Sheeple: Dynamic Object-Orientation for CL

Josh Marchán

22 October 2009

Setting up the strawman. . .

Setting up the strawman. . .

- Write general blueprints, create objects based on them to hold some (limited) changeable state.

Setting up the strawman. . .

- Write general blueprints, create objects based on them to hold some (limited) changeable state.
- Otherwise, objects are fairly static

Setting up the strawman. . .

- Write general blueprints, create objects based on them to hold some (limited) changeable state.
- Otherwise, objects are fairly static
- Nothing object-oriented about classes.

  Saying class-orientation is “object-oriented” is a damn filthy lie.

Setting up the strawman. . .

- Write general blueprints, create objects based on them to hold some (limited) changeable state.
- Otherwise, objects are fairly static
- Nothing object-oriented about classes.
  Saying class-orientation is “object-oriented” is a damn filthy lie.
- "Static constructs considered harmful" (for dynamic languages)

Hope?

Hope?

- Python - arbitrary attributes

Hope?

- Python - arbitrary attributes
- Ruby - open classes

Hope?

- Python - arbitrary attributes
- Ruby - open classes
- Smalltalk - Everything is an instance of a class, including classes.

Hope?

- Python - arbitrary attributes
- Ruby - open classes
- Smalltalk - Everything is an instance of a class, including classes.
- CLOS - class redefinition, change-class, EQL specializers, MOP

Example: Developing a game

Goal: Write a game
You're mapping to conceptual objects. Class-orientation seems like a perfect fit.

Example: Developing a game

Goal: Write a game
You're mapping to conceptual objects. Class-orientation seems like a perfect fit.
Or is it?

Example: The class hierarchy

Let's write a class hierarchy!

Example: The class hierarchy

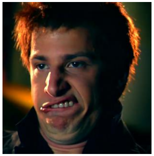
Let's write a class hierarchy!

```
Class GameObject (no instances)
 \_ Class Weapon (no instances)
     \_ Class Sword (no instances)
         \_ Class FireSword (maybe an instance?)
             \_ ??? (What now?)
```

Example: The class hierarchy

Let's write a class hierarchy!

```
Class GameObject (no instances)
 \_ Class Weapon (no instances)
     \_ Class Sword (no instances)
         \_ Class FireSword (maybe an instance?)
             \_ ??? (What now?)
```



Suggestions?

Example: Working toward a solution

Some solutions:

- Class explosion
- Programmatic class creation
- Data-driven programming

Example: Solutions

Class explosion:

Example: Solutions

Class explosion:

- Still class-oriented.
- Lots of manual class writing
- Might make code. . . difficult

Example: Solutions

Class explosion:

- Still class-oriented.
- Lots of manual class writing
- Might make code. . . difficult

  Do you really want to write FireIceMagicMissileGoldenGodForeverSwordOfDoom?

Example: Solutions

Programmatic class creation:

Example: Solutions

Programmatic class creation:

- May not actually be possible in your language
- Programmatic classes are sketchy, at best. Using them may be difficult and arcane.
- Reusability possibly out the window.

Example: Solutions

Data-driven programming

Example: Solutions

Data-driven programming

- Eject! Eject!

  Class-orientation can't actually do what you want, so you need to use something else
- Blessing

  You've escaped the hell that is class-based programming

Example: Screw this.

From a paper on Self

"How hardcore do you want to be? How many lifetimes do you want to waste?"

What is Object-Oriented Programming?

Object-oriented programming: A different paradigm for putting together your program.

Overview of objects

Overview of objects

- Objects are like other objects, and share each others' behavior.

Overview of objects

- Objects are like other objects, and share each others' behavior.
- No separate entity to confuse the process.

Overview of objects

- Objects are like other objects, and share each others' behavior.
- No separate entity to confuse the process.
- Delegate behavior **and** data.

Structuring an application with objects

Structuring an application with objects

- Create an object

Structuring an application with objects

- Create an object
- Copy or delegate to that object

Structuring an application with objects

- Create an object
- Copy or delegate to that object
- New object can act as a prototype for other objects

Structuring an application with objects

- Create an object
- Copy or delegate to that object
- New object can act as a prototype for other objects
- Don't Panic!
  Lots of other "object-oriented" design principles you learned still apply, in a good way.

Example: Game objects revisited

Back to the game. Now with objects:

Example: Game objects revisited

Back to the game. Now with objects:

```
Object
 \_ Object, add some Weapon-like attributes
     \_ Weapon, add some Sword-like attributes
         \_ Sword, add some Fire-like attributes
             \_ ???, add some ???-like attributes
```

Example: Game objects revisited

Back to the game. Now with objects:

```
Object
 \_ Object, add some Weapon-like attributes
     \_ Weapon, add some Sword-like attributes
         \_ Sword, add some Fire-like attributes
             \_ ???, add some ???-like attributes
```

Awesome!

Example: That's totally not different at all! D:<

What's the difference?

- Make objects different by making them different
- Incremental development on a live system
- Objects defined by what they do, and what attributes they have, not by an abstract blueprint

Example (cont'd)

What does this mean?

- Create a new object that implements the new behavior. Delegate what you want.

Example (cont'd)

What does this mean?

- Create a new object that implements the new behavior. Delegate what you want.
- Programmatic creation of objects is easy and natural (not like there's any other way to do it!)

Example (cont'd)

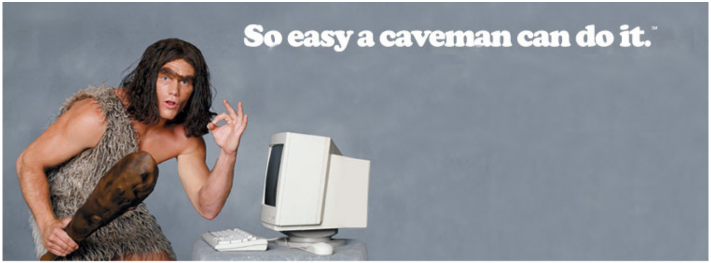
What does this mean?

- Create a new object that implements the new behavior. Delegate what you want.
- Programmatic creation of objects is easy and natural (not like there's any other way to do it!)
- Heck, make programmatic object creation part of the design tool

Example (cont'd)

What does this mean?

- Create a new object that implements the new behavior. Delegate what you want.
- Programmatic creation of objects is easy and natural (not like there's any other way to do it!)
- Heck, make programmatic object creation part of the design tool



Object-orientation in the wild

- Prototype and Properties "patterns" (for classes)
- JavaScript
- MUDs
  - LambdaMOO
  - LPC
  - DGD

CLOS

- Common Lisp Object System. Integrates with Common Lisp
- Multiple inheritance
- Multiple dispatch
- Method combination
- Metaobject Protocol

More dynamic than other class-based systems

More dynamic than other class-based systems

- Redefinition of classes with instance updates (Very complicated, though. See CLHS 4.3.6)

More dynamic than other class-based systems

- Redefinition of classes with instance updates
  (Very complicated, though. See CLHS 4.3.6)
- Adding/removing generic functions and methods.
  - Does not require redefinition of a class.
  - Generic functions and classes conceptually separate.
  - Very dynamic

More dynamic than other class-based systems

- Redefinition of classes with instance updates
  (Very complicated, though. See CLHS 4.3.6)
- Adding/removing generic functions and methods.
    - Does not require redefinition of a class.
    - Generic functions and classes conceptually separate.
    - Very dynamic
- Instances can change classes
  (Again, pretty tricky. CLHS 7.2)

WANT

I want:

WANT

I want:

- something with CLOS' nice features, but object-oriented

WANT

I want:

- something with CLOS' nice features, but object-oriented
- something portable, built on Common Lisp

WANT

I want:

- something with CLOS' nice features, but object-oriented
- something portable, built on Common Lisp
- an API similar to CLOS.

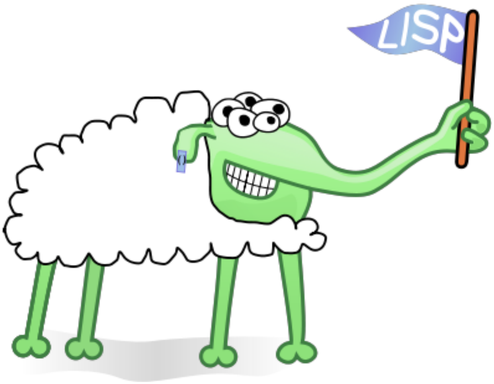
WANT

I want:

- something with CLOS' nice features, but object-oriented
- something portable, built on Common Lisp
- an API similar to CLOS.
- A pony.

Sheeple

Sheeple



Sheeple: Dynamic Object-Orientation for Common Lisp.

Influences

- Written by Josh Marchán and Adlai Chandrasekhar
- Influenced by:
  - CLOS
  - Slate: http://slatelanguage.org
  - Self: http://research.sun.com/self
  - Io: http://iolanguage.com
  - LambdaMOO and similar MUD systems

Objects, with delicious CLOS

- Lisp is dynamic. The object system should dynamic.

Objects, with delicious CLOS

- Lisp is dynamic. The object system should dynamic.
- CLOS+Object-orientation. What more could you possibly want?

Objects, with delicious CLOS

- Lisp is dynamic. The object system should dynamic.
- CLOS+Object-orientation. What more could you possibly want?

```
(object :parents *pony*) => Here's your pony.
```

Looks suspicioully like CLOS on the surface

- Multiple inheritance
- Multiple dispatch
- Method combination
- Lisp integration (autoboxing)
- Interface looks like a M-% of CLOS' API

But has all the proto-goodies you might want underneath. . .

- Objects define behavior
- Completely dynamic
- Dynamic delegation of data (not just behavior)

. . . and it still performs well

Self showed that prototype languages can be efficient.

- Self-inspired maps ("hidden classes")
- Fast property access, reuse of known techniques from Class-land
- Delegation can lead to smaller memory footprint

  NewtonScript exploited this for an embedded system.

It's fun.

Being able to manipulate application on the fly makes programming more fun and productive!

It's fun.

Being able to manipulate application on the fly makes programming more fun and productive!

This T-Rex just downloaded Sheeple.

Object API

Object creation

Object API

Object creation

- Make a new object

```
(object)
```

Object API

Object creation

- Make a new object

```
(object)
```

- Make a new object, that delegates to another object

```
(object :parents *foo*)
```

Object API

Object creation

- Make a new object

```
(object)
```

- Make a new object, that delegates to another object

```
(object :parents *foo*)
```

- Copy an existing object, directly inheriting its properties locally

```
(clone *foo*)
```

Property API

Creating and managing properties

Property API

Creating and managing properties

- Add a property

```
(add-property *foo* 'name "value")
```

Property API

Creating and managing properties

- Add a property

```
(add-property *foo* 'name "value")
```

- Access a property

```
(property-value *foo* 'name)
```

Property API

Creating and managing properties

- Add a property

```
(add-property *foo* 'name "value")
```

- Access a property

```
(property-value *foo* 'name)
```

- Access a delegated property from an object's descendant

```
(property-value (object :parents *foo*) 'name)
```

Property API

Creating and managing properties

- Add a property

```
(add-property *foo* 'name "value")
```

- Access a property

```
(property-value *foo* 'name)
```

- Access a delegated property from an object's descendant

```
(property-value (object :parents *foo*) 'name)
```

- Remove the property

```
(remove-property *foo* 'name)
```

Message API

Creating messages, replies, and dispatching them

Message API

Creating messages, replies, and dispatching them

- Create a message (think generic functions)

```
(defmessage synergize (a b)
  (:documentation "Synergizes A and B"))
```

Message API

Creating messages, replies, and dispatching them

- Create a message (think generic functions)

```
(defmessage synergize (a b)
  (:documentation "Synergizes A and B"))
```

- Create a reply (think methods)

```
(defreply synergize ((a =string=) (b =string=))
  (concatenate 'string a b))
```

Message API

Creating messages, replies, and dispatching them

- Create a message (think generic functions)

```
(defmessage synergize (a b)
  (:documentation "Synergizes A and B"))
```

- Create a reply (think methods)

```
(defreply synergize ((a =string=) (b =string=))
  (concatenate 'string a b))
```

- Call a message (think uhh... calling a function :D)

```
(synergize "foo" "bar") => "foobar"
```

Links:

- Project page on my site: http://sykosomatic.org/sheeple
- Project page on Github: http://github.com/sykopomp/sheeple