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

----

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

```scheme
(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

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

Create these variables for each new instance

- Instance variable name
- Initial value

Could add another variable here. E.g. `x 3`

Can be accessed

Vocab

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

When do you use quotes?

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

STk> (load "lect14.scm")
okay
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

```scheme
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

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

Concept: Local State

What was the argument name in your addX method?

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

Uses the keyword class-vars

Vocab

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

Counter objects respond to the message 'total

Instantiation Variables

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

What will this print?

Counter objects respond to the message 'total
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)))
```

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

```

class

Vocab

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

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
```

Created differently but they work the same way
Initializing class-vars

(define-class (beach-bum name)
  (class-vars (surfer-names '())))

If other instances of the class already exist, do this

(initialize
  (set! surfer-names (se name surfer-names)))

We already knew how to make class variables

(define-class (beach-bum name)
  (class-vars (surfer-names '())))

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)

Solution

SOLUTION
Modify the doubler class

(define-class (doubler)
  (method (say stuff)
    (se stuff 'dude)))

Method name

(method (say stuff)
  (se stuff 'dude)))

Method arguments

(method (add num1 num2)
  (* 2 (+ num1 num2))))

Method body

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

• 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