A Swift Kickstart

Daniel H Steinberg
@dimsumthinking

Protocols

Start here

```
struct Vertex {
    var x, y: Double
    func moveByX(x: Double) -> Vertex {
        return Vertex(x: self.x + x, y: y)
    }
}

struct Size {
    var width, height: Double
}

struct Rectangle {
    var size: Size
    var topLeftCorner: Vertex
}

var point = Vertex(x: 3.0, y: 4.0)
let shiftedPoint = point.moveByX(4)
point
shiftedPoint

let rectangle = Rectangle(size: Size(width: 200, height: 100),
                          topLeftCorner: point)
```

Protocol

```
protocol Movable {
    func moveByX(x: Double) -> Movable
}
```

```
struct Vertex {
    var x, y: Double
    func moveByX(x: Double) -> Vertex {
        return Vertex(x: self.x + x, y: y)
    }
}
```

Use the Protocol

```
protocol Movable {
    func moveByX(x: Double) -> Movable
}

func shiftLeft(movable: Movable) -> Movable {
    return movable.moveByX(-1)
}
```

Not yet

```
protocol Movable {
    func moveByX(x: Double) -> Movable
}

func shiftLeft(movable: Movable) -> Movable {
    return movable.moveByX(-1)
}

var point = Vertex(x: 3.0, y: 4.0)
let shiftLeftPoint = shiftLeft(point)
```

Still no

```
protocol Movable {
    func moveByX(x: Double) -> Movable
}

func shiftLeft(movable: Movable) -> Movable {
    return movable.moveByX(-1)
}

struct Vertex: Movable  {
    var x, y: Double
    func moveByX(x: Double) -> Vertex {
        return Vertex(x: self.x + x, y: y)
    }
}

var point = Vertex(x: 3.0, y: 4.0)
let shiftedPoint = shiftLeft(point)
```

Could change return type for Vertex

```
protocol Movable {
    func moveByX(x: Double) -> Movable
}

func shiftLeft(movable: Movable) -> Movable {
    return movable.moveByX(-1)
}

struct Vertex: Movable  {
    var x, y: Double
    func moveByX(x: Double) -> Movable {
        return Vertex(x: self.x + x, y: y)
    }
}

var point = Vertex(x: 3.0, y: 4.0)
let shiftedPoint = shiftLeft(point)
```

Prefer to change return type for Movable to Self

```
protocol Movable {
    func moveByX(x: Double) -> Self
}

func shiftLeft(movable: Movable) -> Movable {
    return movable.moveByX(-1)
}

struct Vertex: Movable  {
    var x, y: Double
    func moveByX(x: Double) -> Vertex {
        return Vertex(x: self.x + x, y: y)
    }
}

var point = Vertex(x: 3.0, y: 4.0)
let shiftedPoint = shiftLeft(point)
```

Conform to Movable

```
struct Rectangle: Movable {
    var size: Size
    var topLeftCorner: Vertex
    func moveByX(x: Double) -> Rectangle {
        let movedTopLeftCorner
                = topLeftCorner.moveByX(x)
        return Rectangle(size: size,
                 topLeftCorner: movedTopLeftCorner)
    }
}
```

Not Polymorphism

```
var point = Vertex(x: 3.0, y: 4.0)
let rectangle = Rectangle(size: Size(width: 200.0,
                                     height: 100.0),
                          topLeftCorner: point)

let shiftedPoint = point.moveByX(4)
point
shiftedPoint

let shiftedRectangle = rectangle.moveByX(10)
rectangle
shiftedRectangle
```

Polymorphism

```
func shiftLeft(movable: Movable) -> Movable {
    return movable.moveByX(-1.0)
}

let shiftedLeftPoint = shiftLeft(point)
let shiftedLeftRectangle = shiftLeft(rectangle)
```

Not correct

```
func shiftLeft(movable: Movable) -> Movable {
    return movable.moveByX(-1.0)
}

let shiftedLeftPoint = shiftLeft(point)
let shiftedLeftRectangle = shiftLeft(rectangle)

let shiftedLeftRectangle2 = Rectangle(size: rectangle.size,
                                      topLeftCorner: shiftedLeftPoint)
```

as ?

```
func shiftLeft(movable: Movable) -> Movable {
    return movable.moveByX(-1.0)
}
let shiftedLeftRectangle = shiftLeft(rectangle)
var shiftedLeftRectangle2:Rectangle

if let shiftedLeftPoint = shiftLeft(point) as? Vertex {

shiftedLeftRectangle2 = Rectangle(size: rectangle.size,
                                  topLeftCorner: shiftedLeftPoint)
}
```

Generics

```
func shiftLeft<T: Movable>(movable:T) -> T {
    return movable.moveByX(-1.0)
}

let shiftedLeftPoint = shiftLeft(point)

shiftedLeftRectangle2 = Rectangle(size: rectangle.size,
                                  topLeftCorner: shiftedLeftPoint)
```

Another method

```
protocol Movable {
    func moveByX(x: Double) -> Self
    func moveLeftOf(otherMovable: Self) -> Self
}
```

Another method

```
struct Vertex: Movable {
    var x, y: Double
    func moveByX(x: Double) -> Vertex  {
        return Vertex(x: self.x + x, y: y)
    }
    func moveLeftOf(otherVertex: Vertex) -> Vertex  {
        return moveByX(otherVertex.x - x - 1)
    }
}
```

Rectangle as well

```
struct Rectangle: Movable {
    var size: Size
    var topLeftCorner: Vertex
    func moveByX(x: Double) -> Rectangle {
        let movedTopLeftCorner = topLeftCorner.moveByX(x)
        return Rectangle(size: size,
                topLeftCorner: movedTopLeftCorner)
    }
    func moveLeftOf(otherRectangle: Rectangle) -> Rectangle  {
        return moveByX(otherRectangle.topLeftCorner.x
                    + size.width - topLeftCorner.x - 1)
    }
}
```

Restricted Polymorphism

```
func move<T: Movable>(objectToBeMoved: T,
                      nextToObject otherObject: T) -> T {
        return objectToBeMoved.moveLeftOf(otherObject)
}
```

Restricted Polymorphism

```
func move<T: Movable>(objectToBeMoved: T,
    nextToObject otherObject: T) -> T {
        return objectToBeMoved.moveLeftOf(otherObject)
}

move(point, nextToObject:shiftedPoint)
move(rectangle, nextToObject: shiftedRectangle)

move(point, nextToObject: rectangle) // can't do this
move(rectangle, nextToObject: point) // can't do this
```

Try this

- Create a new playground with a Pourable protocol that declares a method named drink() that takes an amount and returns a Pourable

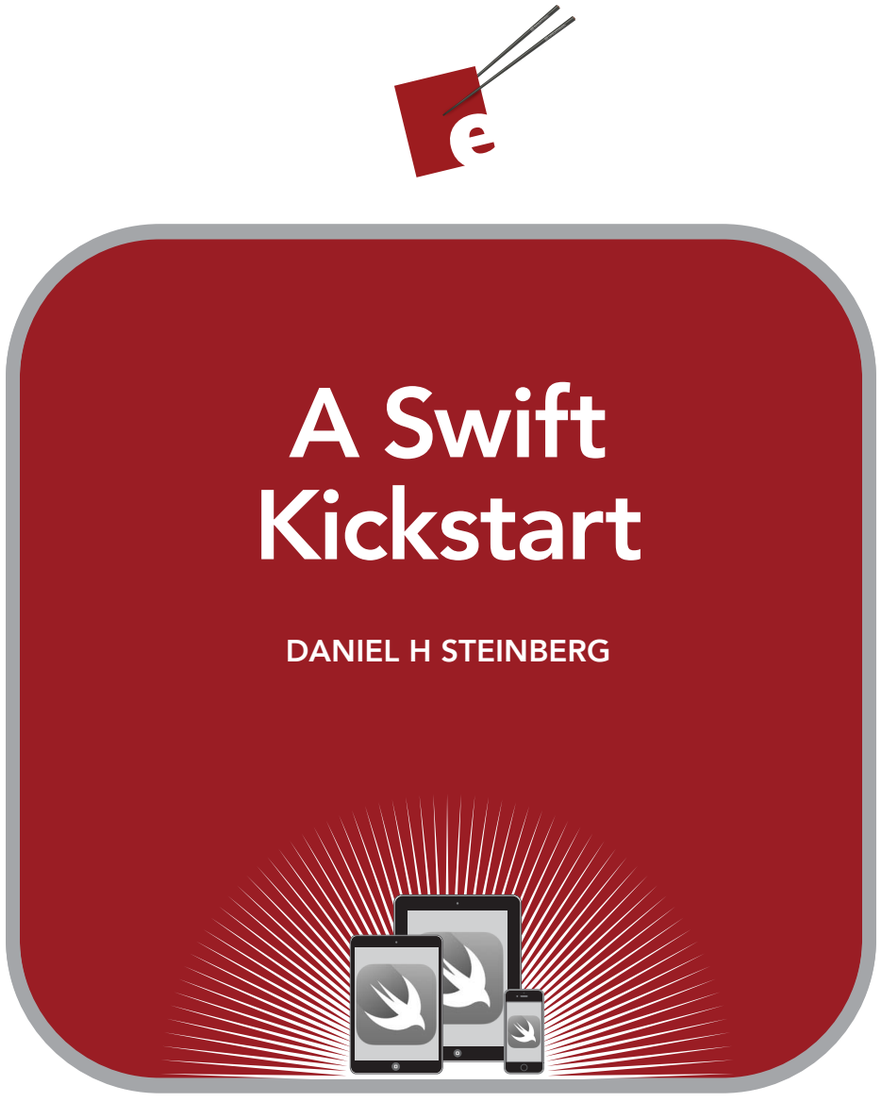
- Create structs named HotBev and ColdBev that are pourable. They keep track of the amountRemaining.

- Implement drink() in each to return a struct of type HotBev/ColdBev with the correct amount remaining.

- Add sip() to HotBev that drinks 1 and gulp() to ColdBev that drinks 4

Try this

```
protocol Pourable {
    func drink(amount: Int) -> Pourable
}
enum SizeOfCup: Int {
    case Small = 8, Medium = 12, Large = 16
}
struct HotBev: Pourable {
    var amountRemaining: Int
    func drink(amount: Int) -> Pourable {
        if amountRemaining > amount {
            return HotBev(amountRemaining: amountRemaining - amount)
        } else {
            return self
        }
    }
    func sip() -> Pourable {
        return drink(1)
    }
}
struct ColdBev: Pourable {
    var amountRemaining: Int
    func drink(amount: Int) -> Pourable {
        if amountRemaining > amount {
            return ColdBev(amountRemaining: amountRemaining - amount)
        } else {
            return self
        }
    }
    func gulp() -> Pourable {
        return drink(4)
    }
}
var coffee = HotBev(amountRemaining: SizeOfCup.Medium.toRaw())
for i in 1...8 { coffee.sip() }
var iceTea = ColdBev(amountRemaining: SizeOfCup.Large.toRaw())
for i in 1...8 { iceTea.gulp()
}
```



https://itunes.apple.com/us/book/a-swift-kickstart/id891801923?mt=11&uo=4&at=11l56E