
          imaginary: aNumber2.
     ^ newComplex
– Complex class>>new
       ^ self real: 0 imaginary: 0
```

Class Methods for Accessing Class Variables

- Example:

class `Float`
class variable: `Pi E Epsilon` …
class method:

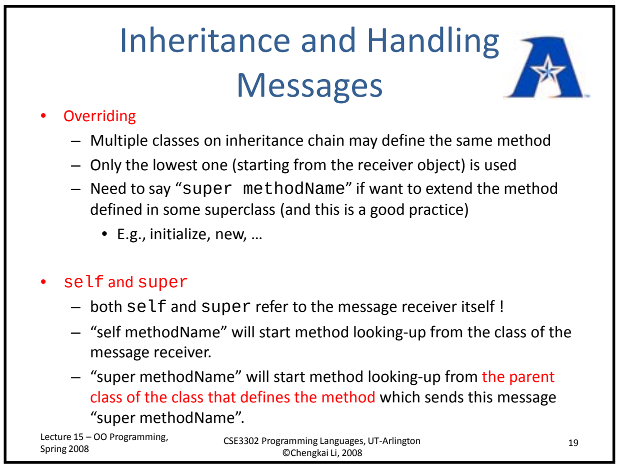
```
– Float class>>pi
     ^Pi
```

- Compare:
  - method `real` is defined in `Complex`, so an instance of `Complex`(e.g., 3+2i) can receive message `real`
  - method `pi` is defined in `Float class` (instead of `Float`), so an instance of `Float class` (i.e., `Float`) can receive message `pi`

Inheritance and Handling Messages

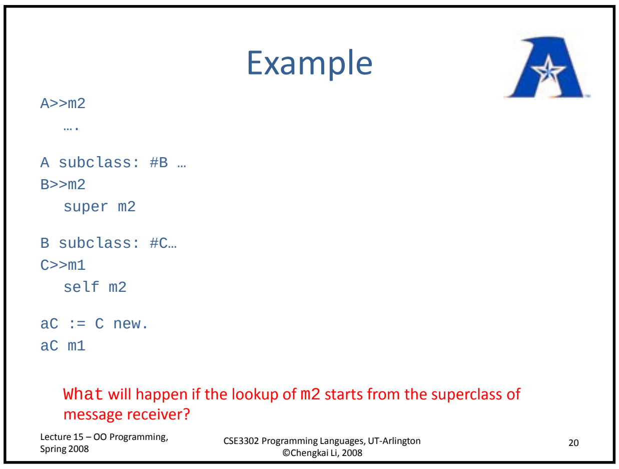
- Inheritance:
  - Smalltalk allows only single inheritance
- Method Lookup along the inheritance chains

  When an object receives a message:
  - If the class of the object has the method, use it;
  - Otherwise check the superclass, and the superclass of the superclass, and so on.
- Return value of a method
  - Message receiver if no explicit return (i.e., no ^)



Inheritance and Handling Messages

- Overriding
  - Multiple classes on inheritance chain may define the same method
  - Only the lowest one (starting from the receiver object) is used
  - Need to say "`super methodName`" if want to extend the method defined in some superclass (and this is a good practice)
    - E.g., initialize, new, …
- `self` and `super`
  - both `self` and `super` refer to the message receiver itself !
  - "self methodName" will start method looking-up from the class of the message receiver.
  - "super methodName" will start method looking-up from the parent class of the class that defines the method which sends this message "super methodName".



Example

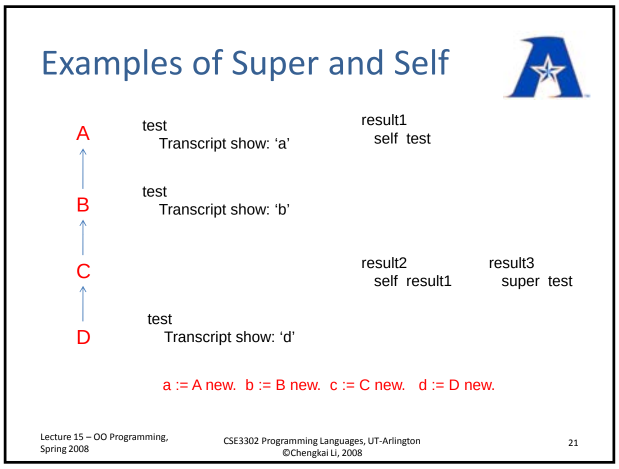
```
A>>m2
  ….

A subclass: #B …
B>>m2
  super m2

B subclass: #C…
C>>m1
  self m2

aC := C new.
aC m1
```

`What` will happen if the lookup of `m2` starts from the superclass of message receiver?



Examples of Super and Self

$$\begin{tikzcd} A \\ B \arrow[u] \\ C \arrow[u] \\ D \arrow[u] \end{tikzcd}$$

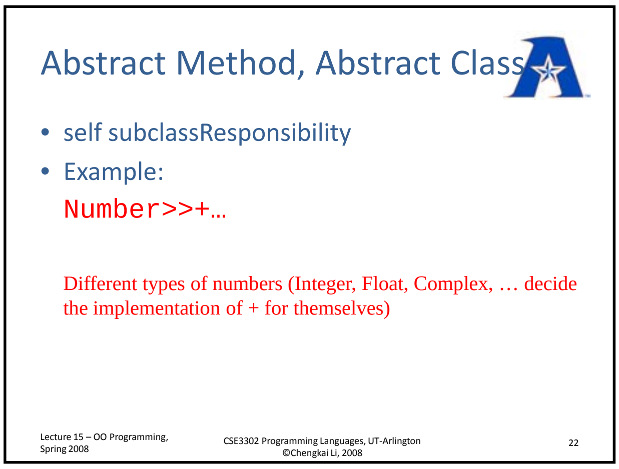
```
A   test                              result1
      Transcript show: 'a'              self  test

B   test
      Transcript show: 'b'

C                                     result2              result3
                                        self  result1        super  test

D   test
      Transcript show: 'd'
```

a := A new. b := B new. c := C new.   d := D new.



Abstract Method, Abstract Class

- self subclassResponsibility
- Example:

  `Number>>+…`

  Different types of numbers (Integer, Float, Complex, … decide the implementation of + for themselves)