Lessons from the Dancing Box Homework

CS 420–520

Andrew P. Black

Portland State UNIVERSITY

Write Purpose Statements

- A purpose statement is a comment just after the method header that explains:
  - what the method does
  - what the method returns, and
  - the rôles of the parameters.
- Use one whenever the *name* of the method alone is insufficient

- Does this method need a purpose statement?

```
method maximum(a:Number, b:Number) → Number { ... }
```

- How about this?

```
//used in update_dir
method update_spd { ... }
```

- How about this?

```
method update_dir(pt:Point) {
        //tells the animator which way the box will move
        ...
}
```

- Better

```
method changeHeading(pt:Point) → Done {
        // modifies heading to try to keep pt inside the canvas
        ...
}
```

- Notes:
  - Put purpose statements *inside* the method that they describe
    - **method** keyword will help us find the method
    - parameter names will be in scope
  - Refer to the parameters by name; say what they do
  - If the method returns something other than Done, say what it "*answers*" or "*returns*" (use those words)
  - Not being able to write a concise purpose statement is a code smell

```
method danceWith(another) {
    if (! another.acceptDance(self)) then {
        return false
    }
    partner := another
    hasPartner := true
    print "{getName} dancing with {partner.getName}"
    dance;
}
```

```
method danceWith(another) → Boolean {
    // asks another to dance; answers false if they decline and ???
    if (! another.acceptDance(self)) then {
        return false
    }
    partner := another
    hasPartner := true
    print "{getName} dancing with {partner.getName}"
    dance;
}
```

Trust your objects

- If alice asks bob to dance, and he accepts, then alice should *trust* bob to move his own feet

```
method dance {
    var increment := (random.integerIn(-2) to(2) @
                      random.integerIn(-2) to(2))
    var count := 0
    animation.while {count < 10} pausing 100 do {
        count := count + 1
        moveBy(increment)
        if (hasPartner) then {
            partner.moveTo(origin - increment)
        }
    }
    ...
```

Use complex objects, not primitives

- Point are 2-D vectors; can be used to represent velocities (speed & direction)

```
var xspd := 0
var yspd := 0
```

why not

```
var velocity = 0@0
```

- Why use a scaler speed and a scalar heading (compass bearing), when you can use a vector velocity?

Sometimes, methods are missing

- No % (modulus) operator on points
- In a better language, we could add it!
- InGrace, we have to fake it

```
method stayInBoundariesFor(location:Point) → Point {
        def effectiveSize = size - extent
        (location.x % effectiveSize.x) @
              (location.y % effectiveSize.y)
}
```