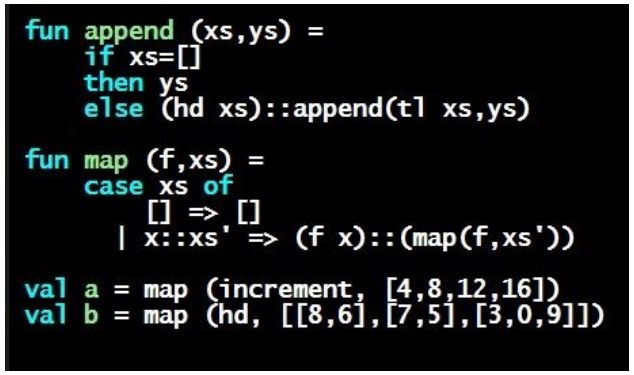

Programming Languages

Dan Grossman
2013

Why Use Subclassing?

Example continued

- Consider alternatives to:

```
class ColorPoint < Point
  attr_accessor  :color
  def initialize(x,y,c)
    super(x,y)
    @color = c
  end
end
```

- Here subclassing is a good choice, but programmers often overuse subclassing in OOP languages

Why subclass

- Instead of creating **`ColorPoint`**, could add methods to **`Point`**
  - That could mess up other users and subclassers of **`Point`**

```
class Point
  attr_accessor :color
  def initialize(x,y,c="clear")
    @x = x
    @y = y
    @color = c
  end
end
```

Why subclass

- Instead of subclassing **`Point`**, could copy/paste the methods
  - Means the same thing *if* you don't use methods like **`is_a?`** and **`superclass`**, but of course code reuse is nice

```
class ColorPoint
  attr_accessor :x, :y, :color
  def initialize(x,y,c="clear")
    …
  end
  def distFromOrigin
    Math.sqrt(@x*@x + @y*@y)
  end
  def distFromOrigin2
    Math.sqrt(x*x + y*y)
  end
end
```

Why subclass

- Instead of subclassing **`Point`**, could use a **`Point`** instance variable
  - Define methods to send same message to the **`Point`**
  - Often OOP programmers overuse subclassing
  - But for **`ColorPoint`**, subclassing makes sense: less work and can use a **`ColorPoint`** wherever code expects a **`Point`**

```
class ColorPoint
  attr_accessor :color
  def initialize(x,y,c="clear")
    @pt = Point.new(x,y)
    @color = c
  end
  def x
    @pt.x
  end
  … # similar "forwarding" methods

    # for y, x=, y=
end
```