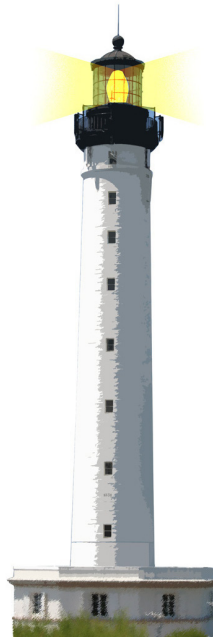


Objects vs. Data

an API perspective studying the class Point

S. Ducasse



Goals

- Difference between an object and a data structure
- Difference between a poor and a good API
- APIs and encapsulation play an important role
- Looking at two concrete implementations of Point: in Java and Pharo
- Understanding the impact of strong API



Java Points - Getters and setters

- `Point getLocation()`: returns the location of this point (to be polymorphic with **Component**. A location is just a point.)
- `void setLocation(double x, double y)`: sets the location of this point to the **specified double coordinates**.
- `void setLocation(int x, int y)`: changes the point to have the specified location.
- `void setLocation(Point p)`: sets the location of the point to the specified location.
- `double getX()`: returns the X coordinate of this `Point2D` in double precision.
- `double getY()`: returns the Y coordinate of this `Point2D` in double precision.



Java Points - the 'rest'

- `boolean equals(Object obj)`: whether or not two points are equal.
- `void move(int x, int y)`: moves this point to the specified location in the (x,y) coordinate plane.
- `void translate(int dx, int dy)`: translates this point, at location (x,y) , by dx along the x axis and dy along the y axis so that it now represents the point $(x+dx,y+dy)$.
- `String toString()`: returns a string representation of this point and its location in the (x,y) coordinate space.

Inherited from `Point2D`

- `distance()` and `clone()`



Analysis: Java Point

- A super poor data structure
- A dry holder of integers
- Super **limited** interface
- Java points do not define behavior **beside** move, translate **and** distance!



Points in Pharo

Rich API (selected part):

- normalized, normal, transposed, reflectedAbout:
- distanceTo:, squaredDistanceTo:
- crossProduct:, dotProduct:
- \ - *, reciprocal,/, +, min // abs max
- >= > <= min:max: min: < closeTo: closeTo:precision: max: =
- negated, translateBy:, scaleBy:, scaleTo:, scaleFrom:to:, adhereTo:,
- triangleArea:with:, to:intersects:to:, to:sideOf:, isInsideCircle:with:with:, sideOf:,
- rectangle:, extent:, corner:



Points in Pharo (Continued)

- degrees, theta,
- onLineFrom:to:, angleWith:, angle, rotateBy:about:, rotateBy:centerAt:, bearingToPoint:,
- roundUpTo:, ceiling, truncated, truncateTo:, roundTo:, floor, roundDownTo:, rounded,
- quadrantOf:, leftRotated, nearestPointAlongLineFrom:to:, flipBy:centerAt:, nearestPointOnLineFrom:to:, squaredDistanceTo:, insideTriangle:with:with:, directionToLineFrom:to:, sign, octantOf:, rightRotated,
- fourNeighbors, grid:, eightNeighbors, fourDirections



Simple example

Point >> crossProduct: aPoint

"Answer a number that is the cross product of the receiver and the argument, aPoint."

$\wedge (x * aPoint y) - (y * aPoint x)$

- Obvious, but still useful
- No need to duplicate it in clients

Simple example: comparing points

< aPoint

"Answer whether the receiver is above and to the left of aPoint."

^ x < aPoint x and: [y < aPoint y]



Example: More challenging

Point >> degrees

"Answer the angle the receiver makes with origin in degrees. right is 0; down is 90."

```
| tan theta |
```

```
^ x = 0
```

```
if True: [ y >= 0
```

```
    if True: [ 90.0 ]
```

```
    if False: [ 270.0 ] ]
```

```
if False: [ tan := y asFloat / x asFloat.
```

```
    theta := tan arcTan.
```

```
    x >= 0
```

```
        if True: [ y >= 0
```

```
            if True: [ theta radiansToDegrees ]
```

```
            if False: [ 360.0 + theta radiansToDegrees ] ]
```

```
        if False: [ 180.0 + theta radiansToDegrees ] ]
```

Nobody wants to be forced to reimplement it.



An example in Java

How to make a robot walk a distance from its current direction (in degrees).

```
public class Bot {  
    int tilt = 0;  
    Point position = new Point(0,0);
```

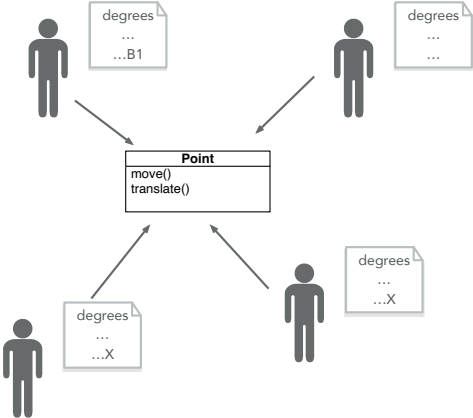
```
    public void go(int distance){  
        position = new Point(  
            (Math.round(Math.cos(Math.toRadians(tilt))) * distance + position.x()),  
            (Math.round(Math.sin(Math.toRadians(tilt))) * distance + position.y()));  
    }  
}
```

Analysis (2)

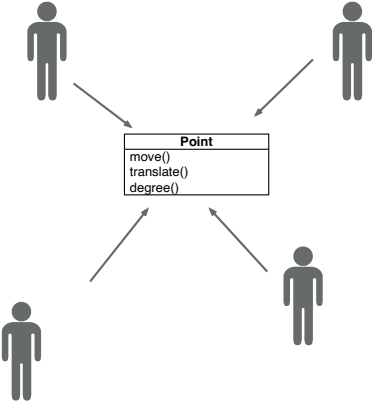
- Have to recreate explicitly a point distance + position.x()
- Arithmetic of Points is defined **outside** of them!
 - Points cannot sum themselves
 - Points cannot shape themselves (rounded, ...)
- When an object exposes a shallow API, it favors **logic duplication** in clients!



Consequences of poor APIs



VS



Bot » go: in Pharo

In Java

```
public void go(int distance){
    position = new Point(
        (Math.round(Math.cos(Math.toRadians(tilt))) * distance + position.x()),
        (Math.round(Math.sin(Math.toRadians(tilt))) * distance + position.y()));
}
```

In Pharo

```
Bot >> go: aDistance
    position := position + ((tilt degreeCos @ tilt degreeSin) * aDistance) rounded
```

- Use **Point**'s addition, multiplication, and rounding
- Use **Number**'s sin and cos
- Points know how to $*$, $+$, $/$, ... **themselves**
- We can compose points, rectangles, and numbers

Analysis

- In Pharo Points are more than a data structure
- They define **advanced** behavior
- Functionality is not in clients
- Clients **benefit and reuse** behavior!



What you should know

- Objects are not data structures
- Objects are more than structure
- Objects are about behavior and services they offer
- An object should encapsulate logic and lets its client **reuse** such logic!

A course by

S. Ducasse, G. Polito, and Pablo Tesone



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>