Object Inheritance Without Classes

Timothy Jones, Michael Homer, James Noble
Victoria University of Wellington
{tim,mwh,kjx}@ecs.vuw.ac.nz

Kim Bruce
Pomona College
kim@cs.pomona.edu

July 21, 2016

Objects v Classes

Objects v Classes

Andrew v Kim

Objects v Classes

Andrew v Kim

Objects-first v Objectdraw

Know Thy Self

This problem is solved!

```
( | parent* = other. | )
```

Know Thy Self

This problem is solved!

```
( | parent* = other. | )
```

```
( | parent* = factory new. | )
```

Know Thy Self

This problem was supposed to be solved…

```
( | parent* = other. | )
```

```
( | parent* = factory new. | )
```

Object Inheritance

```
method graphic(canvas) {
  object {
    ...
  }
}
```

```
def amelia = object {
  inherit graphic(canvas)
  ...
}
```

Semantics

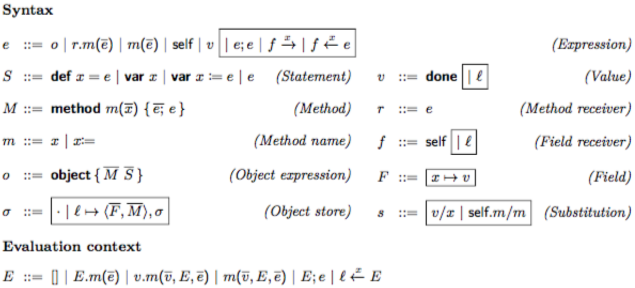
What does this mean?

```
inherit graphic(canvas)
```

Do the inherit semantics actually allow us to implement classes?

- Let's investigate different object inheritance semanticses

Semantics



Syntax

$$e \ ::= \ o \mid r.m(\overline{e}) \mid m(\overline{e}) \mid \mathsf{self} \mid v \ \boxed{\mid e;e \mid f \xrightarrow{x} \mid f \xleftarrow{x} e} \qquad \textit{(Expression)}$$

$$S \ ::= \ \textbf{def}\ x = e \mid \textbf{var}\ x \mid \textbf{var}\ x := e \mid e \qquad \textit{(Statement)}$$

$$v \ ::= \ \textbf{done} \ \boxed{\mid \ell} \qquad \textit{(Value)}$$

$$M \ ::= \ \textbf{method}\ m(\overline{x})\ \{\, \overline{e;}\ e \,\} \qquad \textit{(Method)}$$

$$r \ ::= \ e \qquad \textit{(Method receiver)}$$

$$m \ ::= \ x \mid x{:=} \qquad \textit{(Method name)}$$

$$f \ ::= \ \mathsf{self} \ \boxed{\mid \ell} \qquad \textit{(Field receiver)}$$

$$o \ ::= \ \textbf{object}\ \{\, \overline{M}\ \overline{S} \,\} \qquad \textit{(Object expression)}$$

$$F \ ::= \ \boxed{x \mapsto v} \qquad \textit{(Field)}$$

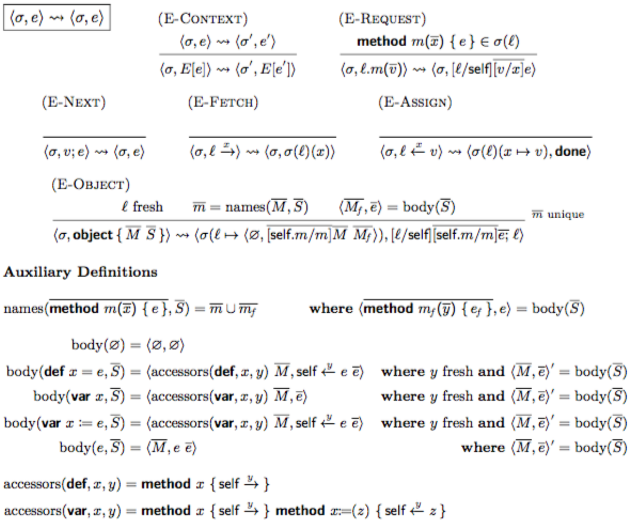
$$\sigma \ ::= \ \boxed{\cdot \mid \ell \mapsto \langle \overline{F}, \overline{M} \rangle, \sigma} \qquad \textit{(Object store)}$$

$$s \ ::= \ \boxed{v/x \mid \mathsf{self}.m/m} \qquad \textit{(Substitution)}$$

Evaluation context

$$E \ ::= \ [] \mid E.m(\overline{e}) \mid v.m(\overline{v}, E, \overline{e}) \mid m(\overline{v}, E, \overline{e}) \mid E;e \mid \ell \xleftarrow{x} E$$

Semantics



$\boxed{\langle \sigma, e\rangle \rightsquigarrow \langle \sigma, e\rangle}$

(E-Context)
$$\frac{\langle \sigma, e\rangle \rightsquigarrow \langle \sigma', e'\rangle}{\langle \sigma, E[e]\rangle \rightsquigarrow \langle \sigma', E[e']\rangle}$$

(E-Request)
$$\frac{\textbf{method}\ m(\overline{x})\ \{\, e \,\} \in \sigma(\ell)}{\langle \sigma, \ell.m(\overline{v})\rangle \rightsquigarrow \langle \sigma, [\ell/\mathsf{self}]\overline{[v/x]}e\rangle}$$

(E-Next)
$$\frac{}{\langle \sigma, v; e\rangle \rightsquigarrow \langle \sigma, e\rangle}$$

(E-Fetch)
$$\frac{}{\langle \sigma, \ell \xrightarrow{x}\rangle \rightsquigarrow \langle \sigma, \sigma(\ell)(x)\rangle}$$

(E-Assign)
$$\frac{}{\langle \sigma, \ell \xleftarrow{x} v\rangle \rightsquigarrow \langle \sigma(\ell)(x \mapsto v), \textbf{done}\rangle}$$

(E-Object)
$$\frac{\ell\ \text{fresh} \qquad \overline{m} = \text{names}(\overline{M}, \overline{S}) \qquad \langle \overline{M_f}, \overline{e}\rangle = \text{body}(\overline{S})}{\langle \sigma, \textbf{object}\ \{\ \overline{M}\ \ \overline{S}\ \}\rangle \rightsquigarrow \langle \sigma(\ell \mapsto \langle \varnothing, \overline{[\mathsf{self}.m/m]M}\ \ \overline{M_f}\rangle), [\ell/\mathsf{self}]\overline{[\mathsf{self}.m/m]}\overline{e}; \ell\rangle}\ \overline{m}\ \text{unique}$$

Auxiliary Definitions

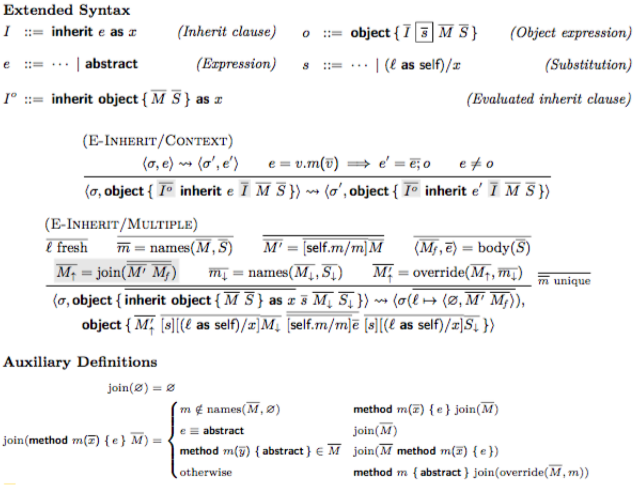
$\text{names}(\overline{\textbf{method}\ m(\overline{x})\ \{\, e \,\}}, \overline{S}) = \overline{m} \cup \overline{m_f}$ **where** $\langle \overline{\textbf{method}\ m_f(\overline{y})\ \{\, e_f \,\}}, e\rangle = \text{body}(\overline{S})$

$$\begin{aligned}
\text{body}(\varnothing) &= \langle \varnothing, \varnothing\rangle \\
\text{body}(\textbf{def}\ x = e, \overline{S}) &= \langle \text{accessors}(\textbf{def}, x, y)\ \overline{M}, \mathsf{self} \xleftarrow{y} e\ \overline{e}\rangle && \textbf{where}\ y\ \text{fresh}\ \textbf{and}\ \langle \overline{M}, \overline{e}\rangle' = \text{body}(\overline{S}) \\
\text{body}(\textbf{var}\ x, \overline{S}) &= \langle \text{accessors}(\textbf{var}, x, y)\ \overline{M}, \overline{e}\rangle && \textbf{where}\ y\ \text{fresh}\ \textbf{and}\ \langle \overline{M}, \overline{e}\rangle' = \text{body}(\overline{S}) \\
\text{body}(\textbf{var}\ x := e, \overline{S}) &= \langle \text{accessors}(\textbf{var}, x, y)\ \overline{M}, \mathsf{self} \xleftarrow{y} e\ \overline{e}\rangle && \textbf{where}\ y\ \text{fresh}\ \textbf{and}\ \langle \overline{M}, \overline{e}\rangle' = \text{body}(\overline{S}) \\
\text{body}(e, \overline{S}) &= \langle \overline{M}, e\ \overline{e}\rangle && \textbf{where}\ \langle \overline{M}, \overline{e}\rangle' = \text{body}(\overline{S})
\end{aligned}$$

$\text{accessors}(\textbf{def}, x, y) = \textbf{method}\ x\ \{\ \mathsf{self} \xrightarrow{y}\ \}$

$\text{accessors}(\textbf{var}, x, y) = \textbf{method}\ x\ \{\ \mathsf{self} \xrightarrow{y}\ \}\ \textbf{method}\ x{:=}(z)\ \{\ \mathsf{self} \xleftarrow{y} z\ \}$

Semantics



Extended Syntax

$I \;::=\; \textsf{\textbf{inherit}}\ e\ \textsf{\textbf{as}}\ x$ *(Inherit clause)* $\quad o \;::=\; \textsf{\textbf{object}}\,\{\,\overline{I}\ \boxed{\overline{s}}\ \overline{M}\ \overline{S}\,\}$ *(Object expression)*

$e \;::=\; \cdots \mid \textsf{\textbf{abstract}}$ *(Expression)* $\quad s \;::=\; \cdots \mid (\ell\ \textsf{\textbf{as}}\ \textsf{self})/x$ *(Substitution)*

$I^{o} \;::=\; \textsf{\textbf{inherit object}}\,\{\,\overline{M}\ \overline{S}\,\}\ \textsf{\textbf{as}}\ x$ *(Evaluated inherit clause)*

(E-Inherit/Context)

$$\frac{\langle \sigma, e\rangle \rightsquigarrow \langle \sigma', e'\rangle \qquad e = v.m(\overline{v}) \implies e' = \overline{e}; o \qquad e \neq o}{\langle \sigma, \textsf{\textbf{object}}\,\{\,\overline{I^{o}}\ \textsf{\textbf{inherit}}\ e\ \overline{I}\ \overline{M}\ \overline{S}\,\}\rangle \rightsquigarrow \langle \sigma', \textsf{\textbf{object}}\,\{\,\overline{I^{o}}\ \textsf{\textbf{inherit}}\ e'\ \overline{I}\ \overline{M}\ \overline{S}\,\}\rangle}$$

(E-Inherit/Multiple)

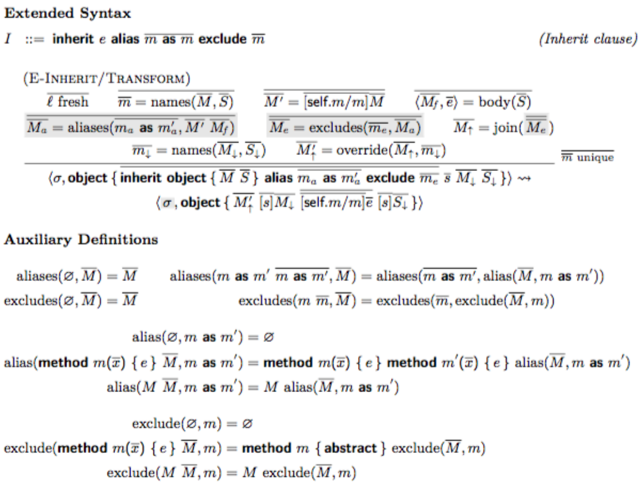
$$\frac{\begin{array}{c}\overline{\ell\ \text{fresh}} \qquad \overline{\overline{m} = \text{names}(\overline{M}, \overline{S})} \qquad \overline{\overline{M'} = [\textsf{self}.m/m]\overline{M}} \qquad \overline{\langle \overline{M_f}, \overline{e}\rangle = \text{body}(\overline{S})} \\ \overline{M_{\uparrow}} = \text{join}(\overline{\overline{M'}\ \overline{M_f}}) \qquad \overline{m_{\downarrow}} = \text{names}(\overline{M_{\downarrow}}, \overline{S_{\downarrow}}) \qquad \overline{M'_{\uparrow}} = \text{override}(\overline{M_{\uparrow}}, \overline{m_{\downarrow}})\end{array}}{\begin{array}{c}\langle \sigma, \textsf{\textbf{object}}\,\{\,\overline{\textsf{\textbf{inherit object}}\,\{\,\overline{M}\ \overline{S}\,\}\ \textsf{\textbf{as}}\ x}\ \overline{s}\ \overline{M_{\downarrow}}\ \overline{S_{\downarrow}}\,\}\rangle \rightsquigarrow \langle \sigma(\overline{\ell \mapsto \langle \varnothing, \overline{M'}\ \overline{M_f}\rangle}), \\ \textsf{\textbf{object}}\,\{\,\overline{M'_{\uparrow}}\ \overline{[s]}[(\ell\ \textsf{\textbf{as}}\ \textsf{self})/x]\overline{M_{\downarrow}}\ \overline{\overline{[\textsf{self}.m/m]}\overline{e}}\ \overline{[s]}[(\ell\ \textsf{\textbf{as}}\ \textsf{self})/x]\overline{S_{\downarrow}}\,\}\rangle\end{array}}\ \overline{\overline{m}\ \text{unique}}$$

Auxiliary Definitions

$$\text{join}(\varnothing) = \varnothing$$

$$\text{join}(\textsf{\textbf{method}}\ m(\overline{x})\ \{\,e\,\}\ \overline{M}) = \begin{cases} m \notin \text{names}(\overline{M}, \varnothing) & \textsf{\textbf{method}}\ m(\overline{x})\ \{\,e\,\}\ \text{join}(\overline{M}) \\ e \equiv \textsf{\textbf{abstract}} & \text{join}(\overline{M}) \\ \textsf{\textbf{method}}\ m(\overline{y})\ \{\,\textsf{\textbf{abstract}}\,\} \in \overline{M} & \text{join}(\overline{M}\ \textsf{\textbf{method}}\ m(\overline{x})\ \{\,e\,\}) \\ \text{otherwise} & \textsf{\textbf{method}}\ m\ \{\,\textsf{\textbf{abstract}}\,\}\ \text{join}(\text{override}(\overline{M}, m)) \end{cases}$$

Semantics



Extended Syntax

$I \quad ::= \quad \textbf{inherit}\ e\ \textbf{alias}\ \overline{m\ \textbf{as}\ m}\ \textbf{exclude}\ \overline{m}$ *(Inherit clause)*

(E-Inherit/Transform)

$$\frac{\begin{array}{c}\overline{\ell\ \text{fresh}} \qquad \overline{\overline{m} = \text{names}(\overline{M}, \overline{S})} \qquad \overline{\overline{M'} = \overline{[\mathsf{self}.m/m]M}} \qquad \overline{\langle \overline{M_f}, \overline{e} \rangle = \text{body}(\overline{S})} \\ \overline{\overline{M_a} = \text{aliases}(\overline{m_a\ \textbf{as}\ m'_a}, \overline{M'}\ \overline{M_f})} \qquad \overline{\overline{M_e} = \text{excludes}(\overline{m_e}, \overline{M_a})} \qquad \overline{M_\uparrow} = \text{join}(\overline{\overline{M_e}}) \\ \overline{m_\downarrow} = \text{names}(\overline{M_\downarrow}, \overline{S_\downarrow}) \qquad \overline{M'_\uparrow} = \text{override}(\overline{M_\uparrow}, \overline{m_\downarrow})\end{array}}{\begin{array}{c}\langle \sigma, \textbf{object}\ \{\ \overline{\textbf{inherit object}\ \{\ \overline{M}\ \overline{S}\ \}\ \textbf{alias}\ \overline{m_a\ \textbf{as}\ m'_a}\ \textbf{exclude}\ \overline{m_e}}\ \overline{s}\ \overline{M_\downarrow}\ \overline{S_\downarrow}\ \} \rangle \rightsquigarrow \\ \langle \sigma, \textbf{object}\ \{\ \overline{M'_\uparrow}\ \overline{[s]M_\downarrow}\ \overline{[\mathsf{self}.m/m]\overline{e}}\ \overline{[s]S_\downarrow}\ \} \rangle\end{array}} \quad \overline{\overline{m}\ \text{unique}}$$

Auxiliary Definitions

$$\text{aliases}(\varnothing, \overline{M}) = \overline{M} \qquad \text{aliases}(m\ \textbf{as}\ m'\ \overline{m\ \textbf{as}\ m'}, \overline{M}) = \text{aliases}(\overline{m\ \textbf{as}\ m'}, \text{alias}(\overline{M}, m\ \textbf{as}\ m'))$$

$$\text{excludes}(\varnothing, \overline{M}) = \overline{M} \qquad \text{excludes}(m\ \overline{m}, \overline{M}) = \text{excludes}(\overline{m}, \text{exclude}(\overline{M}, m))$$

$$\text{alias}(\varnothing, m\ \textbf{as}\ m') = \varnothing$$

$$\text{alias}(\textbf{method}\ m(\overline{x})\ \{\ e\ \}\ \overline{M}, m\ \textbf{as}\ m') = \textbf{method}\ m(\overline{x})\ \{\ e\ \}\ \textbf{method}\ m'(\overline{x})\ \{\ e\ \}\ \text{alias}(\overline{M}, m\ \textbf{as}\ m')$$

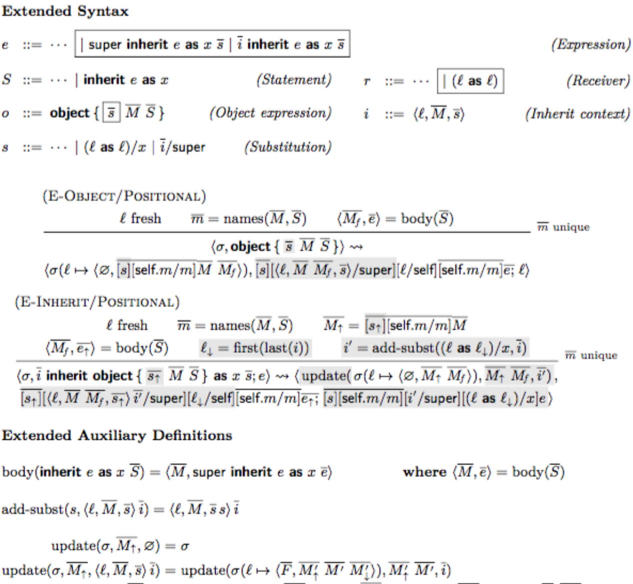
$$\text{alias}(M\ \overline{M}, m\ \textbf{as}\ m') = M\ \text{alias}(\overline{M}, m\ \textbf{as}\ m')$$

$$\text{exclude}(\varnothing, m) = \varnothing$$

$$\text{exclude}(\textbf{method}\ m(\overline{x})\ \{\ e\ \}\ \overline{M}, m) = \textbf{method}\ m\ \{\ \textbf{abstract}\ \}\ \text{exclude}(\overline{M}, m)$$

$$\text{exclude}(M\ \overline{M}, m) = M\ \text{exclude}(\overline{M}, m)$$

Semantics



Extended Syntax

$e \ ::= \ \cdots \ \boxed{\mid \mathsf{super}\ \mathbf{inherit}\ e\ \mathbf{as}\ x\ \overline{s} \mid \overline{i}\ \mathbf{inherit}\ e\ \mathbf{as}\ x\ \overline{s}}$ *(Expression)*

$S \ ::= \ \cdots \mid \mathbf{inherit}\ e\ \mathbf{as}\ x$ *(Statement)* $r \ ::= \ \cdots \ \boxed{\mid (\ell\ \mathbf{as}\ \ell)}$ *(Receiver)*

$o \ ::= \ \mathbf{object}\ \{\ \boxed{\overline{s}}\ \overline{M}\ \overline{S}\ \}$ *(Object expression)* $i \ ::= \ \langle \ell, \overline{M}, \overline{s} \rangle$ *(Inherit context)*

$s \ ::= \ \cdots \mid (\ell\ \mathbf{as}\ \ell)/x \mid \overline{i}/\mathsf{super}$ *(Substitution)*

(E-Object/Positional)

$$\frac{\ell \text{ fresh} \qquad \overline{m} = \mathrm{names}(\overline{M}, \overline{S}) \qquad \langle \overline{M_f}, \overline{e} \rangle = \mathrm{body}(\overline{S})}{\begin{array}{c}\langle \sigma, \mathbf{object}\ \{\ \overline{s}\ \overline{M}\ \overline{S}\ \} \rangle \rightsquigarrow \\ \langle \sigma(\ell \mapsto \langle \varnothing, \overline{[s]}\overline{[\mathsf{self}.m/m]M}\ \overline{M_f} \rangle), \overline{[s]}[\langle \ell, \overline{M}\ \overline{M_f}, \overline{s} \rangle/\mathsf{super}][\ell/\mathsf{self}]\overline{[\mathsf{self}.m/m]e};\ \ell \rangle\end{array}} \ \overline{m} \text{ unique}$$

(E-Inherit/Positional)

$$\frac{\begin{array}{c}\ell \text{ fresh} \qquad \overline{m} = \mathrm{names}(\overline{M}, \overline{S}) \qquad \overline{M_\uparrow} = \overline{[s_\uparrow]}\overline{[\mathsf{self}.m/m]M} \\ \langle \overline{M_f}, \overline{e_\uparrow} \rangle = \mathrm{body}(\overline{S}) \qquad \ell_\downarrow = \mathrm{first}(\mathrm{last}(i)) \qquad i' = \text{add-subst}((\ell\ \mathbf{as}\ \ell_\downarrow)/x, \overline{i})\end{array}}{\begin{array}{c}\langle \sigma, \overline{i}\ \mathbf{inherit\ object}\ \{\ \overline{s_\uparrow}\ \overline{M}\ \overline{S}\ \}\ \mathbf{as}\ x\ \overline{s}; e \rangle \rightsquigarrow \langle \mathrm{update}(\sigma(\ell \mapsto \langle \varnothing, \overline{M_\uparrow}\ \overline{M_f} \rangle), \overline{M_\uparrow}\ \overline{M_f}, \overline{i'}), \\ \overline{[s_\uparrow]}[\langle \ell, \overline{M}\ \overline{M_f}, \overline{s_\uparrow} \rangle\ \overline{i'}/\mathsf{super}][\ell_\downarrow/\mathsf{self}]\overline{[\mathsf{self}.m/m]e_\uparrow};\ \overline{[s]}[\mathsf{self}.m/m][i'/\mathsf{super}][(\ell\ \mathbf{as}\ \ell_\downarrow)/x]e \rangle\end{array}} \ \overline{m} \text{ unique}$$

Extended Auxiliary Definitions

$\mathrm{body}(\mathbf{inherit}\ e\ \mathbf{as}\ x\ \overline{S}) = \langle \overline{M}, \mathsf{super}\ \mathbf{inherit}\ e\ \mathbf{as}\ x\ \overline{e} \rangle$ **where** $\langle \overline{M}, \overline{e} \rangle = \mathrm{body}(\overline{S})$

$\text{add-subst}(s, \langle \ell, \overline{M}, \overline{s} \rangle\ \overline{i}) = \langle \ell, \overline{M}, \overline{s}\ s \rangle\ \overline{i}$

$$\mathrm{update}(\sigma, \overline{M_\uparrow}, \varnothing) = \sigma$$

$$\mathrm{update}(\sigma, \overline{M_\uparrow}, \langle \ell, \overline{M}, \overline{s} \rangle\ \overline{i}) = \mathrm{update}(\sigma(\ell \mapsto \langle \overline{F}, \overline{M'_\uparrow}\ \overline{M'}\ \overline{M'_\downarrow} \rangle), \overline{M'_\uparrow}\ \overline{M'}, \overline{i})$$

Implementation

Runnable semantics with PLT Redex

`https://github.com/zmthy/graceless-redex`

	Reg.	Down.	Dist.	Stable	Exist.	Mult.
Forwarding						
Delegation						
Concatenation						
Merged						
Uniform						
Mult. Uniform						
Transform U.						
Positional U.						
Java	yes	yes	no	yes	class	no

(* indicates true for construction, then reversed afterwards)

Object Inheritance

Objects inherit directly from one another

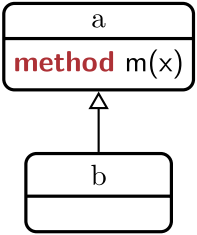
Three foundational models:

- Forwarding (as in E)
- Delegation (as in JavaScript and Self)
- Concatenation (as in Kevo)

Forwarding

Requests to inherited methods go directly to inherited object

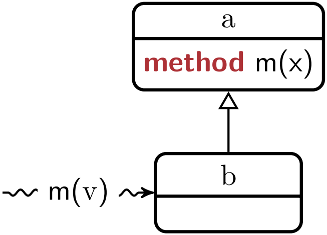
- Simplest semantics



Forwarding

Requests to inherited methods go directly to inherited object

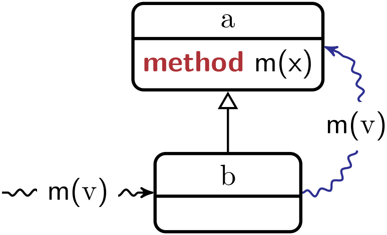
- Simplest semantics



Forwarding

Requests to inherited methods go directly to inherited object

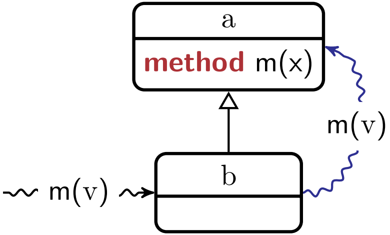
- Simplest semantics



Forwarding

Requests to inherited methods go directly to inherited object

- Simplest semantics



No down-calls (cannot modify existing implementation)

Down-calls

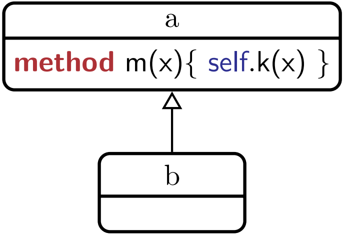
```
method graphic(canvas) {
  object {
    method image { abstract }
    method draw {
      canvas.render( image )
    }
  }
}
```

```
def amelia = object {
  inherit graphic(canvas)
  def image = images.amelia
}
```

Delegation

Requests to inherited methods have self bound to original object

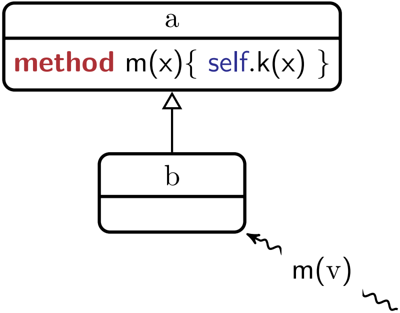
- The standard semantics of object inheritance



Delegation

Requests to inherited methods have self bound to original object

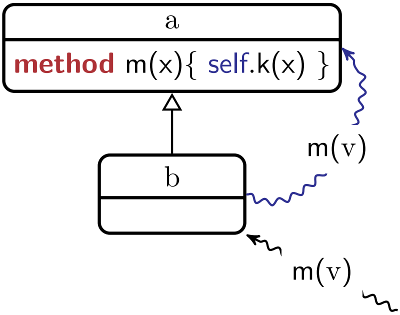
- The standard semantics of object inheritance



Delegation

Requests to inherited methods have self bound to original object

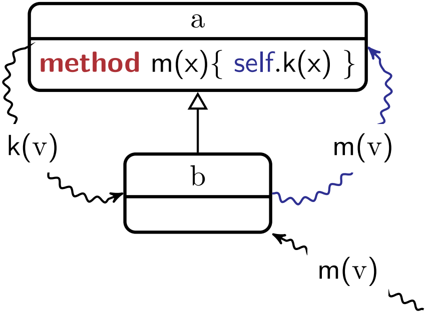
- The standard semantics of object inheritance



Delegation

Requests to inherited methods have self bound to original object

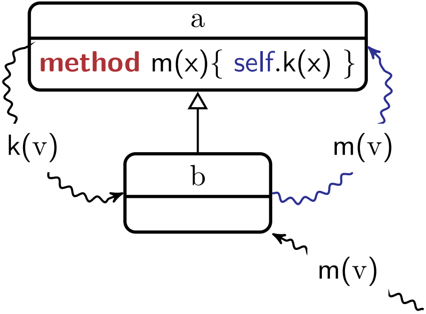
- The standard semantics of object inheritance



Delegation

Requests to inherited methods have self bound to original object

- The standard semantics of object inheritance

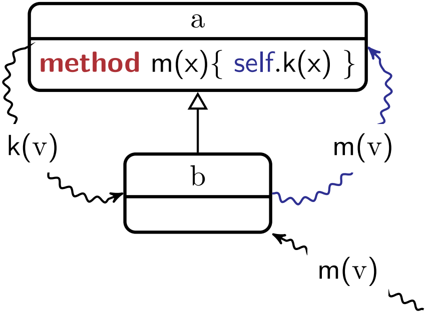


Vampire problem

Delegation

Requests to inherited methods have self bound to original object

- The standard semantics of object inheritance



Vampire problem

Surprising behaviour if you're used to classes

Action at a Distance

```
method graphic(canvas) {
  object {
    var name := "A graphic"
  }
}
```

```
def parent = graphic(canvas)

def amelia = object {
  inherit parent
  name := "Amelia"
}
```

Delegation (as in Self)

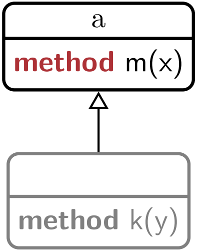
```
above = (|
  value ← 3.
  run = (|| say).
  say = (|| 'above' printLine)
|).

below = (|
  parent* = above.
  say = (|| 'below' printLine)
|
  run.
  value: 5).
```

```
other = (|
  parent* = above.
| value print).
```

Concatenation

Copy the methods and fields from the inherited object

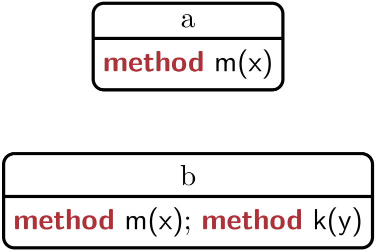
- Removes direct relationship between inheritor and inheritee



Concatenation

Copy the methods and fields from the inherited object

- Removes direct relationship between inheritor and inheritee



a

method m(x)

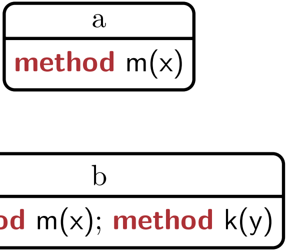
b

method m(x); **method** k(y)

Concatenation

Copy the methods and fields from the inherited object

- Removes direct relationship between inheritor and inheritee



a

method m(x)

b

method m(x); **method** k(y)

Changes to inherited object are not reflected in inheriting object

Registration

```
method graphic(canvas) {
  object {
    canvas.register( self )
  }
}
```

```
def amelia = object {
  inherit graphic(canvas)
}
```

Emulating Classes

Objects inherit from calls to constructor methods

Two class-like models

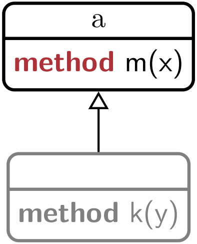
- Merged Identity (as in C++)
- Uniform Identity (as in Java)

Cannot inherit from preëxisting objects

Merged Identity

Inheriting object ‘becomes’ the inherited object

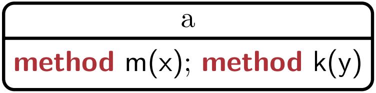
- Registered identities *eventually* resolve to the intended object



Merged Identity

Inheriting object ‘becomes’ the inherited object

- Registered identities *eventually* resolve to the intended object



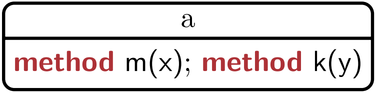
a

method m(x); **method** k(y)

Merged Identity

Inheriting object 'becomes' the inherited object

- Registered identities *eventually* resolve to the intended object



a

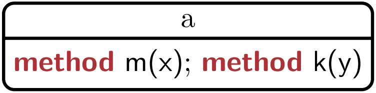
method m(x); **method** k(y)

Body-snatchers problem

Merged Identity

Inheriting object ‘becomes’ the inherited object

- Registered identities *eventually* resolve to the intended object



a

method m(x); **method** k(y)

Body-snatchers problem

Objects not stable during construction

Stability

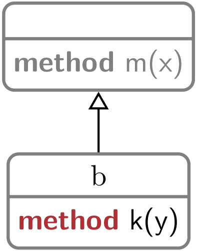
```
method graphic(canvas) {
  object {
    image

    method image { abstract }
  }
}
```

```
def amelia = object {
  inherit graphic(canvas)
  def image = images.amelia
}
```

Uniform Identity

Inherited initialisation code runs as the inheriting object

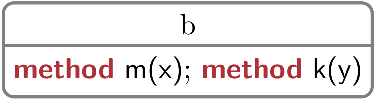
- Basically magic



Uniform Identity

Inherited initialisation code runs as the inheriting object

- Basically magic



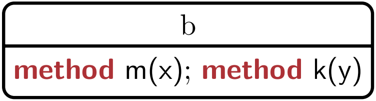
b

method m(x); **method** k(y)

Uniform Identity

Inherited initialisation code runs as the inheriting object

- Basically magic



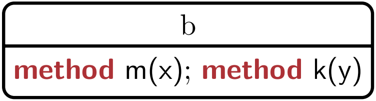
b

method m(x); **method** k(y)

Uniform Identity

Inherited initialisation code runs as the inheriting object

- Basically magic



b

method m(x); **method** k(y)

Uninitialised state during construction

Emulating Classes

Not very satisfactory as foundational models

- No inheritance from preëxisting objects

Emulating Classes

Not very satisfactory as foundational models

- No inheritance from preëxisting objects

Other languages (JavaScript, E) achieve this using other features

Classes in JavaScript

```
function Above() {
   this.value = 3;
   this.say();
}

Above.prototype.run = function () { this.say(); };

function Below() { Above.call(this); }

Below.prototype.say = function () { console.log("hello"); };

new Below().run();
```

Classes in E

```
def makeAbove(self) {
   def above { to run() { self.say() } }
   self ← say()
   return above
}

def below extends makeAbove(below) {
   to say() { println("hello") }
}

below.run()
```

Multiple Inheritance

Every model except merged identity

Various different conflict resolution schemes

- Named supers
- Method transformations
- Positional inheritance

	Reg.	Down.	Dist.	Stable	Exist.	Mult.
Forwarding	no	no	yes	yes	yes	can
Delegation	no	no*	yes	no	yes	can
Concatenation	no	no*	no	no	yes	can
Merged	yes	no*	no	no*	fresh	can't
Uniform	yes	yes	no	yes	fresh	no
Mult. Uniform	yes	yes	no	yes	fresh	yes
Transform U.	yes	yes	no	no	fresh	yes
Positional U.	yes	yes	no	no	fresh	yes
Java	yes	yes	no	yes	class	no

(* indicates true for construction, then reversed afterwards)

	Reg.	Down.	Dist.	Stable	Exist.	Mult.
Forwarding	no	no	yes	yes	yes	can
Delegation	no	no*	yes	no	yes	can
Concatenation	no	no*	no	no	yes	can
Merged	yes	no*	no	no*	fresh	can't
Uniform	yes	yes	no	yes	fresh	no
Mult. Uniform	yes	yes	no	yes	fresh	yes
Transform U.	yes	yes	no	no	fresh	yes
Positional U.	yes	yes	no	no	fresh	yes
Java	yes	yes	no	yes	class	no

(* indicates true for construction, then reversed afterwards)

Conclusion

No obviously superior semantics for object inheritance

Emulating classes requires magic or complicated language features

Ultimately depends on the design goals for the language

Lessons

OO language designers

- Simple foundations do not imply simple design

Lessons

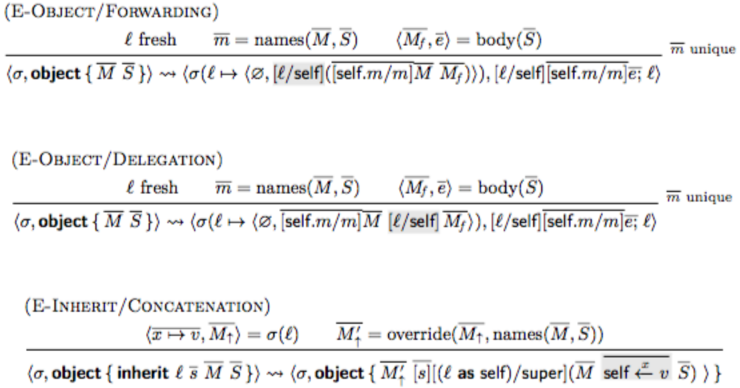
OO language designers

- Simple foundations do not imply simple design

Everyone else

- Problems are hidden in solved designs

Semantics



(E-Object/Forwarding)

$$\frac{\ell \text{ fresh} \qquad \overline{m} = \text{names}(\overline{M}, \overline{S}) \qquad \langle \overline{M_f}, \overline{e} \rangle = \text{body}(\overline{S})}{\langle \sigma, \textbf{object}\,\{\, \overline{M}\ \overline{S}\,\}\rangle \leadsto \langle \sigma(\ell \mapsto \langle \varnothing, [\ell/\mathsf{self}](\overline{[\mathsf{self}.m/m]}\overline{M}\ \overline{M_f})\rangle), [\ell/\mathsf{self}]\overline{[\mathsf{self}.m/m]}\overline{e};\ \ell\rangle} \ \overline{m} \text{ unique}$$

(E-Object/Delegation)

$$\frac{\ell \text{ fresh} \qquad \overline{m} = \text{names}(\overline{M}, \overline{S}) \qquad \langle \overline{M_f}, \overline{e} \rangle = \text{body}(\overline{S})}{\langle \sigma, \textbf{object}\,\{\, \overline{M}\ \overline{S}\,\}\rangle \leadsto \langle \sigma(\ell \mapsto \langle \varnothing, \overline{[\mathsf{self}.m/m]}\overline{M}\ [\ell/\mathsf{self}]\ \overline{M_f}\rangle), [\ell/\mathsf{self}]\overline{[\mathsf{self}.m/m]}\overline{e};\ \ell\rangle} \ \overline{m} \text{ unique}$$

(E-Inherit/Concatenation)

$$\frac{\langle \overline{x \mapsto v}, \overline{M_\uparrow} \rangle = \sigma(\ell) \qquad \overline{M'_\uparrow} = \text{override}(\overline{M_\uparrow}, \text{names}(\overline{M}, \overline{S}))}{\langle \sigma, \textbf{object}\,\{\, \textbf{inherit}\ \ell\ \overline{s}\ \overline{M}\ \overline{S}\,\}\rangle \leadsto \langle \sigma, \textbf{object}\,\{\, \overline{M'_\uparrow}\ \overline{[s]}[(\ell\ \textbf{as}\ \mathsf{self})/\mathsf{super}](\overline{M}\ \ \overline{\mathsf{self} \xleftarrow{x} v}\ \ \overline{S})\ \rangle\,\}}$$

Forwarding (as in E)

```
def above {
  to run() {
    above.say()
  }

  to say() {
    println("above")
  }
}
```

Forwarding (as in E)

```
def above {
  to run() {
    above.say()
  }

  to say() {
    println("above")
  }
}
```

```
def below extends above {
  to say() {
    println("below")
  }
}

below.run()
```

Delegation (as in Self)

```
above = (|
   value ← 3.
   run = (|| say).
   say = (|| 'above' printLine)
|).

below = (|
   parent* = above.
   say = (|| 'below' printLine)
| run).
```

Delegation (as in Self)

```
above = (|
   value ← 3.
   run = (|| say).
   say = (|| 'above' printLine)
|).

below = (|
   parent* = above.
   say = (|| 'below' printLine)
| value: 5).
```

```
other = (|
   parent* = above.
| value print).
```

Delegation (as in JavaScript-ish)

```
let above = {};
above.value = 3;
above.run = function () { this.say(); };
above.say = function () { console.log("above"); };

let below = Object.create(above);
below.say = function () { console.log("below"); };
below.value = 5;

below.run();
console.log(above.value ≠ below.value);
```