OBJECT-ORIENTED PROGRAMMING 7

COMPUTER SCIENCE 61AS

Basics of OOP

1. What are the three types of variables in an OOP system?

   Solution: Instantiation, Instance, and Class. Youll see examples of all of them soon!

2. Let's say we're trying to write a `Dog` class. Give examples of how we can use each one of the variables in this class.

   Solution: Instantiation: name
   Instance: owner, age, favorite-food
   Class: cat-lovers, number-of-dogs

3. Why would I ever want to make a class have a parent? Why would I want a class to have a child?

   Solution: Sometimes, the parent might have some useful methods. If the child is a subtype or specialization of the parent, then we reduce duplicate code by using a parent!

4. What's the proper syntax to call an object's method?

   Solution: `(ask dog 'owner)`

   More generally,
   `(ask <object> <method-name> <args>)`

Practice with Classes and Methods

The following problems deal with the definition of the dog class below:

```
(define-class (dog name)
    (instance-vars (owner 'no-one))
    (method (bark) '(woof woof!)) )
```

1. Fill in the blanks.

```
> (define fido (instantiate dog 'fido))
> (ask fido 'bark)
  (woof woof!)

> (ask fido 'name)
_____________
```

Solution: `fido`

```
> (ask fido _____________)
  no-one
```

Solution: `'owner`

2. Currently, there are a bunch of stray dogs on the streets. We want to find these dogs some owners! Write a new method `follow-home` that changes the owner of the dog. This method takes in one argument, the person the dog follows. Because the dog is so happy with his new owner, he should bark at the end of this method.

```
(define-class (dog name)
    (instance-vars (owner 'no-one))
    (method (bark) '(woof woof!))
    (method (follow-home person)
        ;;; YOUR CODE HERE!
```

Solution:

```
(set! owner person)
(ask self 'bark)
```

It's very important that we use `(ask self 'bark)}` at the end rather than `'(woof woof!)`. If we had done the latter, everytime we change the `bark` method we would

have to remember to update the `follow-home` method. In general, hard-coding and duplicating code is bad, especially when we can so easily avoid it.

3. Write a `person` class. A person should have a name as an instantiation variable, and a list of pets as an instance variable. It should have the sole method `adopt-pet`. Don't worry about making `adopt-pet` change the dog's owner instance variable–dogs pick their own owners.

Solution:

```
(define-class (person name)
    (instance-vars (list-of-pets '()))
    (method (adopt-pet pet)
        (set! list-of-pets (cons pet list-of-pets))) )
```

4. When a dog follows a person home, we want the `adopt-pet` method to be called. Otherwise, the dog will think he has an owner, but the person wont consider him a pet :(Modify the `follow-home` method in the dog class so that it calls the person's `adopt-pet` method.

Solution:

```
(method (follow-home person)
    (set! owner person)
    (ask person 'adopt-pet self)
    (ask self 'bark) )
```

5. Not all people are "dog people". The dogs do their best to steer clear of the cat-lovers. In fact, they keep a master list of these cat-lovers. All dogs share this list!

   Modify the `follow-home` method of the dog class so that a dog will not make a cat-lover his owner. When a dog realizes he followed home a cat-lover, he should add that person to the list `cat-lovers`. The dog should still bark at the end of the method so he can scare the cat-lover. `cat-lover?` should be an instantation variable in the person class. You can make any other changes to the dog/person class to make this work.

Solution:

```
(define-class (person name cat-lover?)  ;;; CHANGED
    (instance-vars (list-of-pets '()))
    (method (adopt-pet pet)
        (set! list-of-pets (cons pet list-of-pets))) )
```

```
(define-class (dog name)
    (instance-vars (owner 'no-one))
    (class-vars (cat-lovers '())) ;;; ADDED
    (method (bark) '(woof woof!))

    (method (follow-home person)   ;;; CHANGED THIS METHOD
        (if (ask person 'cat-lover?)
            (set! cat-lovers (cons person cat-lovers))
            (begin
                (set! owner person)
                (ask person 'adopt-pet self)))
        (ask self 'bark) ))
```

Inheritance

All dogs are not created equal. Write a `german-shepherd` class that is a child class of `dog`. The only difference between a `german-shepherd` and a generic `dog` is that the `german-shepherd` has a more formidable bark. Avoid writing duplicate code as much as possible!

Solution:

```
(define-class (german-shepherd name)
    (parent (dog name))
    (method (bark) '(I am an oak tree)) )
```

What will Scheme Print?

```
> (define Beethoven (instantiate german-shepherd 'Beethoven))
> (ask Beethoven 'name)
```

Solution: `Beethoven`

```
> (ask Beethoven 'owner)
```

Solution: `no-one`

```
> (ask Beethoven 'bark)
```

Solution: `(I am an oak tree`

```
> (ask Beethoven 'cat-lovers)
```

Solution: `()`

```
> (ask german-shepherd 'cat-lovers)
```

Solution: `ERROR`

```
> (ask dog 'cat-lovers)
```

Solution: `()`

```
> (define Andrew (instantiate person 'Andrew #t))
> (ask Beethoven 'follow-home Andrew)
```

Solution: `(I am an oak tree)`

```
> (ask Beethoven 'owner)
```

Solution: `no-one`

```
> (ask Andrew 'list-of-pets)
```

Solution: `()`