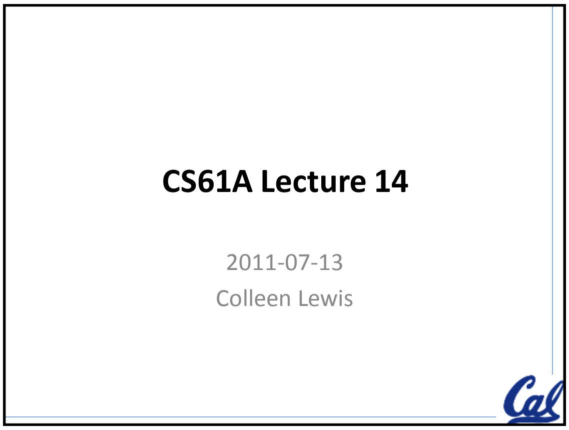
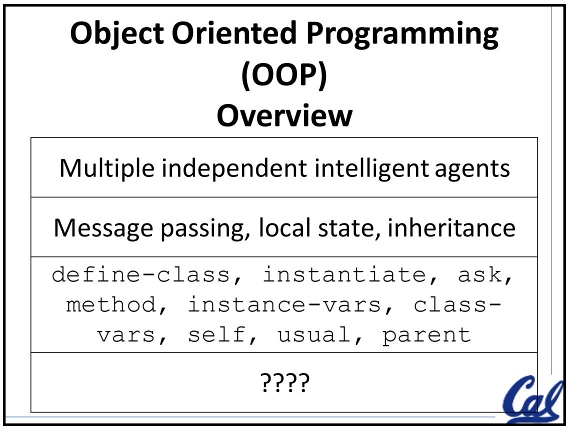


CS61A Lecture 14

2011-07-13

Colleen Lewis



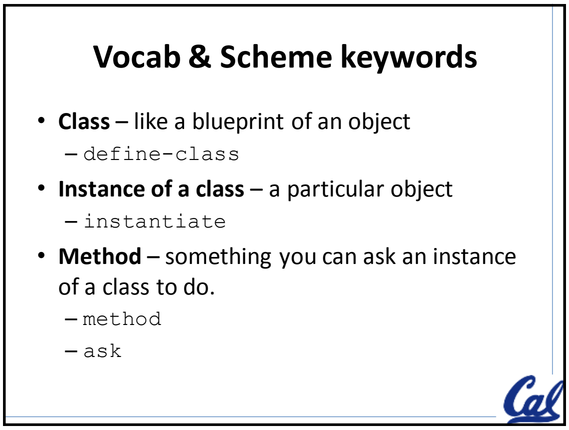
Object Oriented Programming (OOP) Overview

Multiple independent intelligent agents

Message passing, local state, inheritance

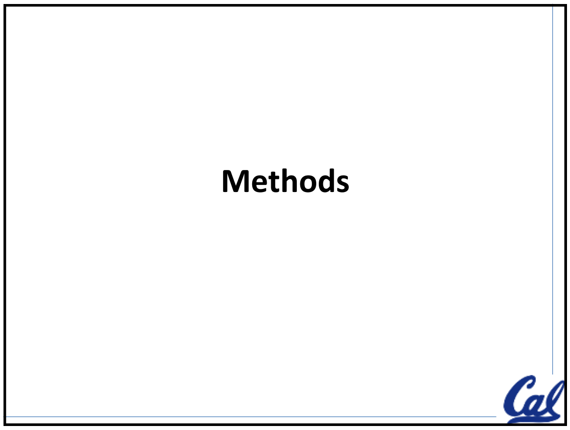
`define-class, instantiate, ask, method, instance-vars, class-vars, self, usual, parent`

????

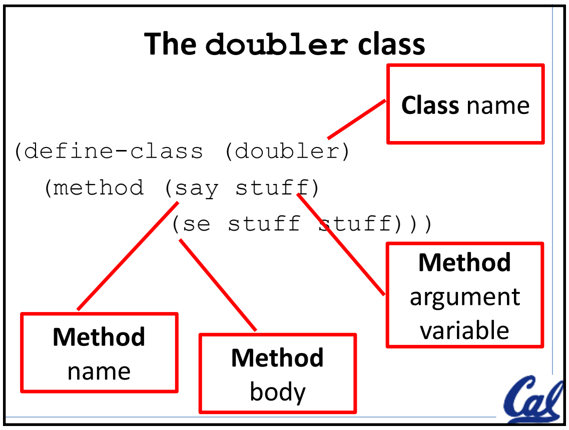


Vocab & Scheme keywords

- **Class** – like a blueprint of an object
  - `define-class`
- **Instance of a class** – a particular object
  - `instantiate`
- **Method** – something you can ask an instance of a class to do.
  - `method`
  - `ask`

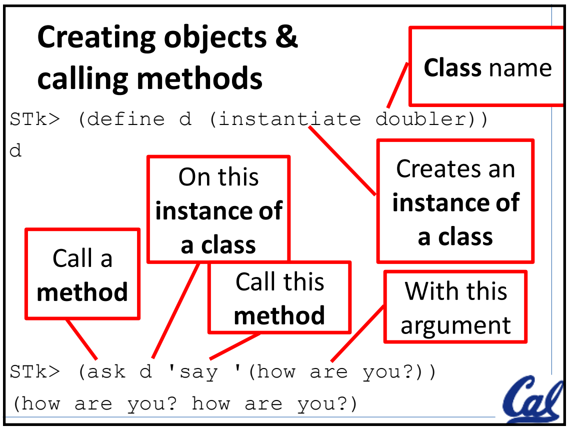


Methods



The `doubler` class

```
(define-class (doubler)
  (method (say stuff)
    (se stuff stuff)))
```



Creating objects & calling methods

```
STk> (define d (instantiate doubler))
d
```

```
STk> (ask d 'say '(how are you?))
(how are you? how are you?)
```

Modify the `doubler` class

```
(define-class (doubler)
  (method (say stuff)
          (se stuff stuff)))

STk> (ask d 'add 2 3)
10
STk> (ask d 'add 1 1)
4
```

'add is a: A) function B) method C) class D)message

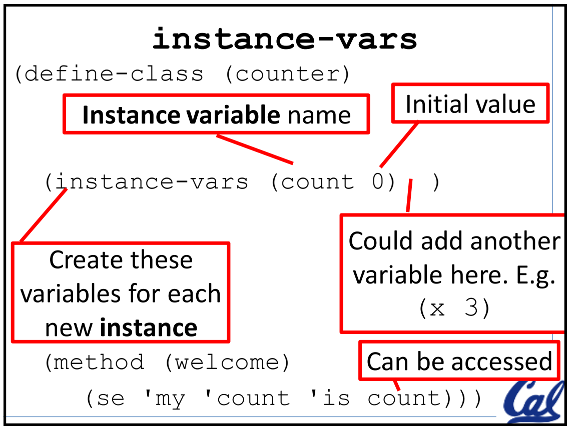
instance variables

instance-vars

Vocab

- **Instance variables** – variables local to an instance of a class
  - instance-vars

instance-vars



```
(define-class (counter)

  (instance-vars (count 0)  )

  (method (welcome)
     (se 'my 'count 'is count)))
```

Instance variable name

Initial value

Create these variables for each new **instance**

Could add another variable here. E.g. (x 3)

Can be accessed

When do you use quotes?

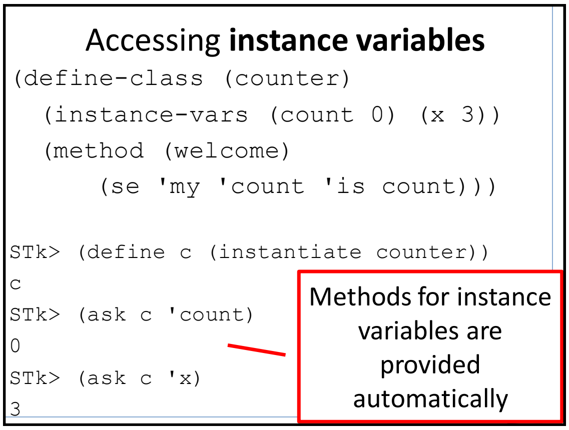
```
(define-class (counter)
  (instance-vars (count 0))
  (method (welcome)
        (se 'my 'count 'is count)))
STk> (define c (instantiate ?counter))
c
STk> (ask c ?welcome)
(my count is 0)
```

Which needs a quote?

A) Class name B) method name C) both D) neither

If you change the class, ALWAYS recreate your objects

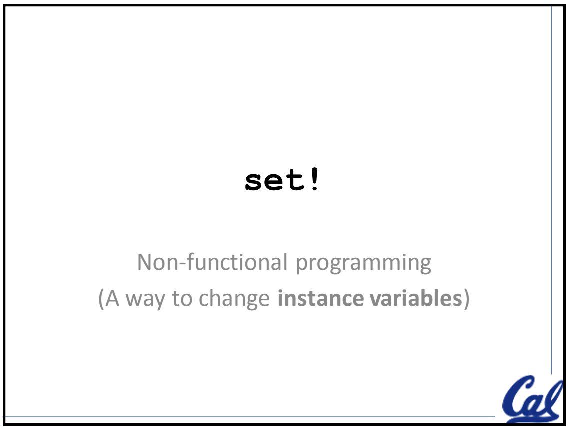
```
STk> (load "lect14.scm")
okay
STk> (define c (instantiate counter))
c
STk> (ask c 'welcome)
(my count is 0)
```



Accessing **instance variables**

```
(define-class (counter)
  (instance-vars (count 0) (x 3))
  (method (welcome)
       (se 'my 'count 'is count)))

STk> (define c (instantiate counter))
c
STk> (ask c 'count)
0
STk> (ask c 'x)
3
```

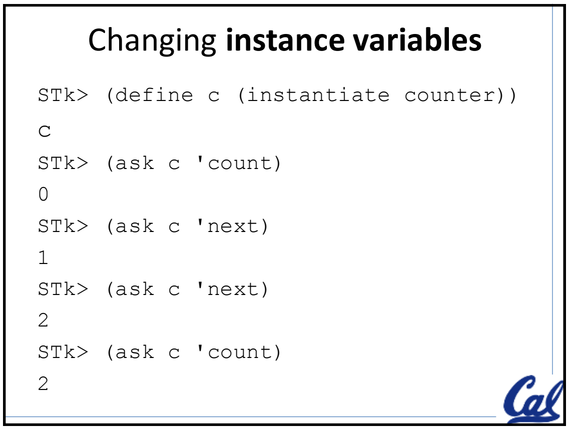
Methods for instance variables are provided automatically



set!

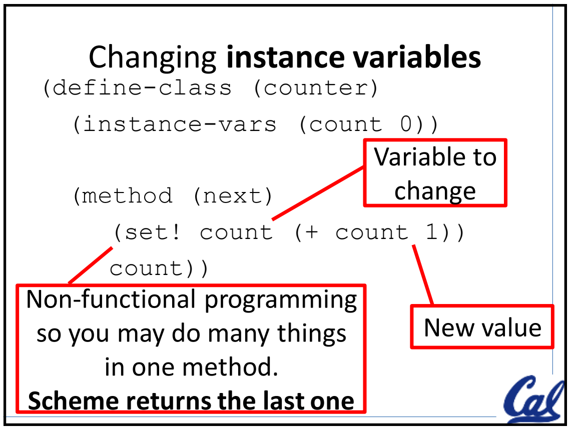
Non-functional programming

(A way to change **instance variables**)



Changing **instance variables**

```
STk> (define c (instantiate counter))
c
STk> (ask c 'count)
0
STk> (ask c 'next)
1
STk> (ask c 'next)
2
STk> (ask c 'count)
2
```



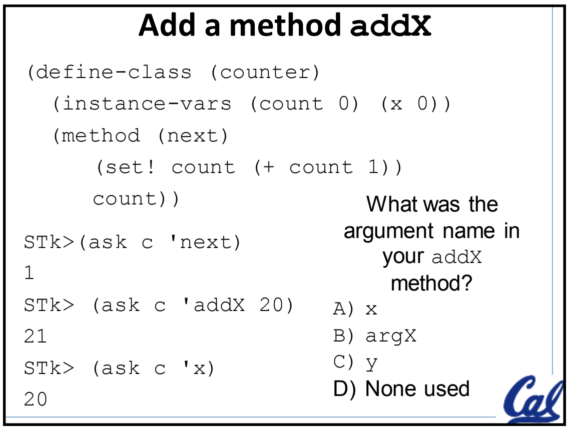
Changing **instance variables**

```
(define-class (counter)
  (instance-vars (count 0))

  (method (next)
    (set! count (+ count 1))
    count))
```

Variable to change

New value

Non-functional programming so you may do many things in one method.
Scheme returns the last one

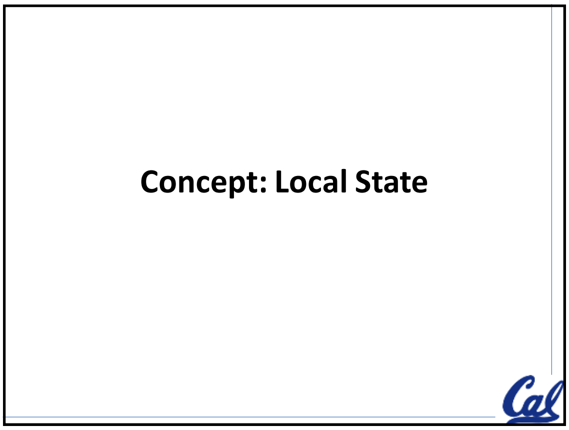


Add a method addX

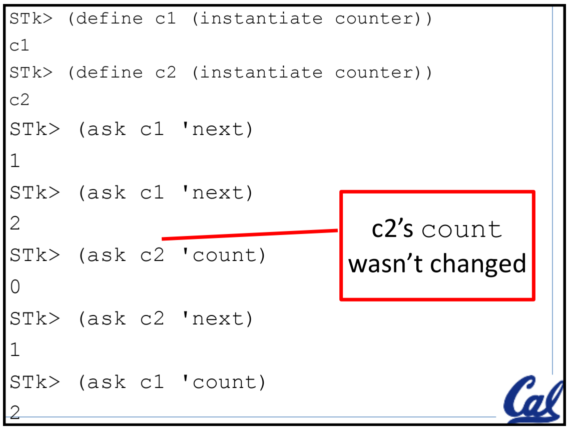
```
(define-class (counter)
  (instance-vars (count 0) (x 0))
  (method (next)
     (set! count (+ count 1))
     count))

STk>(ask c 'next)
1
STk> (ask c 'addX 20)
21
STk> (ask c 'x)
20
```

What was the argument name in your addX method?

A) x
B) argX
C) y
D) None used

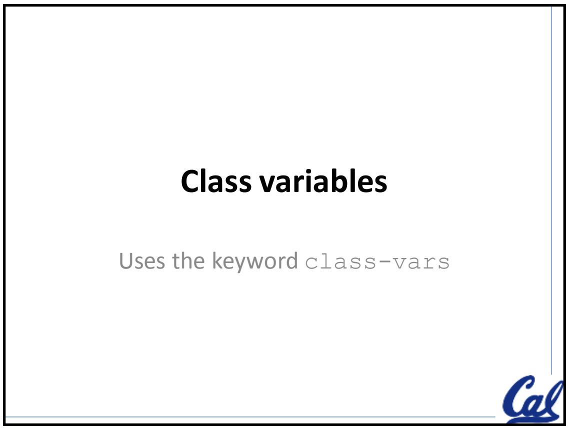


Concept: Local State



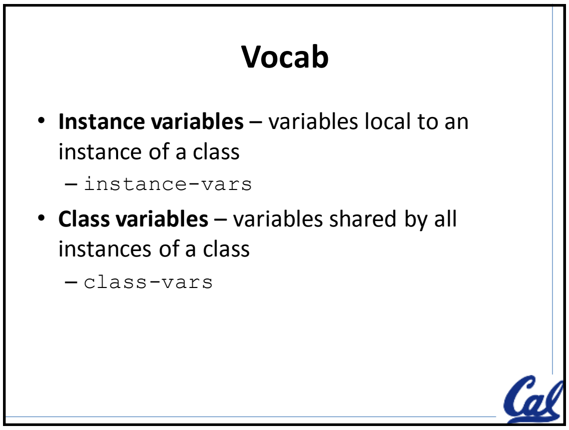
```
STk> (define c1 (instantiate counter))
c1
STk> (define c2 (instantiate counter))
c2
STk> (ask c1 'next)
1
STk> (ask c1 'next)
2
STk> (ask c2 'count)
0
STk> (ask c2 'next)
1
STk> (ask c1 'count)
2
```

c2's count wasn't changed



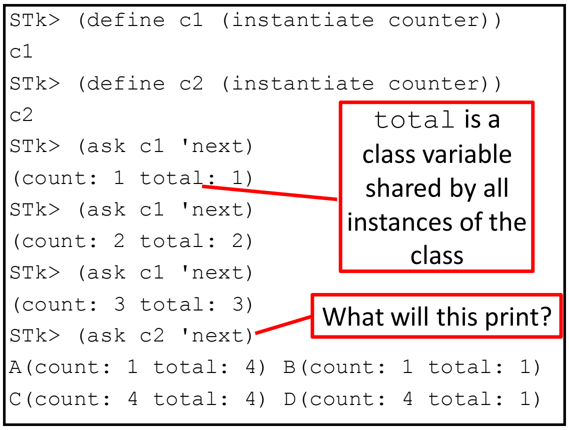
Class variables

Uses the keyword `class-vars`



Vocab

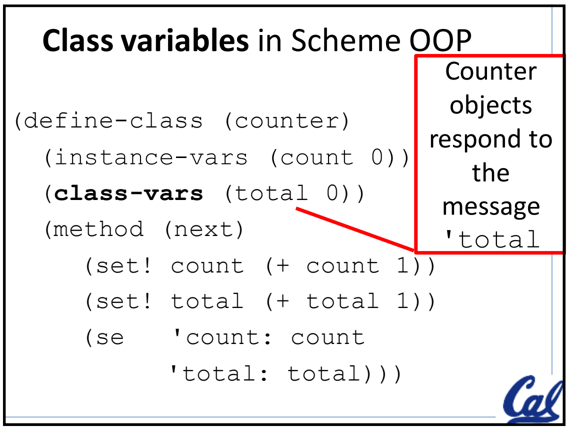
- **Instance variables** – variables local to an instance of a class
  - `instance-vars`
- **Class variables** – variables shared by all instances of a class
  - `class-vars`



```
STk> (define c1 (instantiate counter))
c1
STk> (define c2 (instantiate counter))
c2
STk> (ask c1 'next)
(count: 1 total: 1)
STk> (ask c1 'next)
(count: 2 total: 2)
STk> (ask c1 'next)
(count: 3 total: 3)
STk> (ask c2 'next)
A(count: 1 total: 4) B(count: 1 total: 1)
C(count: 4 total: 4) D(count: 4 total: 1)
```

`total` is a class variable shared by all instances of the class

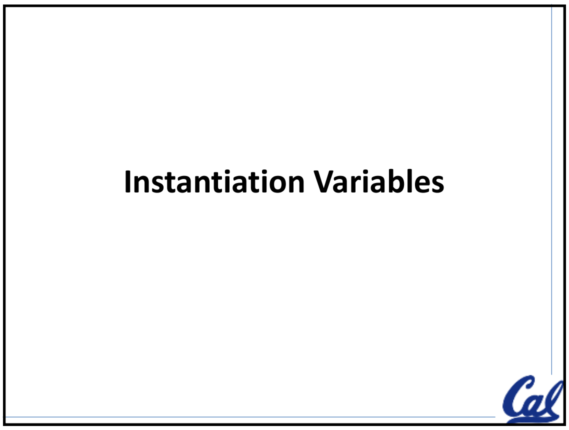
What will this print?



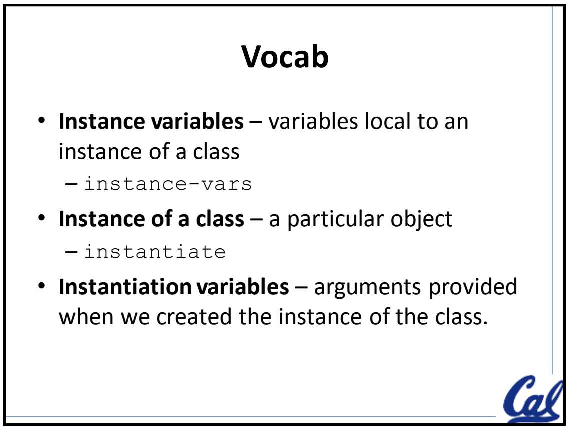
Class variables in Scheme OOP

```
(define-class (counter)
  (instance-vars (count 0))
  (class-vars (total 0))
  (method (next)
    (set! count (+ count 1))
    (set! total (+ total 1))
    (se   'count: count
          'total: total)))
```

Counter objects respond to the message `'total`

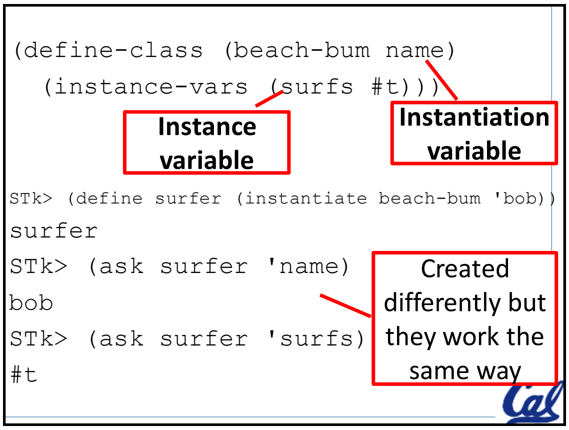


Instantiation Variables



Vocab

- **Instance variables** – variables local to an instance of a class
  - `instance-vars`
- **Instance of a class** – a particular object
  - `instantiate`
- **Instantiation variables** – arguments provided when we created the instance of the class.



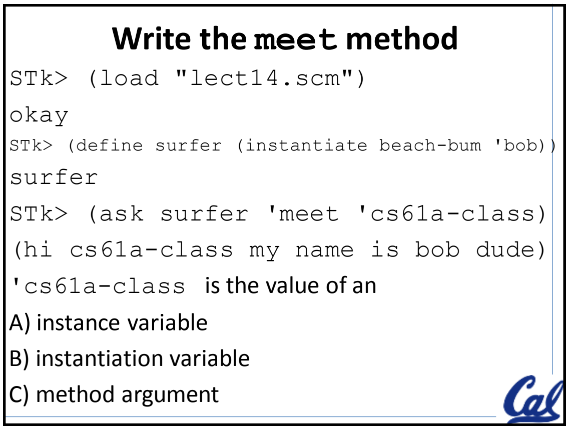
```
(define-class (beach-bum name)
  (instance-vars (surfs #t)))
```

Instance variable

Instantiation variable

```
STk> (define surfer (instantiate beach-bum 'bob))
surfer
STk> (ask surfer 'name)
bob
STk> (ask surfer 'surfs)
#t
```

Created differently but they work the same way



Write the `meet` method

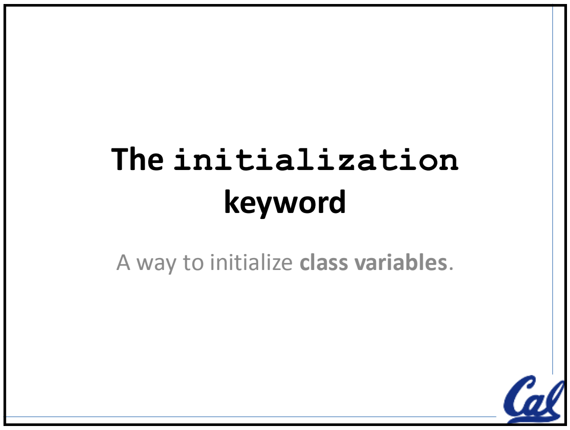
```
STk> (load "lect14.scm")
okay
STk> (define surfer (instantiate beach-bum 'bob))
surfer
STk> (ask surfer 'meet 'cs61a-class)
(hi cs61a-class my name is bob dude)
```

`'cs61a-class` is the value of an

A) instance variable

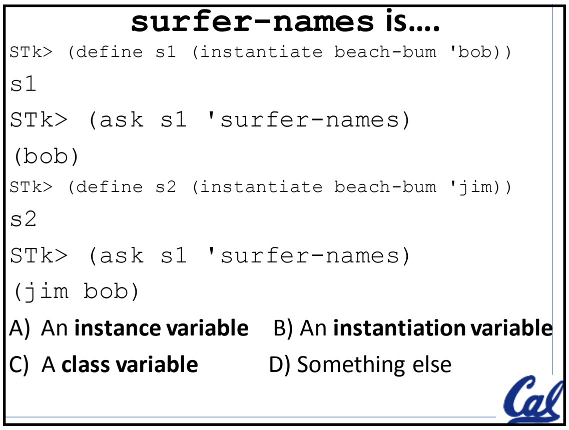
B) instantiation variable

C) method argument



The `initialization` keyword

A way to initialize **class variables**.

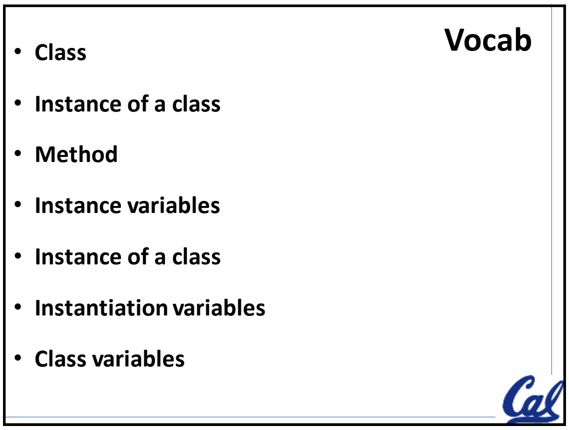


`surfer-names` is….

```
STk> (define s1 (instantiate beach-bum 'bob))
s1
STk> (ask s1 'surfer-names)
(bob)
STk> (define s2 (instantiate beach-bum 'jim))
s2
STk> (ask s1 'surfer-names)
(jim bob)
```

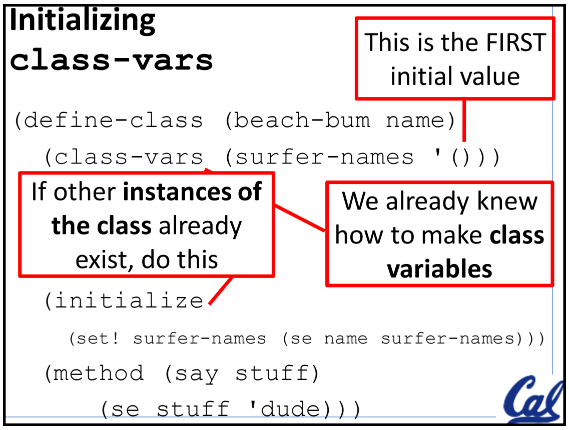
A) An **instance variable** B) An **instantiation variable**

C) A **class variable** D) Something else



Vocab

- **Class**
- **Instance of a class**
- **Method**
- **Instance variables**
- **Instance of a class**
- **Instantiation variables**
- **Class variables**



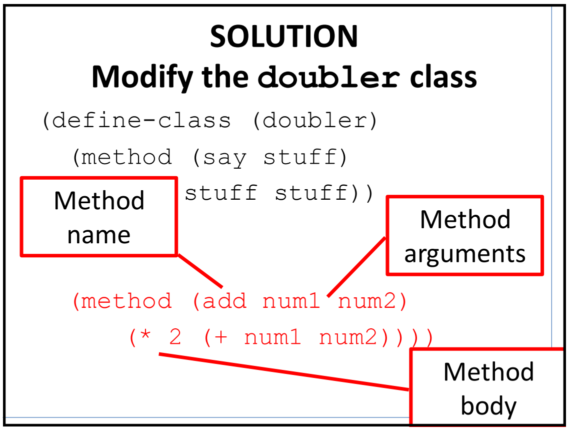
Initializing `class-vars`

```
(define-class (beach-bum name)
  (class-vars (surfer-names '()))
  (initialize
    (set! surfer-names (se name surfer-names)))
  (method (say stuff)
     (se stuff 'dude)))
```

This is the FIRST initial value

If other instances of **the class** already exist, do this

We already knew how to make **class variables**



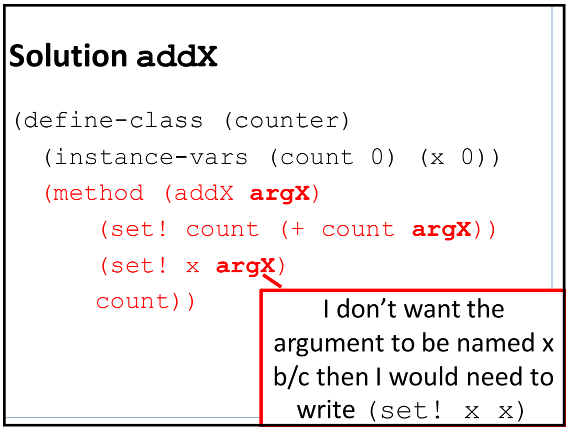
SOLUTION Modify the `doubler` class

```
(define-class (doubler)
  (method (say stuff)
     stuff stuff))
  (method (add num1 num2)
     (* 2 (+ num1 num2))))
```

Method name

Method arguments

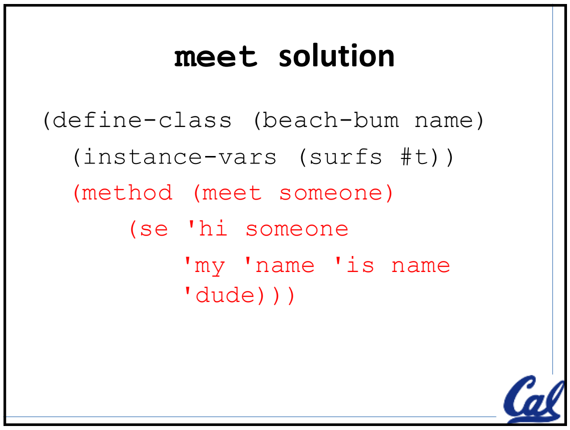
Method body



Solution `addX`

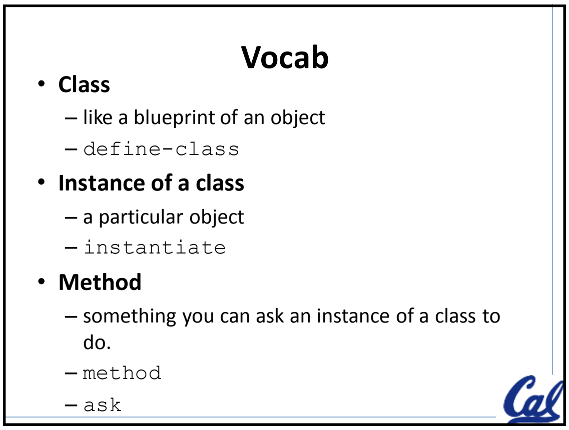
```
(define-class (counter)
  (instance-vars (count 0) (x 0))
  (method (addX argX)
     (set! count (+ count argX))
     (set! x argX)
     count))
```

I don't want the argument to be named x b/c then I would need to write `(set! x x)`



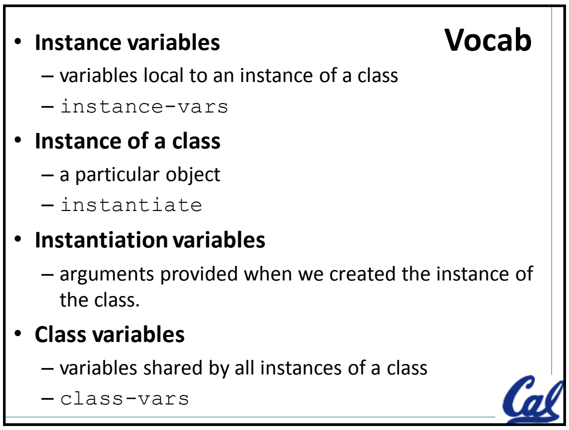
`meet` solution

```
(define-class (beach-bum name)
  (instance-vars (surfs #t))
  (method (meet someone)
     (se 'hi someone
        'my 'name 'is name
        'dude)))
```



Vocab

- **Class**
  - like a blueprint of an object
  - `define-class`
- **Instance of a class**
  - a particular object
  - `instantiate`
- **Method**
  - something you can ask an instance of a class to do.
  - `method`
  - `ask`



Vocab

- **Instance variables**
  - variables local to an instance of a class
  - `instance-vars`
- **Instance of a class**
  - a particular object
  - `instantiate`
- **Instantiation variables**
  - arguments provided when we created the instance of the class.
- **Class variables**
  - variables shared by all instances of a class
  - `class-vars`