Delegation isn't *quite* Inheritance

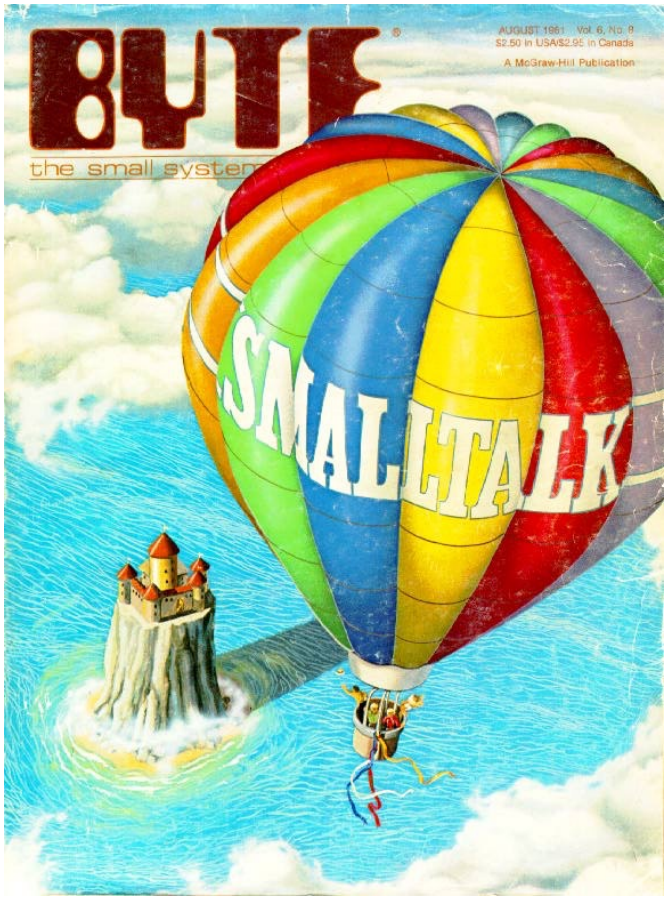
James Noble
school of engineering and computer science
victoria university of wellington
new zealand

Prefixing

```
Class Graphic (X, Y); Real X, Y;  ! Class with two parameters;
 Begin
    Colour Ink; ! drawing ink

    Procedure Draw;       ! Methods
    Begin
    End Draw;

    Ink := new Colour("Black");
    SystemCanvas.register(this)
End;

Graphic Class Rectangle (Width, Height); Real Width, Height;
 Begin
    Procedure Draw;
    Begin
     SystemCanvas.DrawRectangle(X,Y,Width,Height)
    End
 End


Rectangle X := New Rectangle(10,10,20,20);
```

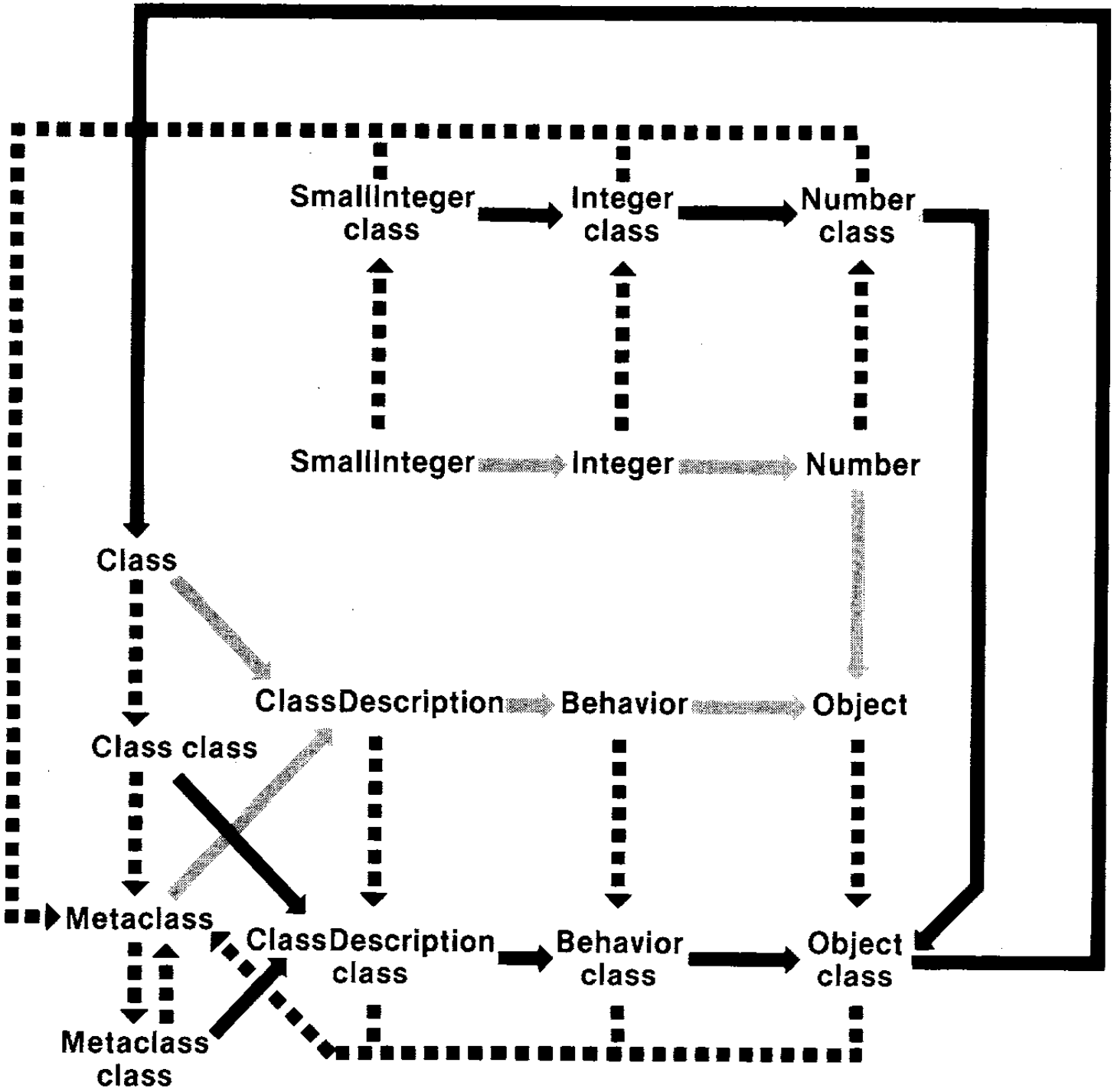
Subclassing

Object subclass: #Rectangle
 instanceVariableNames: 'origin corner'
 classVariableNames: ''
 category: ‘Kernel-BasicObjects'



origin: originPoint **corner:** cornerPoint
 "*Answer an instance of me whose corners*
 (top left and bottom right) are determined by the arguments."

 ^self basicNew setPoint: originPoint point: cornerPoint

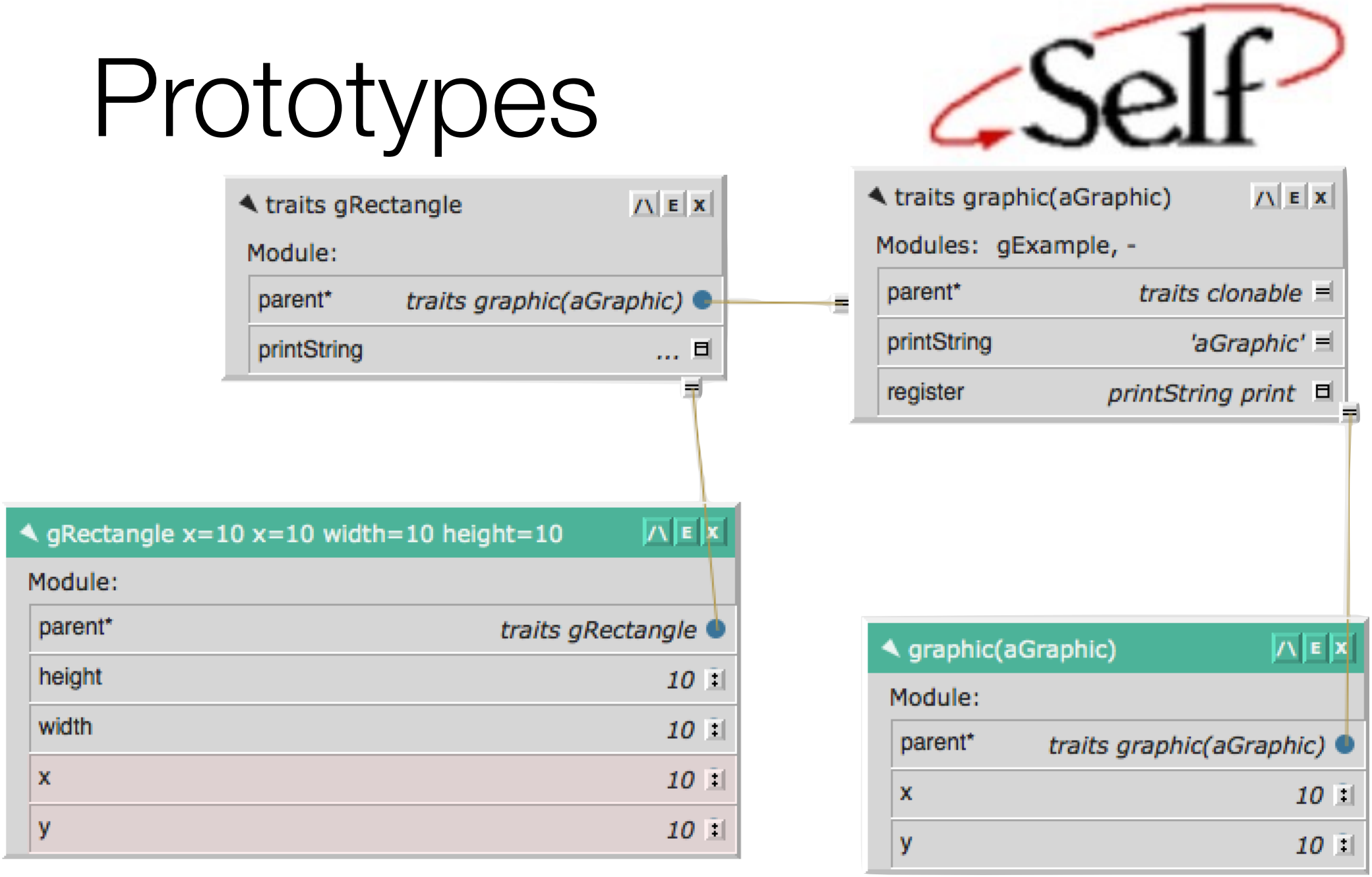

SmallInteger class
Integer class
Number class
SmallInteger
Integer
Number
Class
ClassDescription
Behavior
Object
Class class
Metaclass
ClassDescription class
Behavior class
Object class
Metaclass class

Object Constructors

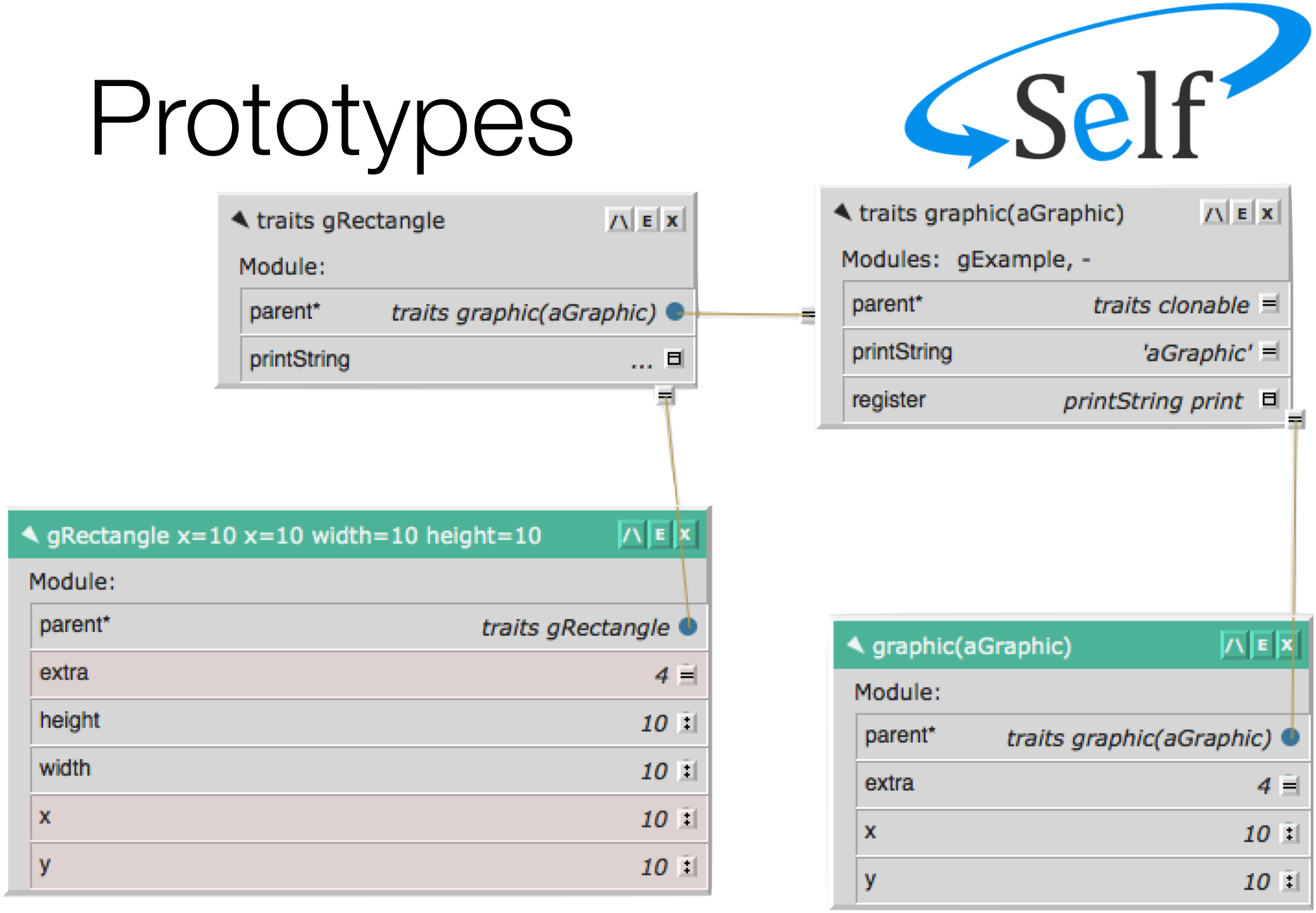
```
const graphicFactory ← object gf
   export function create[ x: Real, y : Real ] → [ r : Graphic ]
      r ← object thisGraphic
               var ink : Colour ← colour["black"]
               export operation draw
               end draw
               process
                  systemCanvas.register[ graphic ]
               end process
            end thisGraphic
   end create
end gf
```

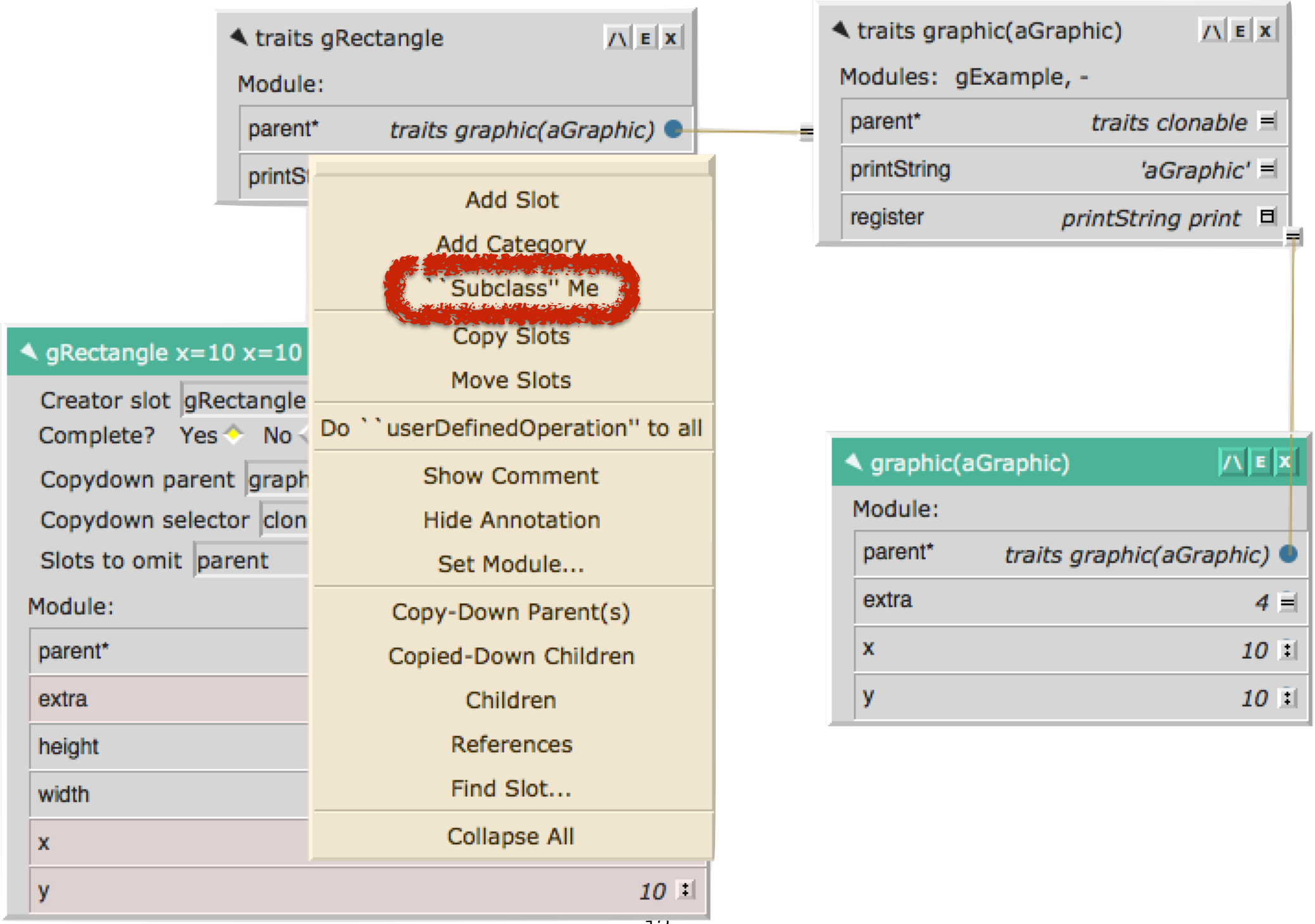
Emerald

Prototypes


traits gRectangle
Module:
parent* traits graphic(aGraphic)
printString ...
traits graphic(aGraphic)
Modules: gExample, -
parent* traits clonable
printString 'aGraphic'
register printString print
gRectangle x=10 x=10 width=10 height=10
Module:
parent* traits gRectangle
height 10
width 10
x 10
y 10
graphic(aGraphic)
Module:
parent* traits graphic(aGraphic)
x 10
y 10

Prototypes


traits gRectangle
Module:
parent* traits graphic(aGraphic)
printString ...
traits graphic(aGraphic)
Modules: gExample, -
parent* traits clonable
printString 'aGraphic'
register printString print
gRectangle x=10 x=10 width=10 height=10
Module:
parent* traits gRectangle
extra 4
height 10
width 10
x 10
y 10
graphic(aGraphic)
Module:
parent* traits graphic(aGraphic)
extra 4
x 10
y 10


traits gRectangle
Module:
parent* traits graphic(aGraphic)
traits graphic(aGraphic)
Modules: gExample, -
parent* traits clonable
printString 'aGraphic'
register printString print
Add Slot
Add Category
``Subclass'' Me
Copy Slots
Move Slots
Do ``userDefinedOperation'' to all
Show Comment
Hide Annotation
Set Module...
Copy-Down Parent(s)
Copied-Down Children
Children
References
Find Slot...
Collapse All
gRectangle x=10 x=10
Creator slot gRectangle
Complete? Yes No
Copydown parent graph
Copydown selector clon
Slots to omit parent
Module:
parent*
extra
height
width
x
y 10
graphic(aGraphic)
Module:
parent* traits graphic(aGraphic)
extra 4
x 10
y 10

```
let mouseFactory =
 function mouseFactory () { return
    Object.assign(Object.create(animal), {

 extend(object,
   Events);

 let mouse =
 Object.assign(
    Object.create(animal), {

assign({}, // create a new object
 skydiving,
 ninja,
 mouse,
 wingsuit);
```

https://medium.com/javascript-scene

```
method Graphic (x : Number, y : Number) = object {
  var ink : Colour = colour("black")
  method draw is abstract { }
  systemCanvas.register(self)
 }
}
```

Grace

```
def rectangle = object {
        inherits Graphic(x , y)
    method draw is override {
        systemCanvas.drawRectangle(x, y, width, height)
      }
  }
}
```

```
class Graphic (x : Number, y : Number) {
  var ink : Colour = colour("black")
  method draw is abstract { }
  systemCanvas.register(self)
}
```

Grace

```
class Rectangle (width : Number, height : Number) {
      inherits Graphic(x , y)
   method draw is override {
      systemCanvas.drawRectangle(x, y, width, height)
   }
}
```

Let's pretend it's 1995 (and dance to Teenage Fanclub)

Begin forwarded message:

From: James Noble <kjx@ecs.vuw.ac.nz>
Subject: Minutes of Teleconference 2-3.8.12
Date: 3 August 2012 15:02:09 pm NZST
To: Kim Bruce <kim@cs.pomona.edu>, "Andrew P. Black" <black@cs.pdx.edu>
Cc: grace-core@cecs.pdx.edu

We talked mostly about inheritance, a little about dialects

* Delegation is strictly stronger than concatenation - because
  concatenation can be simulated by delegating to a (shallow) copy
  (from Michael "Mr Literal" Homer)

* Reiterated from last week: PICK TWO:
 1. "classical" inheritance semantics - "self" bound to sub-object
    while super-object literal executes
 2. inheritance from an arbitrary object
 3. a simple explanation of classes in terms of objects

Classes?

- **Self** — copy down slots, subclassing
- **JS** — 20+? different “class” libraries
- **Lua** —13 different “class” libraries
  (http://lua-users.org/wiki/ObjectOrientedProgramming)
- **Emerald** — implemented classes, didn’t admit it

Traits

```
trait Graphic(x : Number, y : Number) {
  method x is confidential, abstract {}
  method y is confidential, abstract {}
  var ink : Colour = colour("black")
  method ink -> Colour is abstract
  method ink:= (c : Colour) is abstract
  method draw is abstract { }

  systemCanvas.register(self)
}
```

Grace

Multiple Traits

```
class AnimatedRectangle (x' : Number, y' : Number,
                         width : Number, height : Number) {
    uses Graphic
    uses Animated
  def x = x'
  def y = y'
  var ink : Colour = colour("black")
  method draw {
    systemCanvas.drawRectangle(x, y, width, height)
  }
  ....
}
```

Grace

Prefixing

```
class Top {
    method a { … }
}
class Middle {
    inherits Top
        alias topA = a
    method a { … }
}
class Bottom {
    inherits Middle
    method c { … }
}
```

```
class Bottom {
    method topA { … }

    method a { … }

    method c { … }
}
```