Variations in the O-O model
Self

Amer Diwan

Introduction

• In the next 2 lectures we will look at different trends in object-oriented languages
  – First, we will look at a language that tries to strip the power of O-O (and more) to a bare minimum
  – Then we will look at language extensions that researchers are proposing in literature today
Self: the power of simplicity

• Starting point: Smalltalk
  – Everything is an object
  – Primary operation is a message
  – No builtin control structures: handled using “blocks” or closures
  – Same cryptic syntax…

Comparison to Smalltalk (cont.)

• Differences
  – No distinction between classes, metaclasses, and instances
  – No distinction between variable access and message send
  – No static inheritance
  – Methods are also objects
No distinction between classes, meta-classes, and instances

- Objects are created by cloning an existing object
Cloning discussion

• Can clone any object, but which one should I clone?
  – By convention, a program may designate an object as a “prototypical” object (e.g., prototypical point)
  – A prototypical object is similar to a class in this function

• What is a “parent*” field?
  – A place where you continue searching if a particular slot is not found in the receiver
  – Not too different from a superclass

No distinction between variable access and method sends

• Everything is a message send

To change the value of x, a point must send itself the message: x: 10
More on slots

- Depending on how one declares a “selector” (a field in traditional lingo), one gets one or two slots
  - A “read-only” selector gets only one slot (i.e., no x: slot)
  - A “read-write” selector gets two slots (i.e., x and x:)

Advantages of “even field accesses are messages”

- Can compute a field instead of storing it
- Shared state

```
<table>
<thead>
<tr>
<th>parent#</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>2</td>
</tr>
</tbody>
</table>

| y:      | < |
```

```
<table>
<thead>
<tr>
<th>parent#</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>4</td>
</tr>
</tbody>
</table>

| y:      | < |
```
No static inheritance

• If a “parent” is declared read-write, inheritance can be changed on the fly
• Would you want to do that?

Methods are also objects

• Methods are represented by an “activation record” that gets cloned and filled in at a call
Advertised advantages of the prototype based model

- **Simpler relationships**
  - only relationship is “inherits from”
- **Copying instead of instantiating from a class is more natural**
- **Examples of preexisting modules**
  - Modules are more concrete than classes and thus better descriptions
- **Support for one-of-a-kind objects**
- **Elimination of meta-regress**
Where is SELF headed?

- **Behaviorism**
  - The only way to know an object is by its actions
- **Computation viewed as refinement**
  - Refinement of shared behavior
- **Parents viewed as shared parts**

Discussion

- How simple is Self?
- Are you convinced in favor of prototype based languages over class based languages?
- Is it too powerful for programming-in-the-large?