Developer Tools

#WWDC15

# Protocol-Oriented Programming in Swift

Session 408

Dave Abrahams Professor of Blowing-Your-Mind

## Meet Crusty

Don’t call him “Jerome”

## Classes Are Awesome

## Classes Are Awesome

- Encapsulation

## Classes Are Awesome

- Encapsulation
- Access Control

## Classes Are Awesome

- Encapsulation
- Access Control
- Abstraction

# Classes Are Awesome

- Encapsulation
- Access Control
- Abstraction
- Namespace

## Classes Are Awesome

- Encapsulation
- Access Control
- Abstraction
- Namespace
- Expressive Syntax

## Classes Are Awesome

- Encapsulation
- Access Control
- Abstraction
- Namespace
- Expressive Syntax
- Extensibility

# Classes Are Awesome

- Encapsulation
- Access Control
- Abstraction
- Namespace
- Expressive Syntax
- Extensibility

## Classes Are Awesome

- Encapsulation
- Access Control
- Abstraction
- Namespace
- Expressive Syntax
- Extensibility

## ~~Classes~~ Types Are Awesome

- Encapsulation
- Access Control
- Abstraction
- Namespace
- Expressive Syntax
- Extensibility

I can do all that with structs

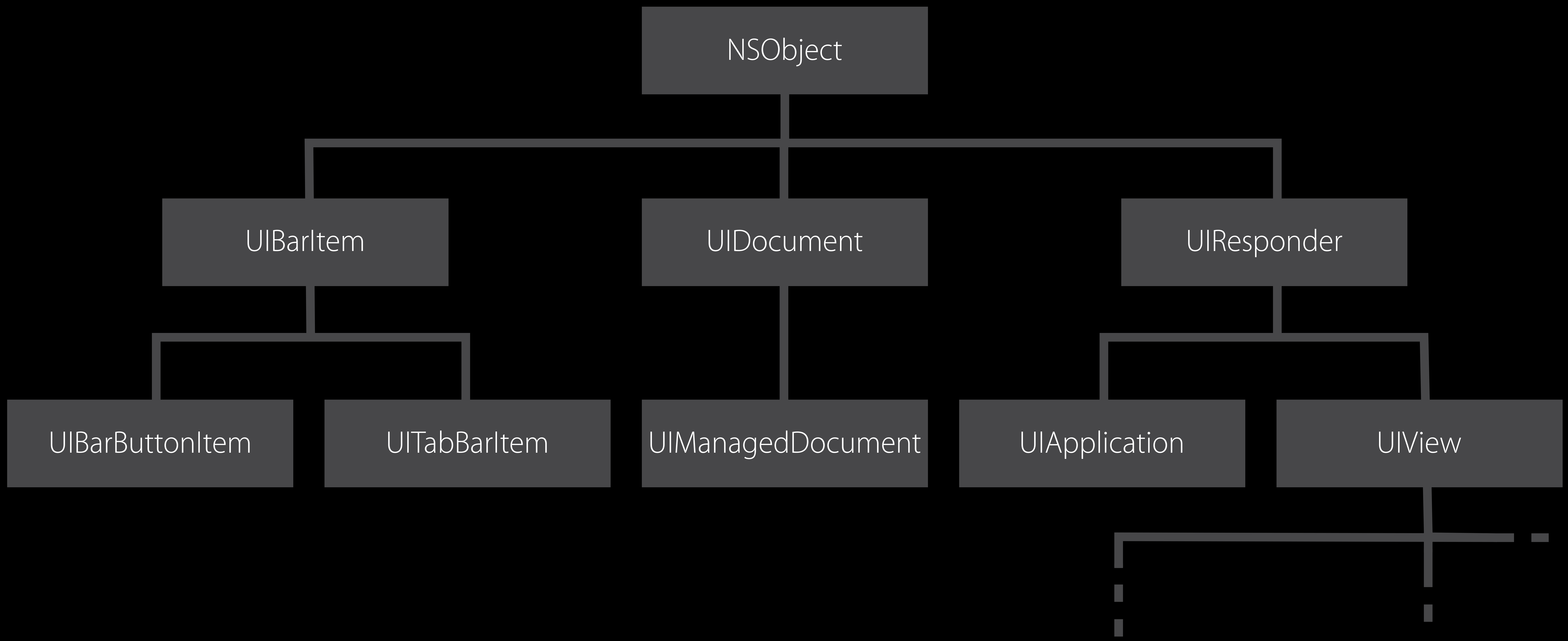
# ~~Classes~~ Types Are Awesome

- Encapsulation
- Access Control
- Abstraction
- Namespace
- Expressive Syntax
- Extensibility

I can do all that with structs and enums.

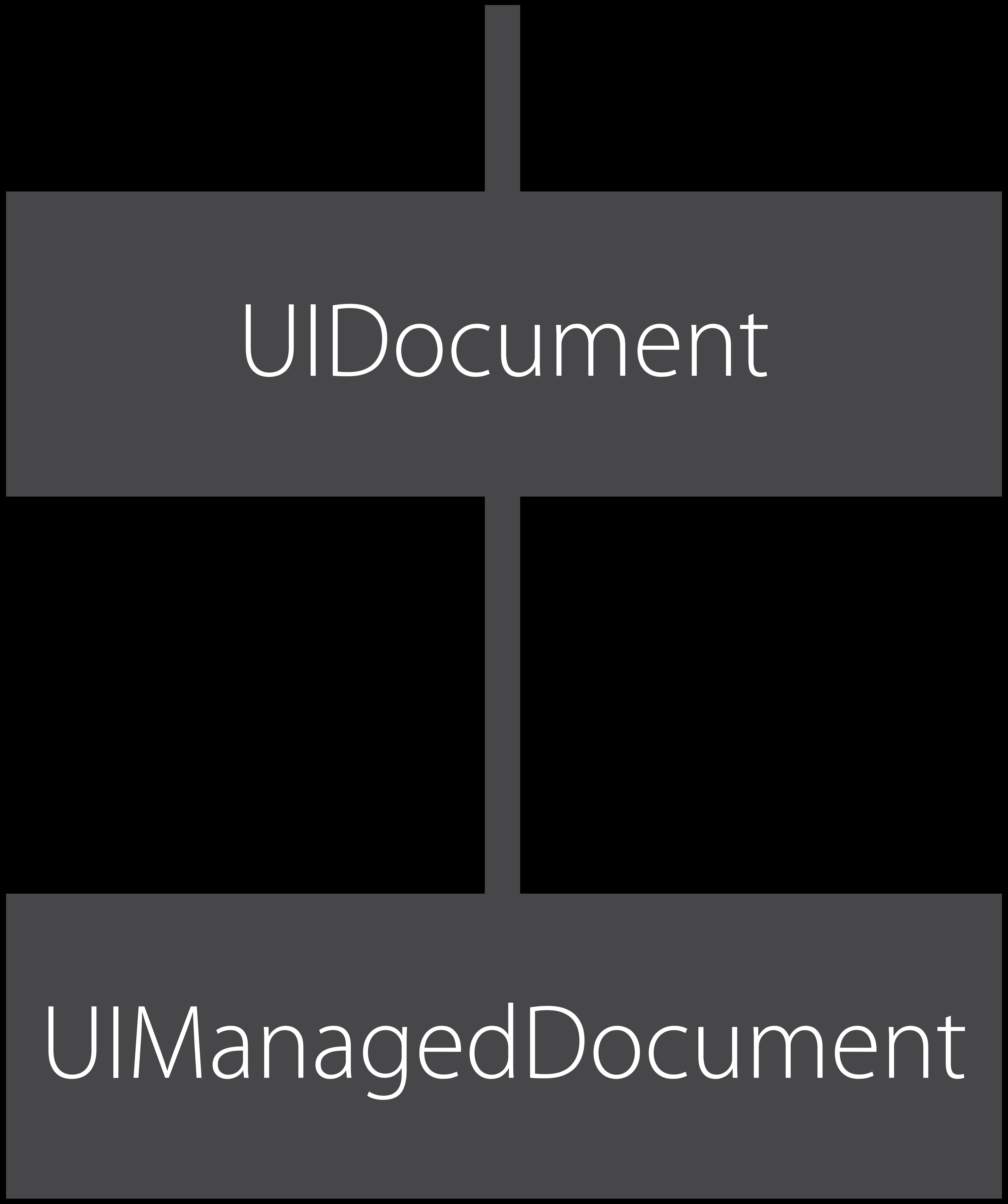
# Classes Are Awesome

## Inheritance Hierarchies



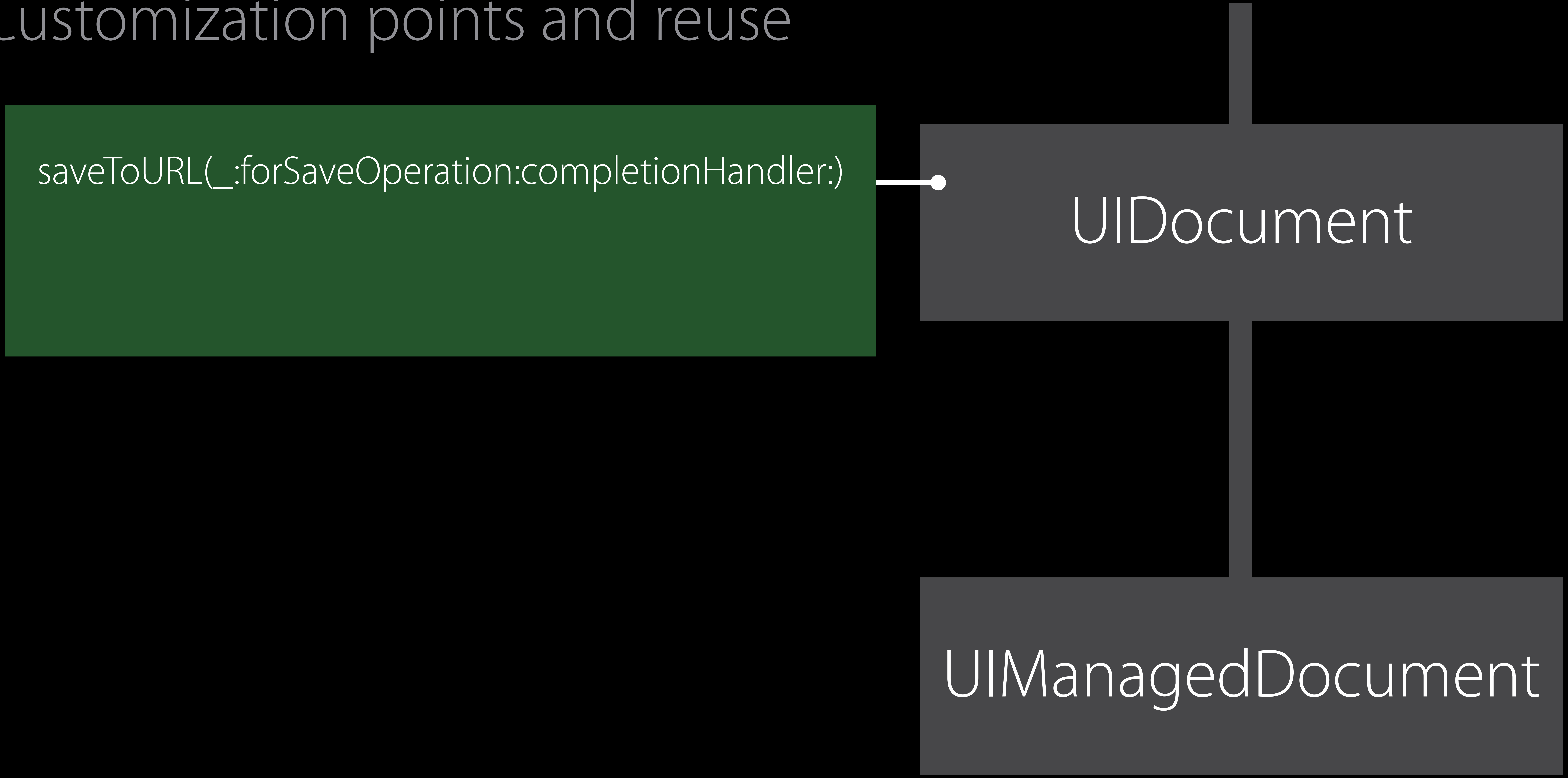
# Classes Are Awesome

Customization points and reuse



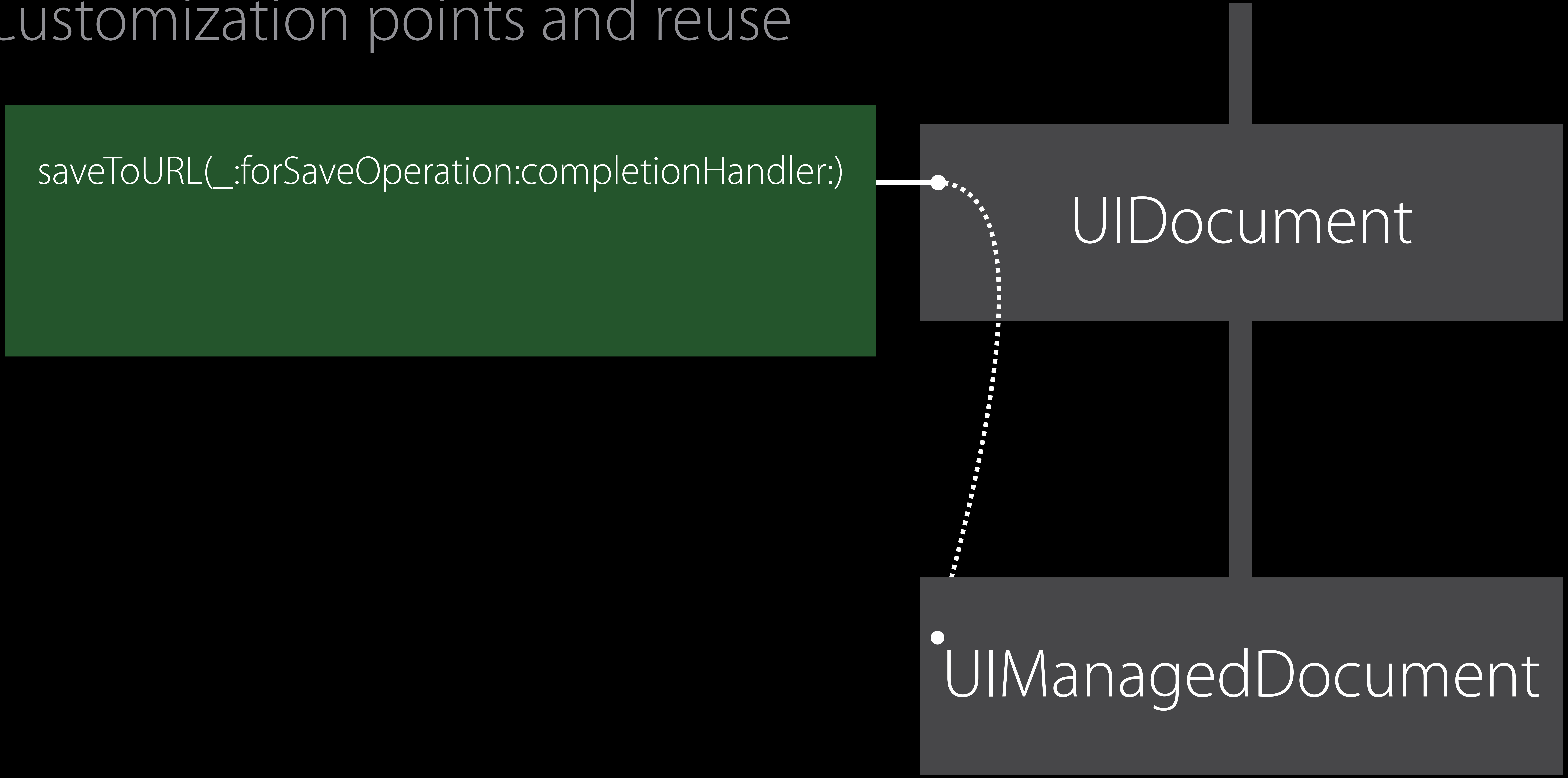
# Classes Are Awesome

Customization points and reuse



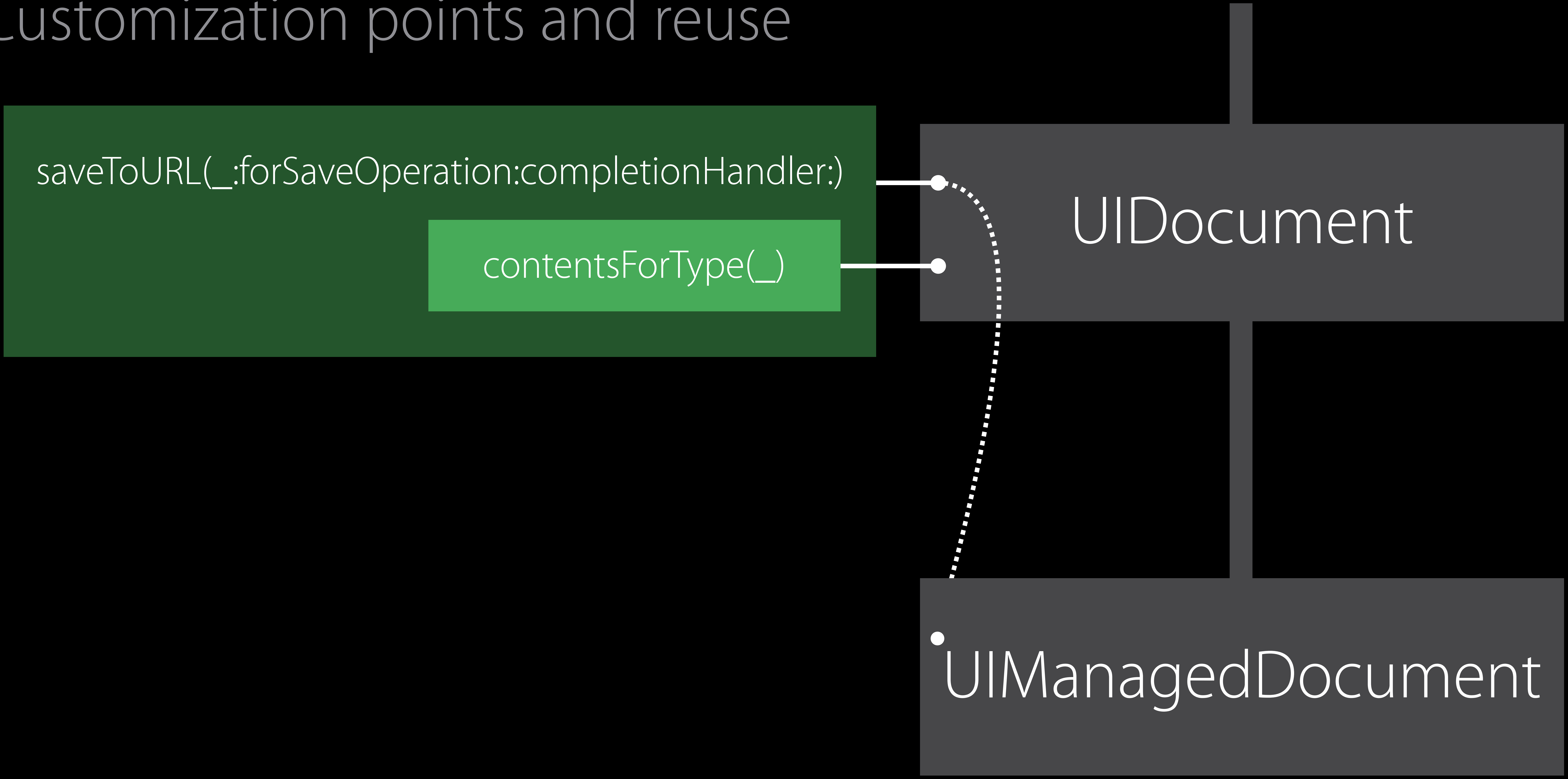
# Classes Are Awesome

Customization points and reuse



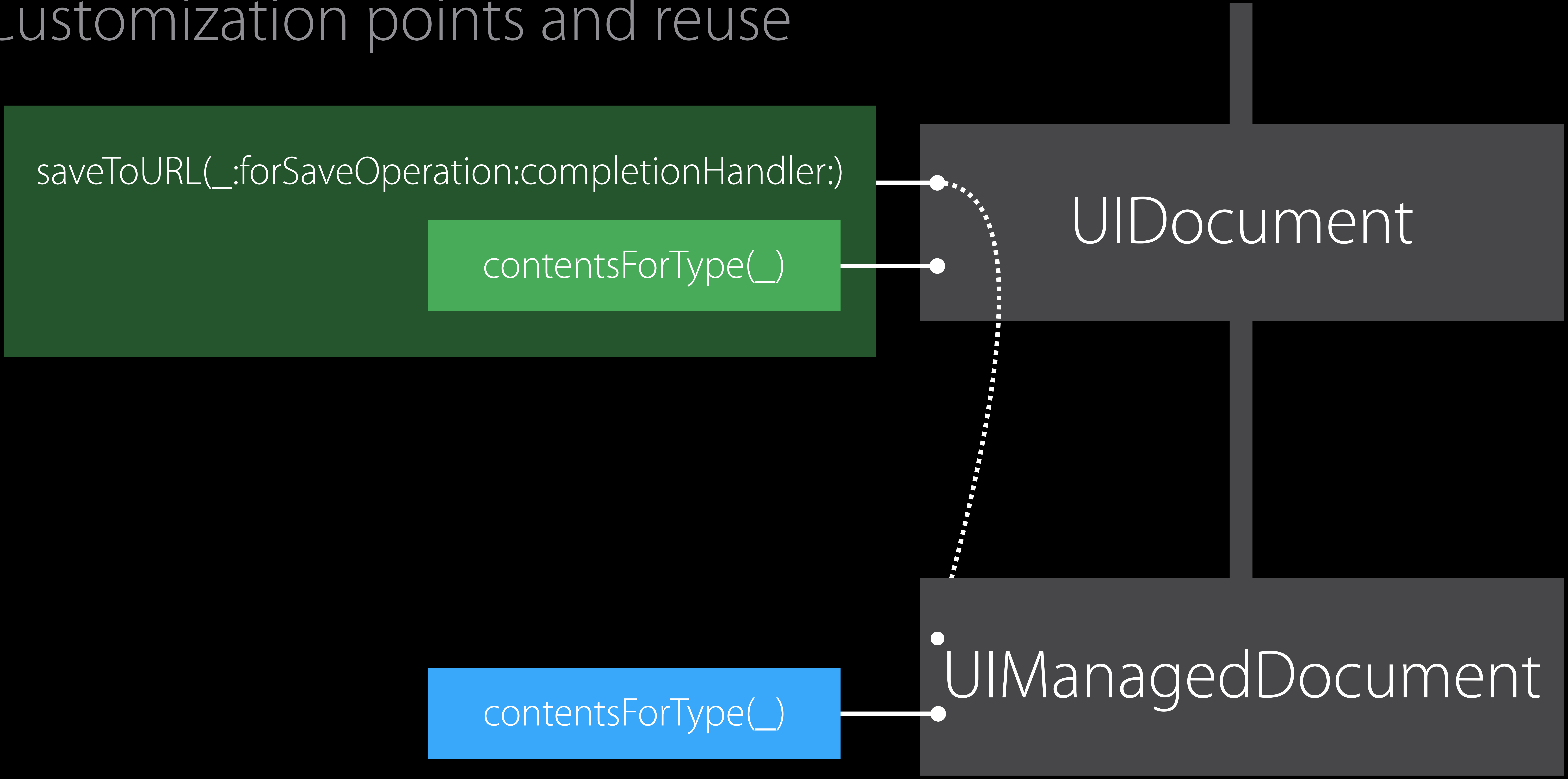
# Classes Are Awesome

## Customization points and reuse



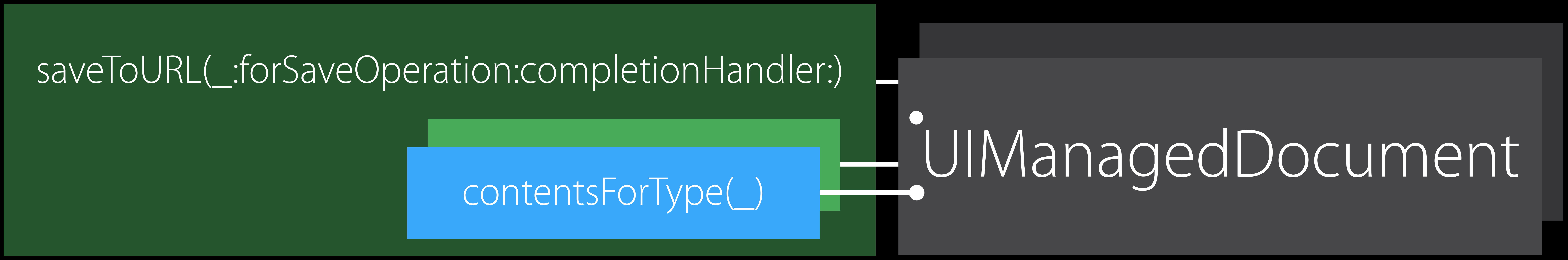
# Classes Are Awesome

## Customization points and reuse



# Classes Are Awesome

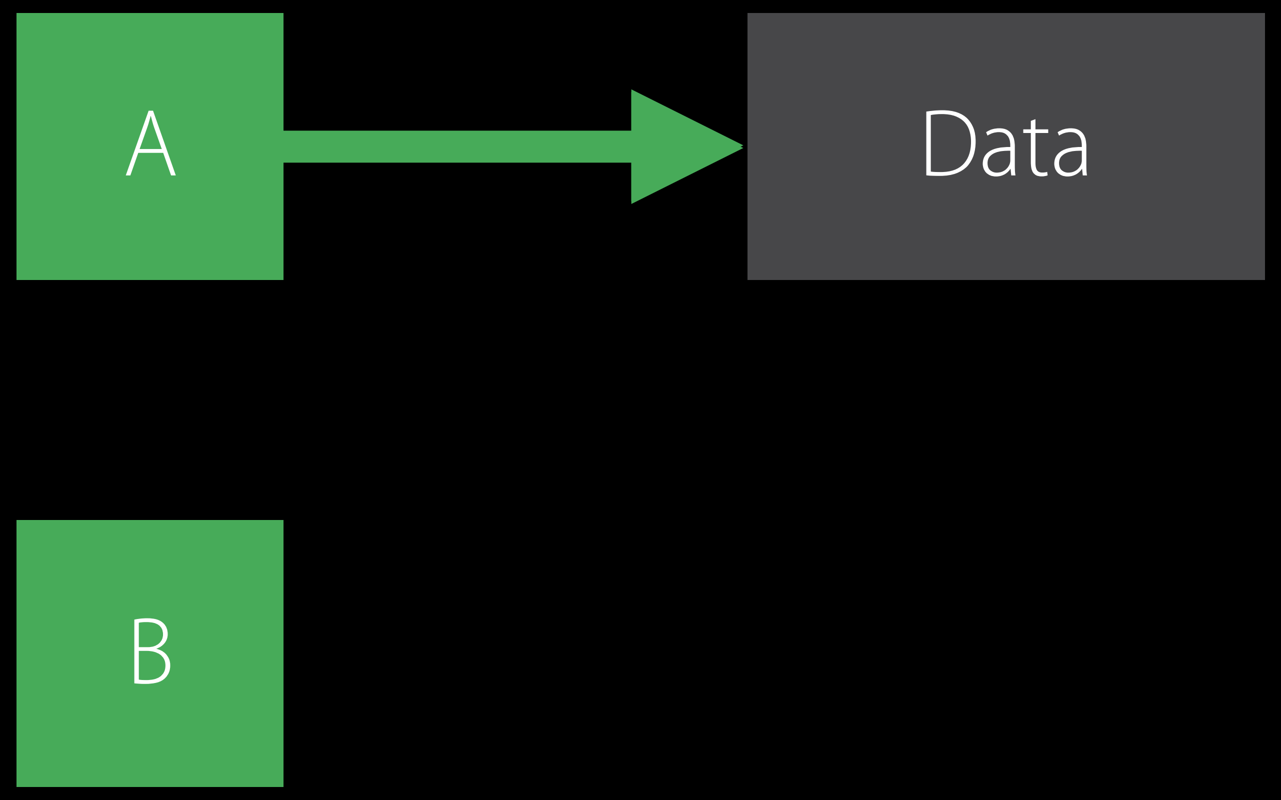
## Customization points and reuse



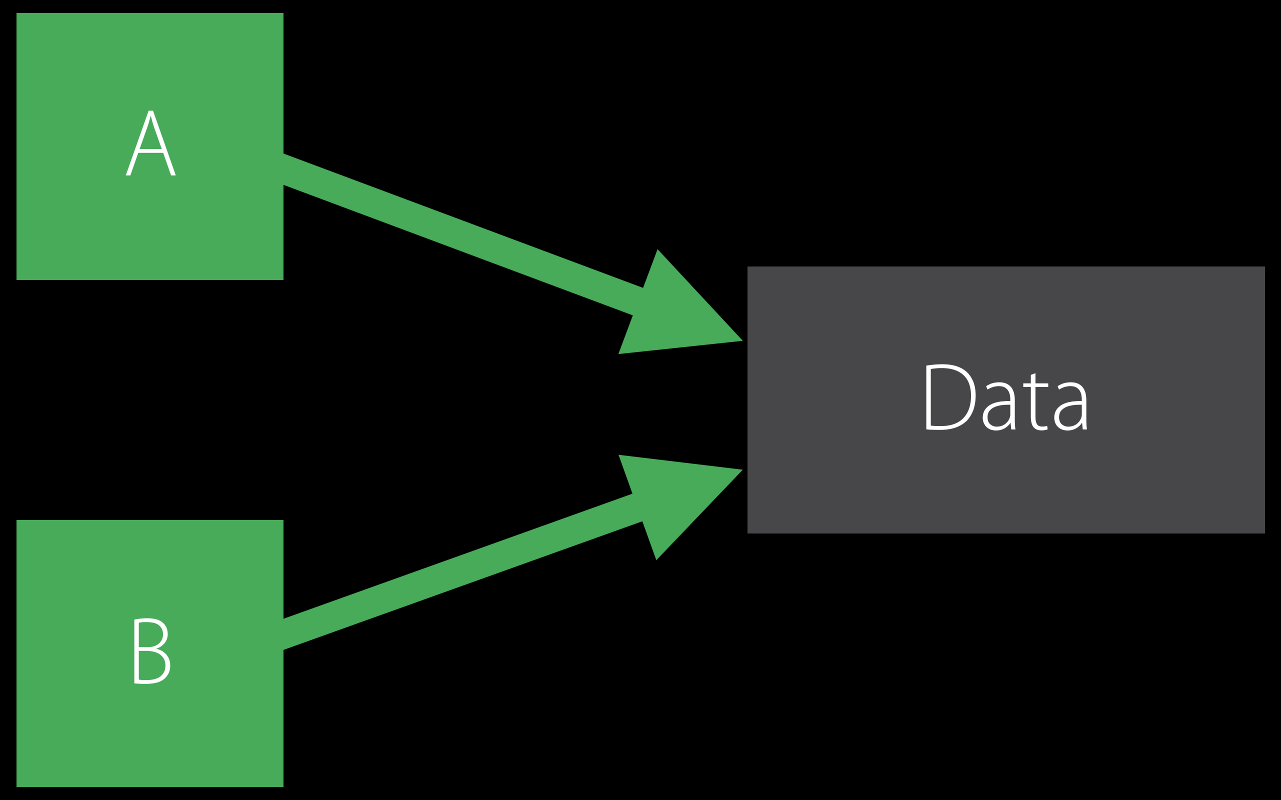
## The Three Beefs

Crusty's litany of complaints

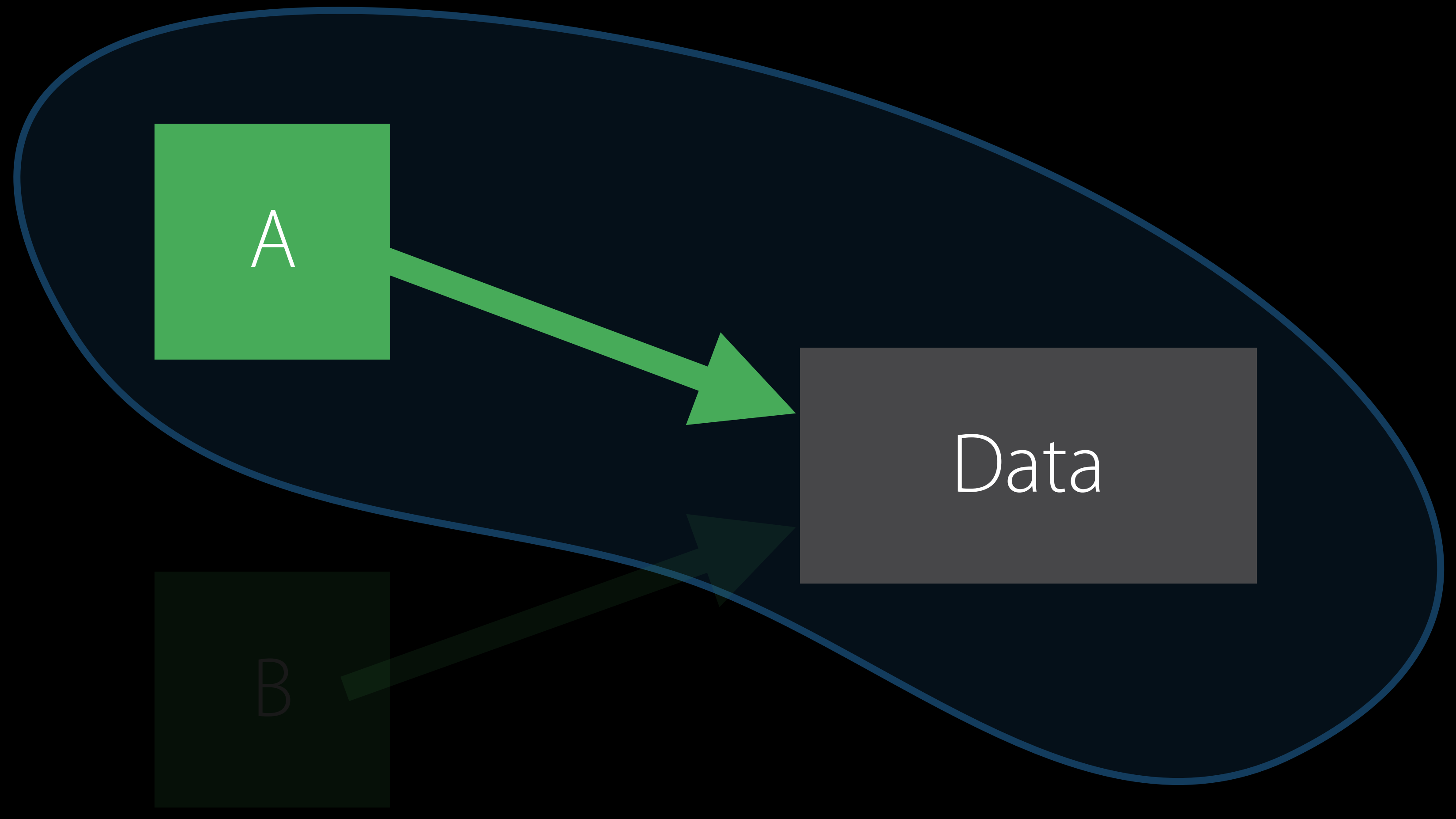
# 1. Implicit Sharing



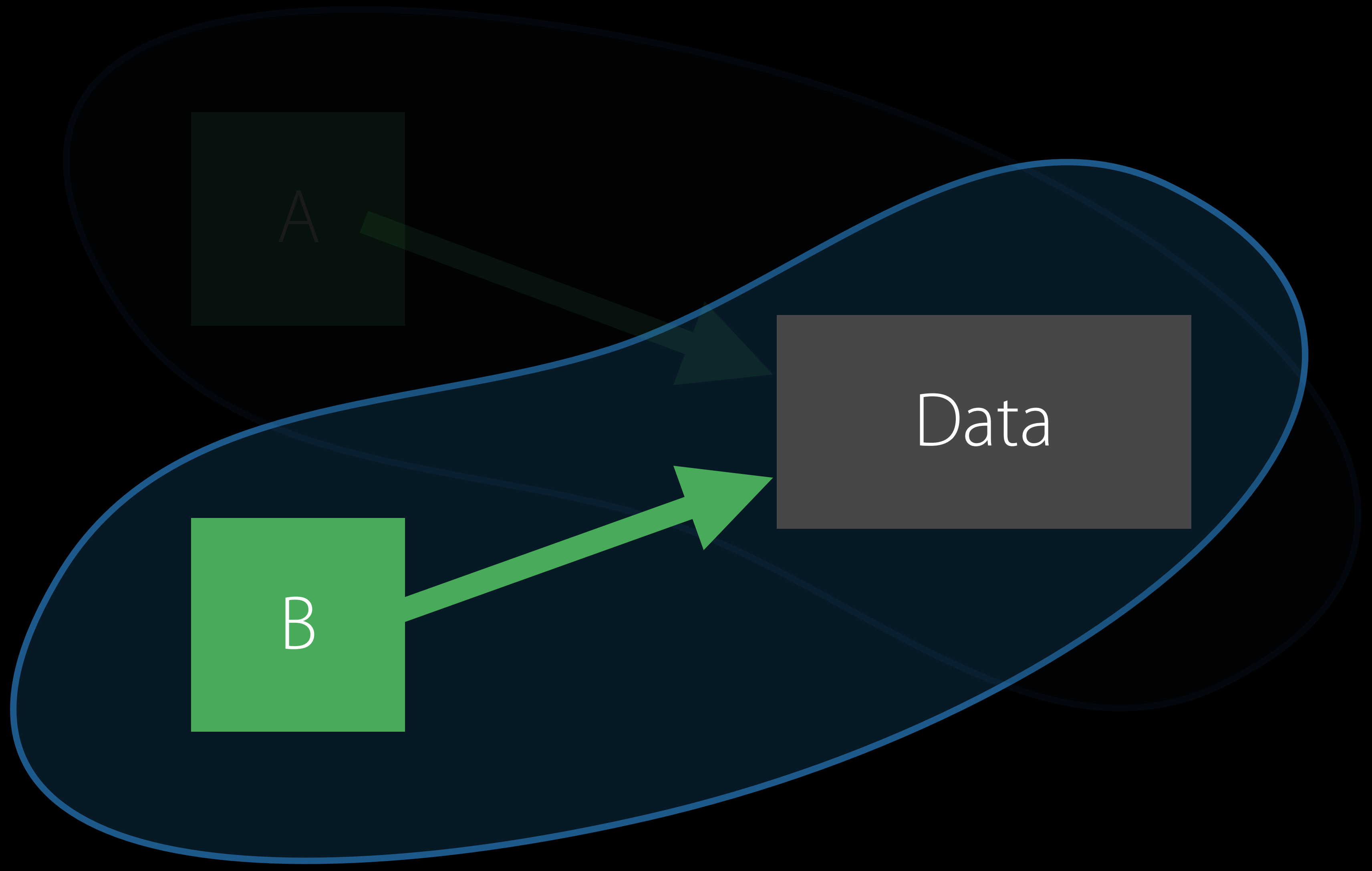
## 1. Implicit Sharing



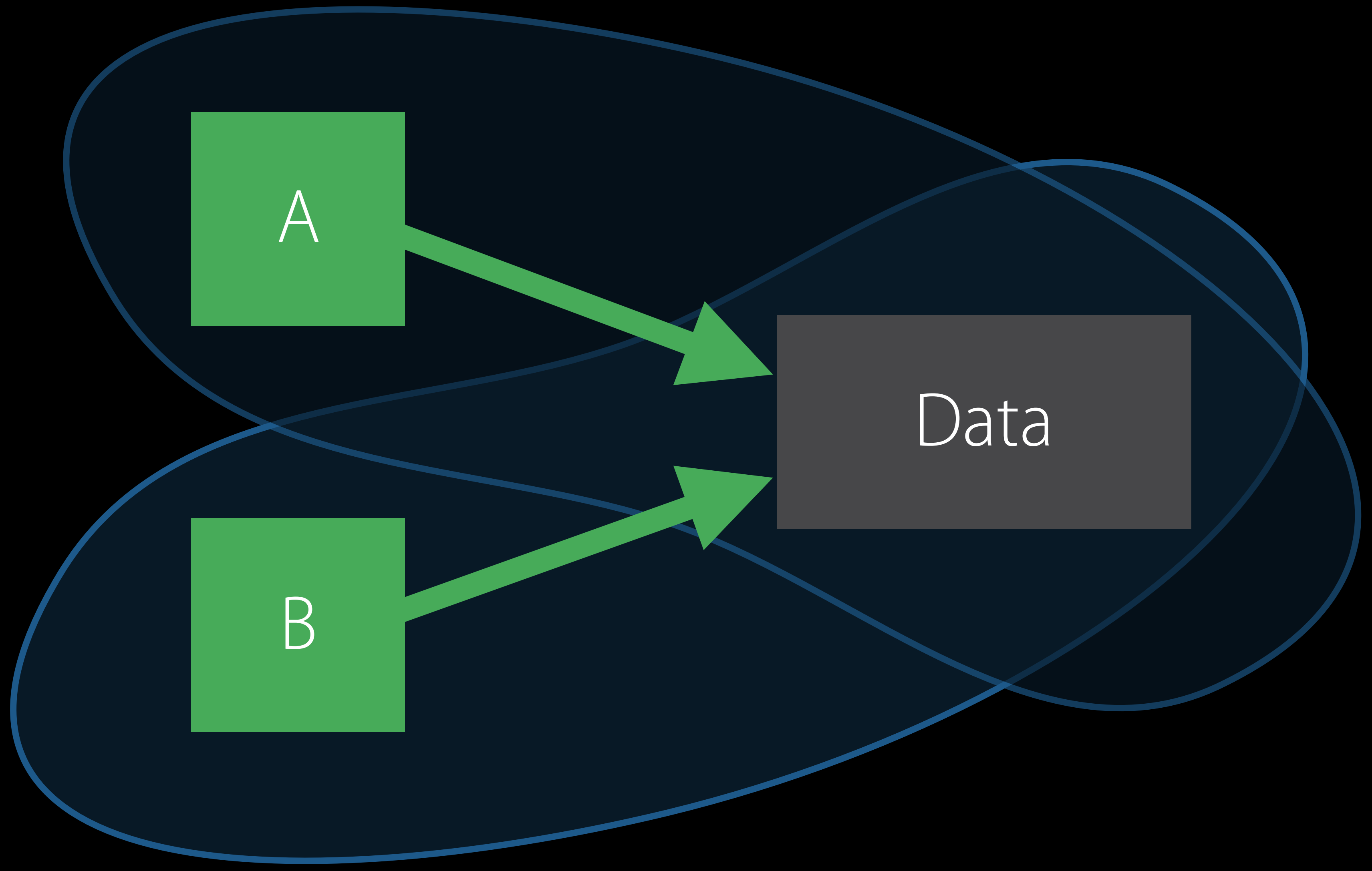
## 1. Implicit Sharing



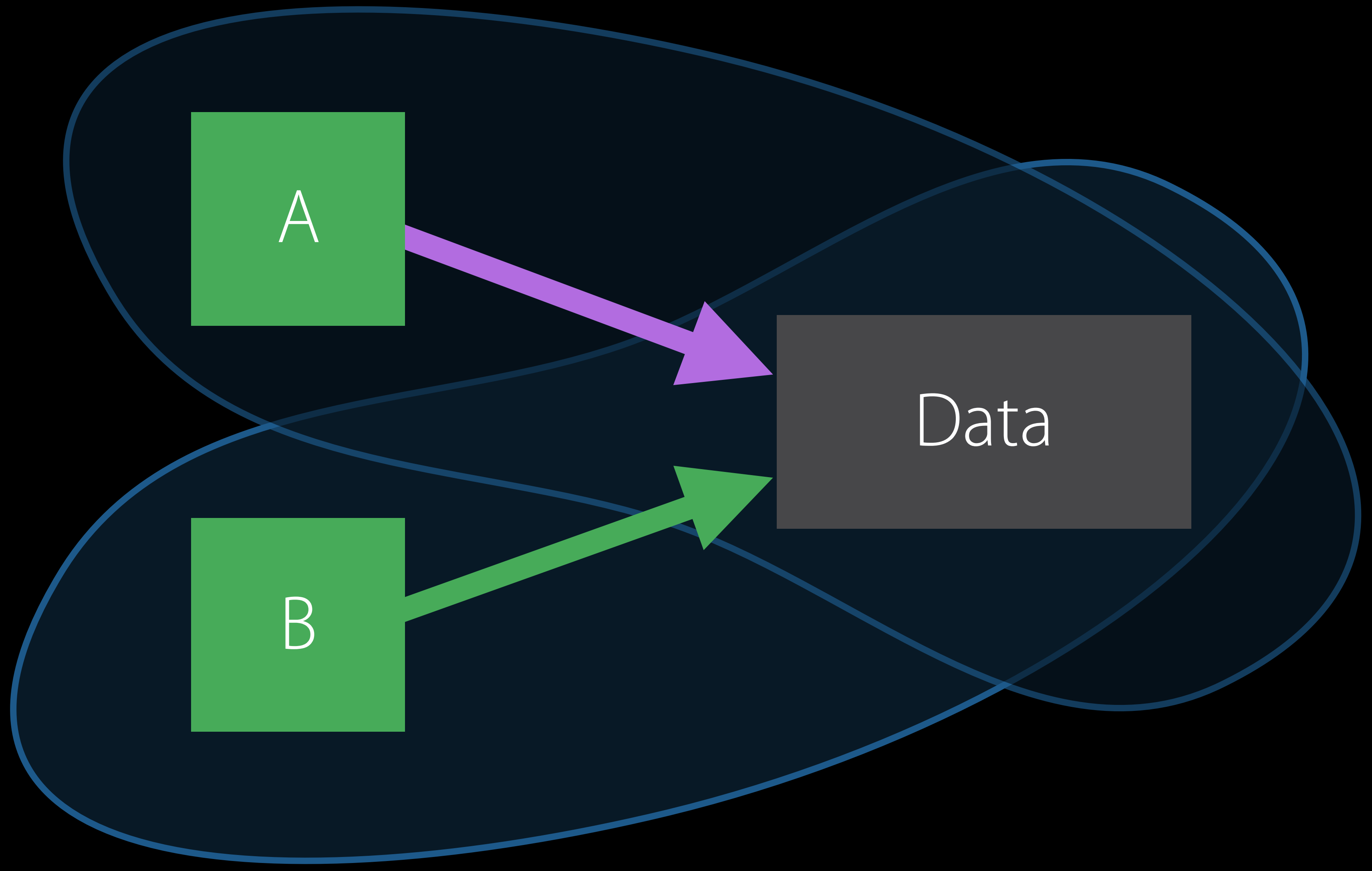
## 1. Implicit Sharing



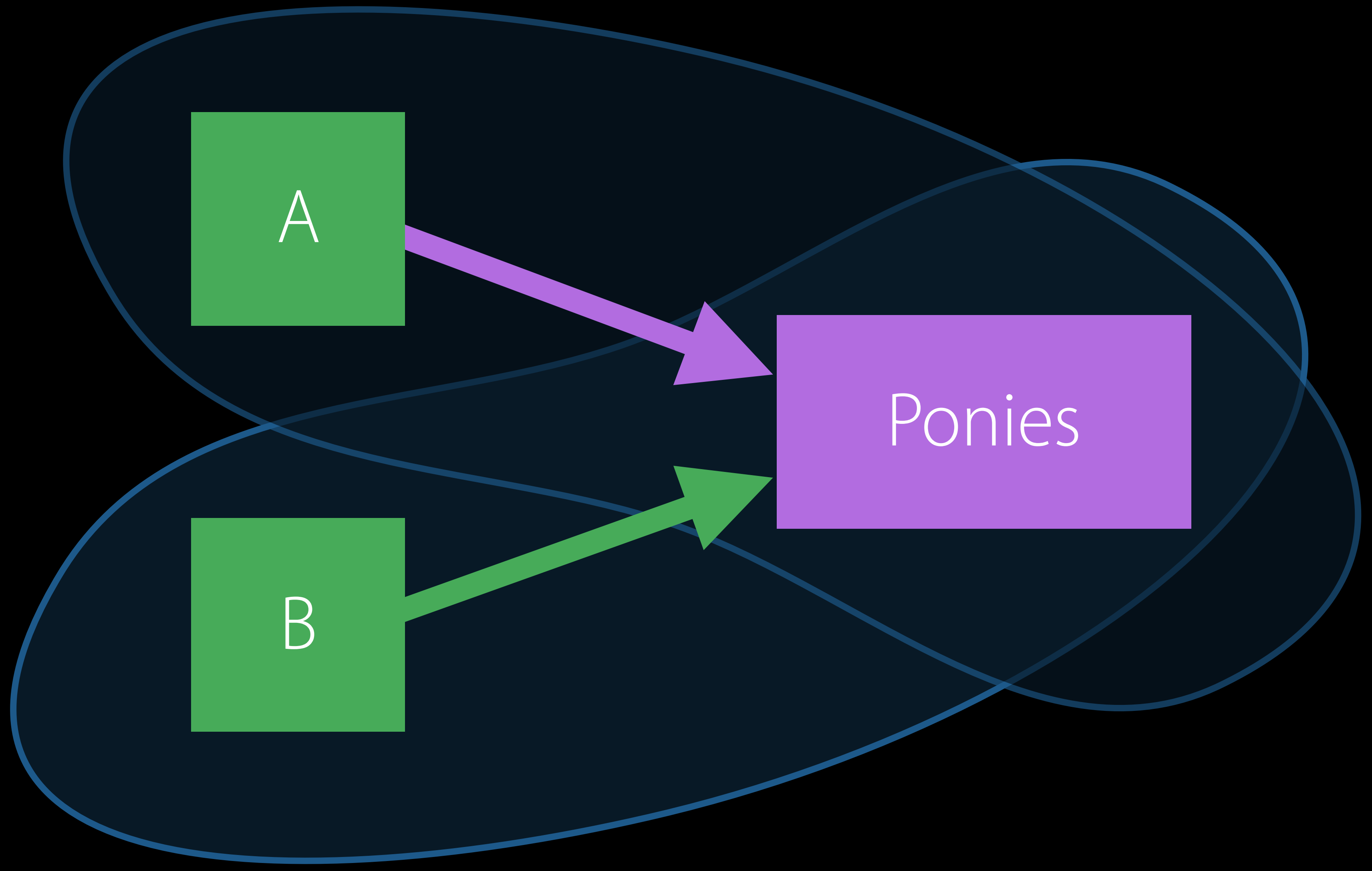
# 1. Implicit Sharing



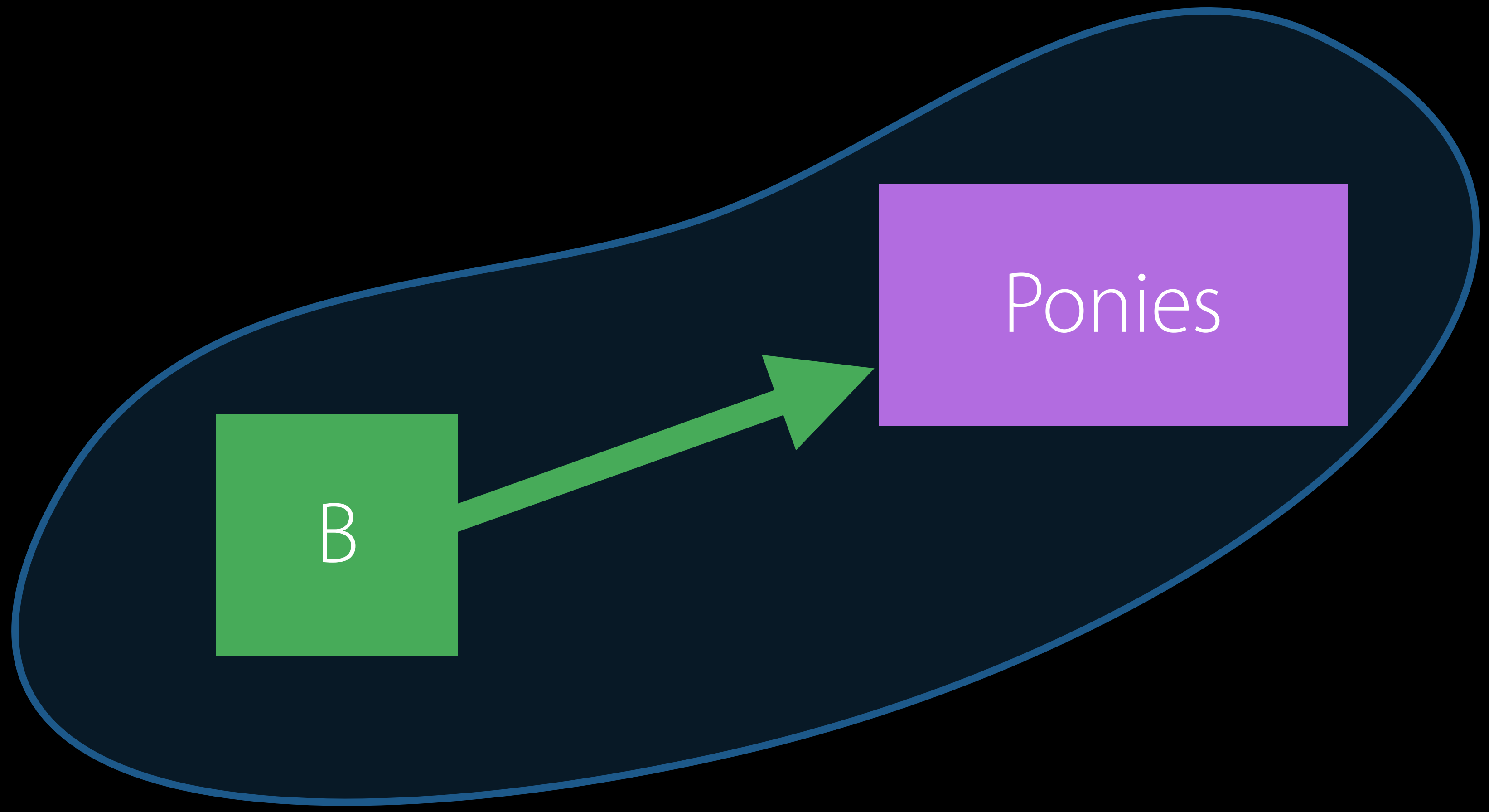
## 1. Implicit Sharing



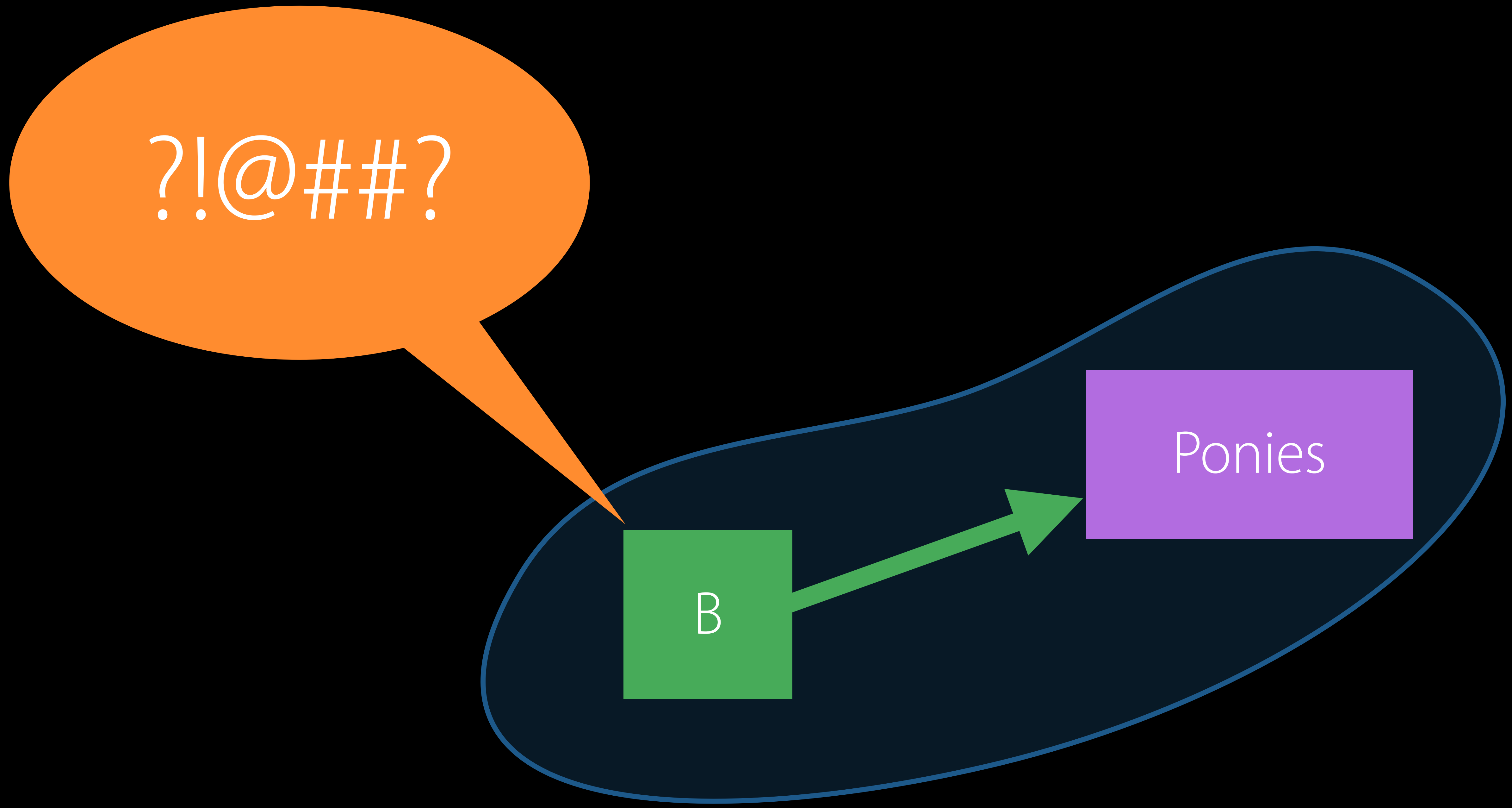
## 1. Implicit Sharing



## 1. Implicit Sharing



## 1. Implicit Sharing



## 1. Implicit Sharing

The sad story

## 1. Implicit Sharing

### The sad story

Defensive Copying

# 1. Implicit Sharing

## The sad story

Defensive Copying

Inefficiency

## 1. Implicit Sharing

### The sad story

Defensive Copying

Inefficiency

Race Conditions

## 1. Implicit Sharing

### The sad story

Defensive Copying

Inefficiency

Race Conditions

Locks

## 1. Implicit Sharing

### The sad story

Defensive Copying

Inefficiency

Race Conditions

Locks

More Inefficiency

## 1. Implicit Sharing

### The sad story

- Defensive Copying
- Inefficiency
- Race Conditions
- Locks
- More Inefficiency
- Deadlock

# 1. Implicit Sharing

## The sad story

Defensive Copying

Inefficiency

Race Conditions

Locks

More Inefficiency

Deadlock

Complexity

## 1. Implicit Sharing

### The sad story

Defensive Copying

Inefficiency

Race Conditions

Locks

More Inefficiency

Deadlock

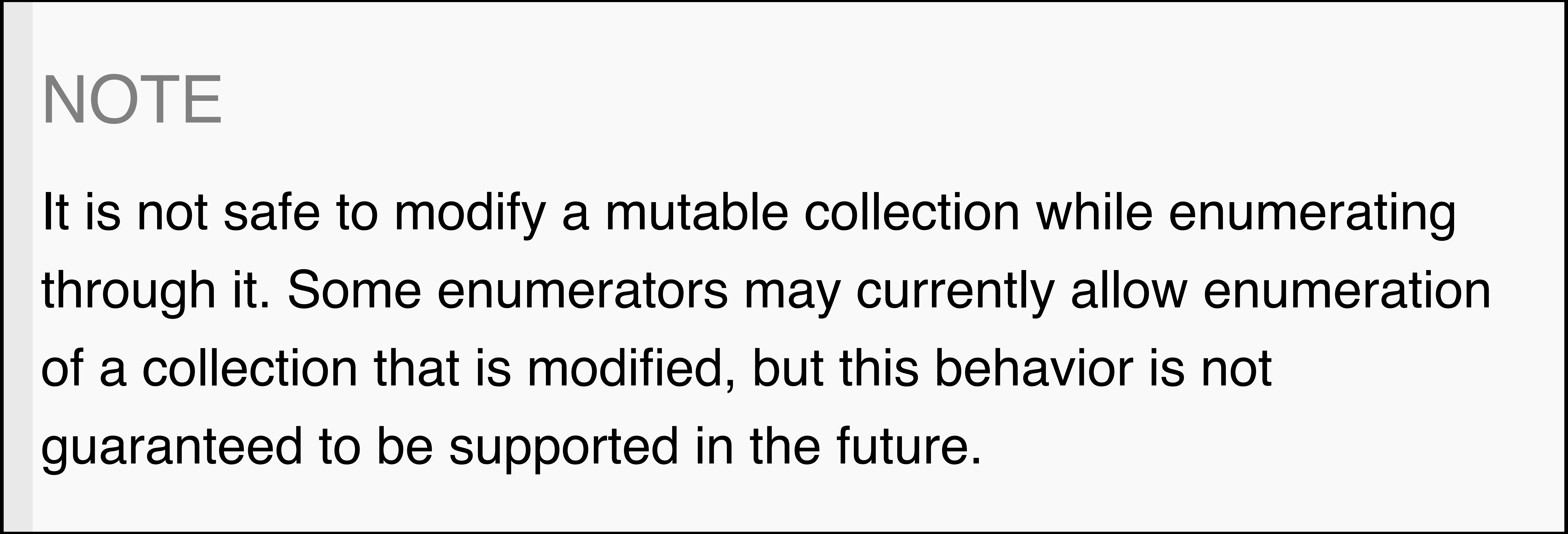
Complexity

Bugs!

## This is not news.

@property(copy), coding conventions…

## 1. Implicit Sharing

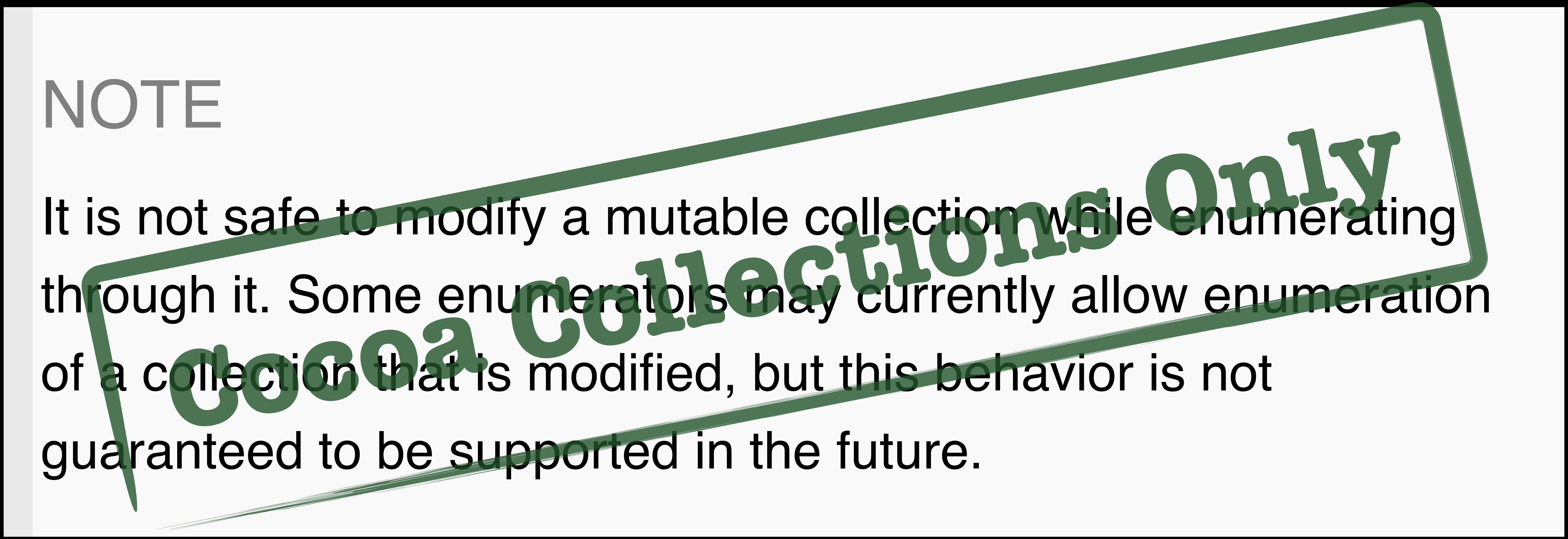


NOTE

It is not safe to modify a mutable collection while enumerating through it. Some enumerators may currently allow enumeration of a collection that is modified, but this behavior is not guaranteed to be supported in the future.

One effect of implicit sharing on Cocoa

## 1. Implicit Sharing



NOTE

It is not safe to modify a mutable collection while enumerating through it. Some enumerators may currently allow enumeration of a collection that is modified, but this behavior is not guaranteed to be supported in the future.

Cocoa Collections Only

One effect of implicit sharing on Cocoa

## Values Don't Share.

(That's a good thing).

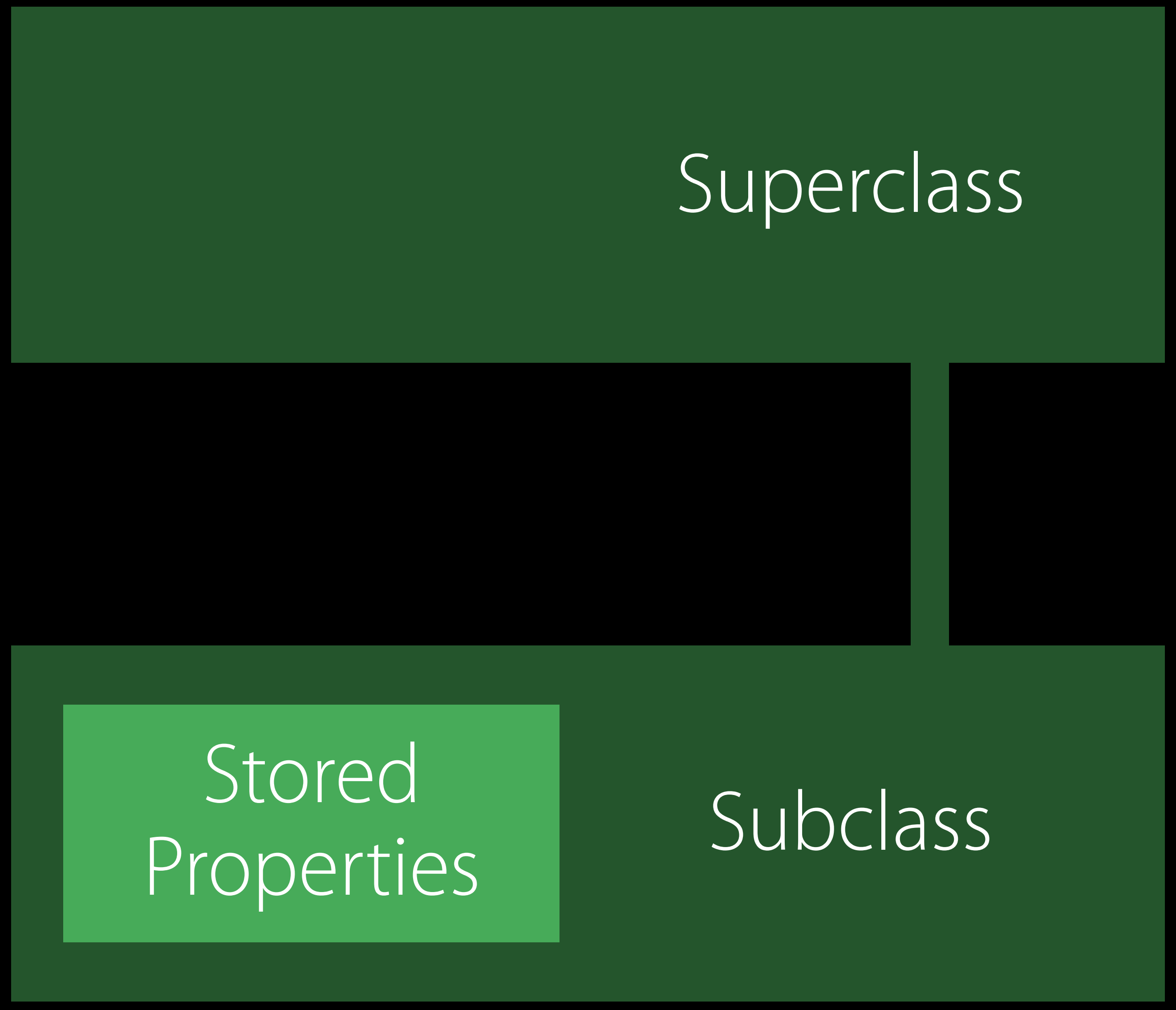
Classes? They overshare.

## 2. Inheritance All Up In Your Business

Stored Properties
Subclass

# 2. Inheritance All Up In Your Business

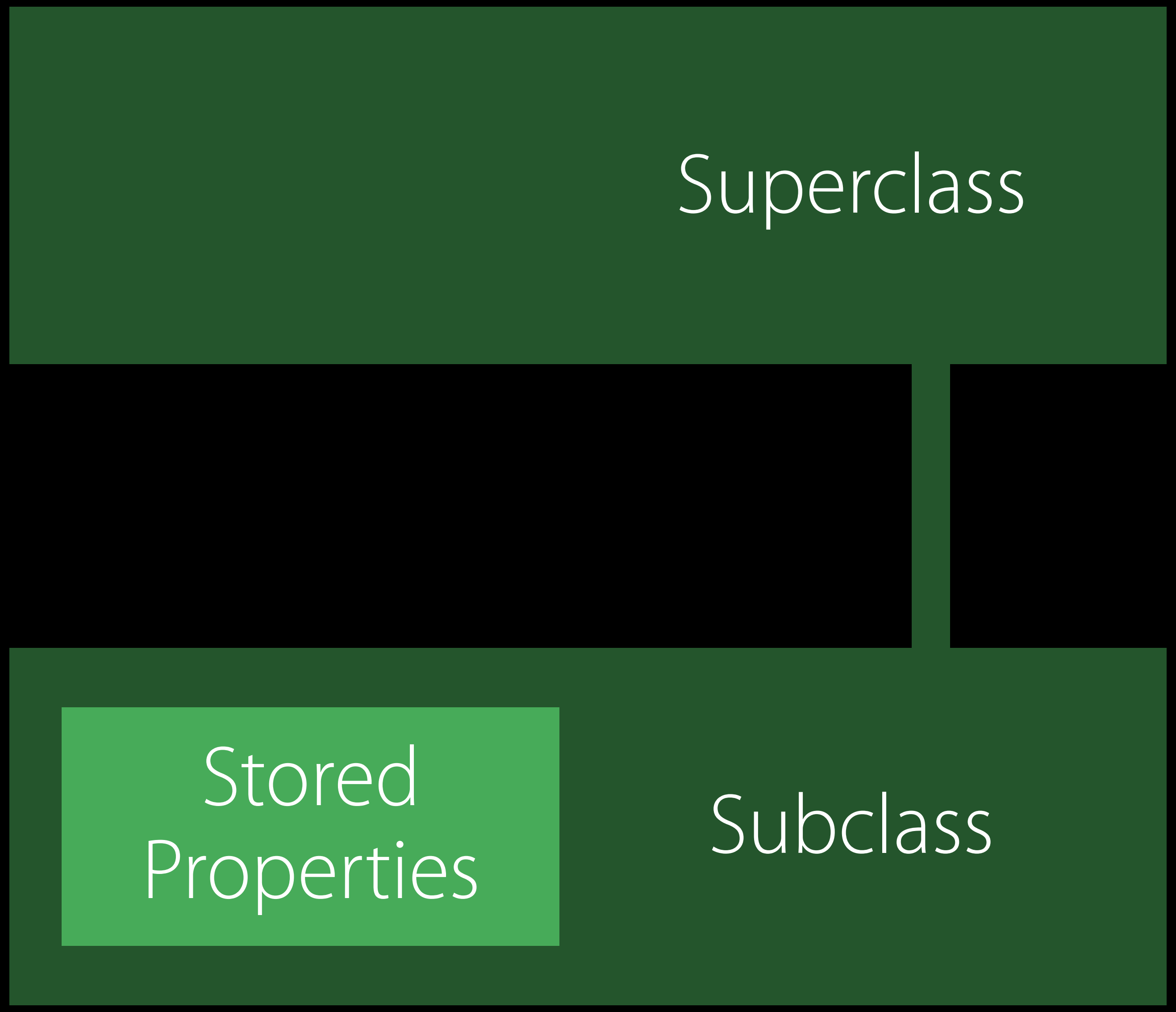
One superclass — choose well!



## 2. Inheritance All Up In Your Business

One superclass — choose well!

Single Inheritance weight gain

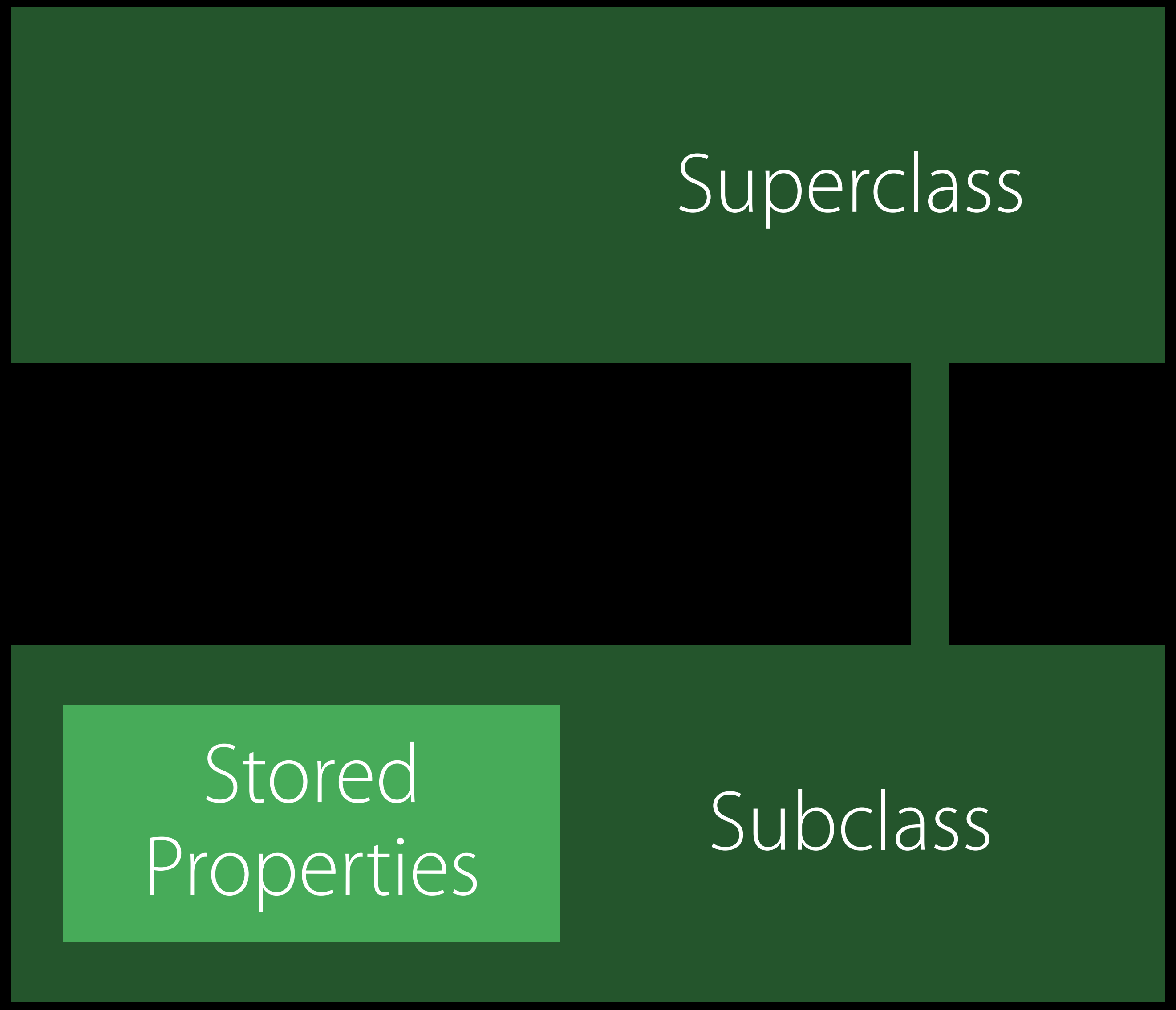


# 2. Inheritance All Up In Your Business

One superclass — choose well!

Single Inheritance weight gain

No retroactive modeling



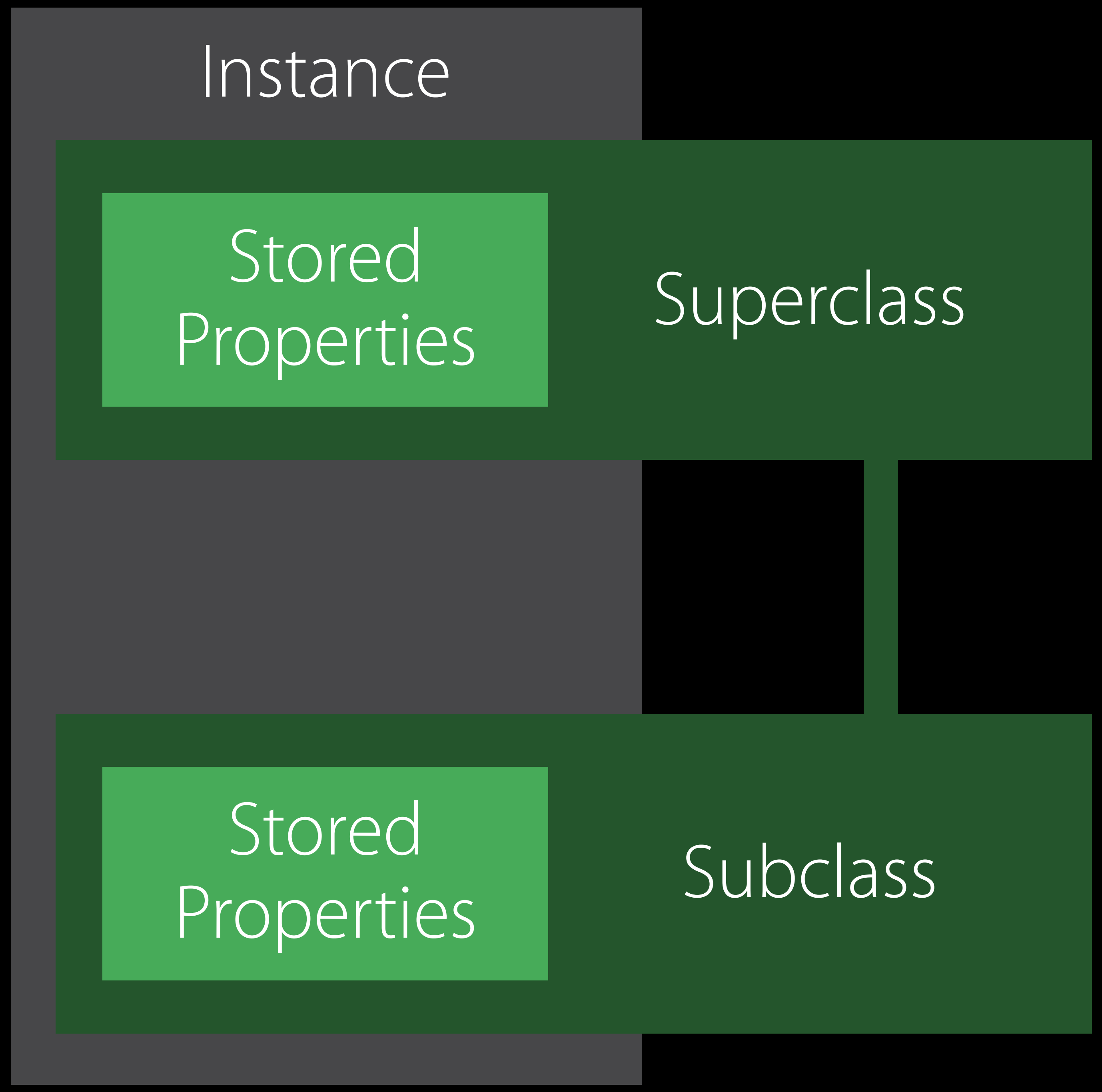
# 2. Inheritance All Up In Your Business

One superclass — choose well!

Single Inheritance weight gain

No retroactive modeling

Superclass may have stored properties



## 2. Inheritance All Up In Your Business

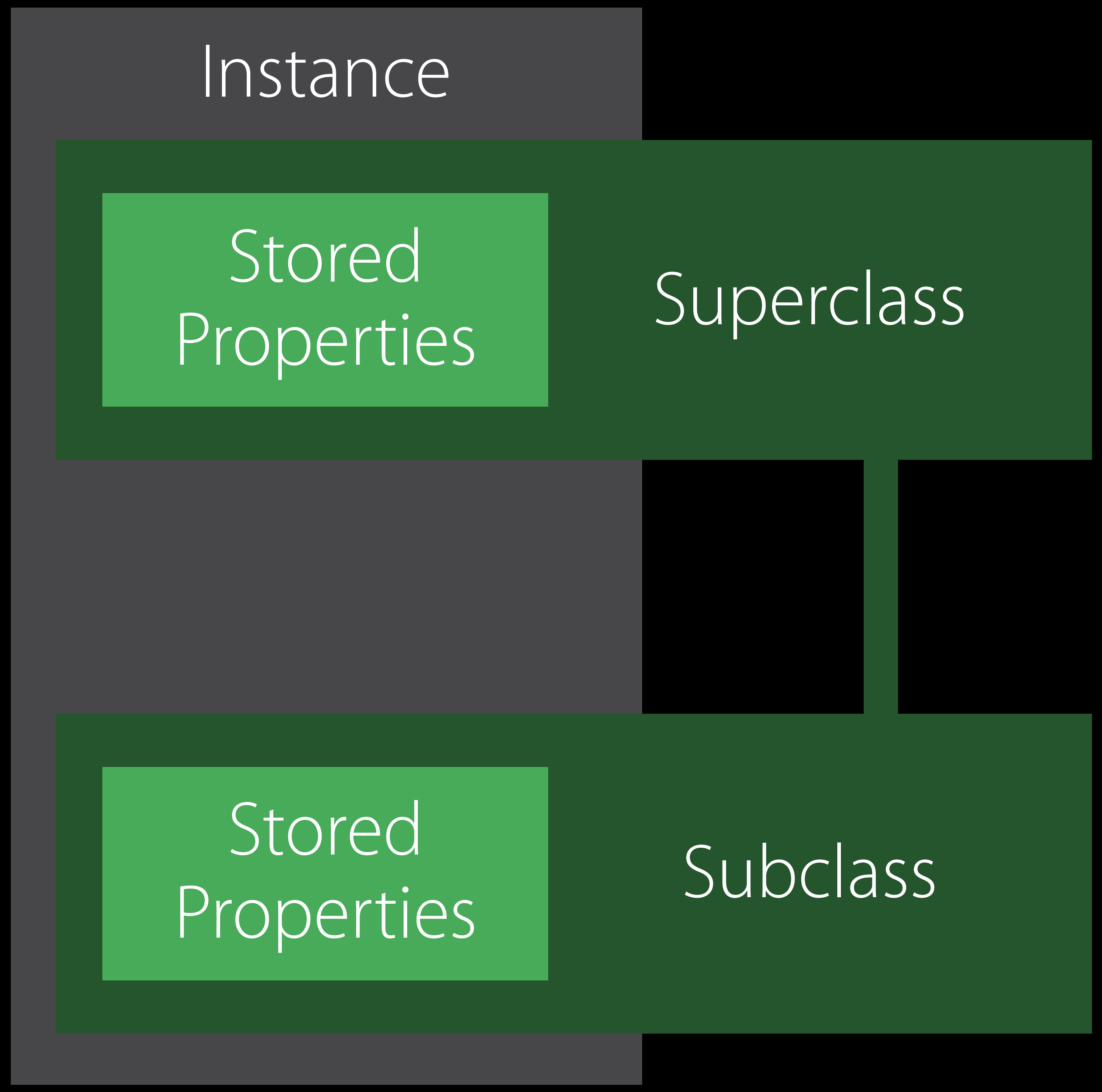
One superclass — choose well!

Single Inheritance weight gain

No retroactive modeling

Superclass may have stored properties

- You must accept them



# 2. Inheritance All Up In Your Business

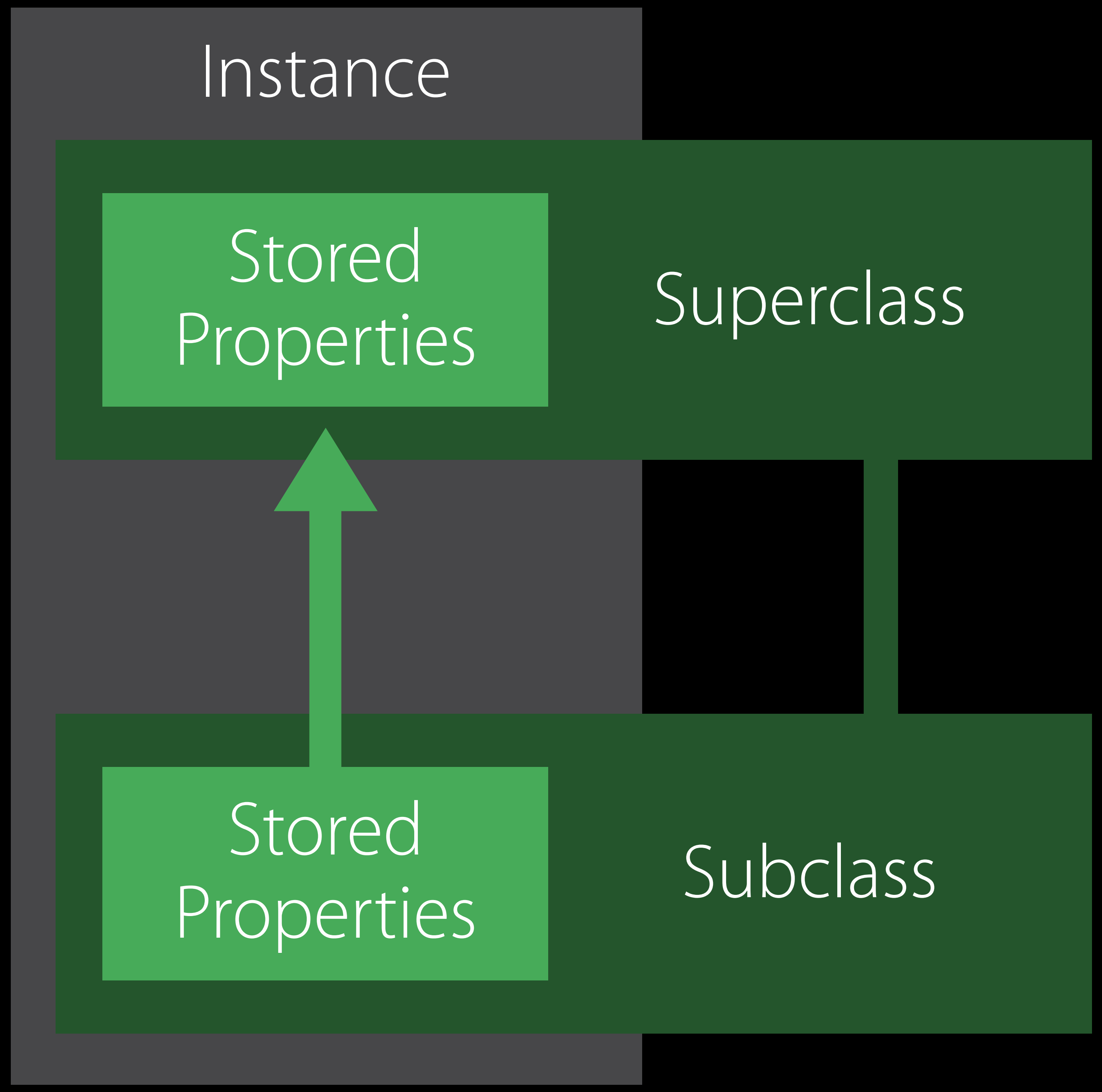
One superclass — choose well!

Single Inheritance weight gain

No retroactive modeling

Superclass may have stored properties

- You must accept them
- Initialization burden



# 2. Inheritance All Up In Your Business

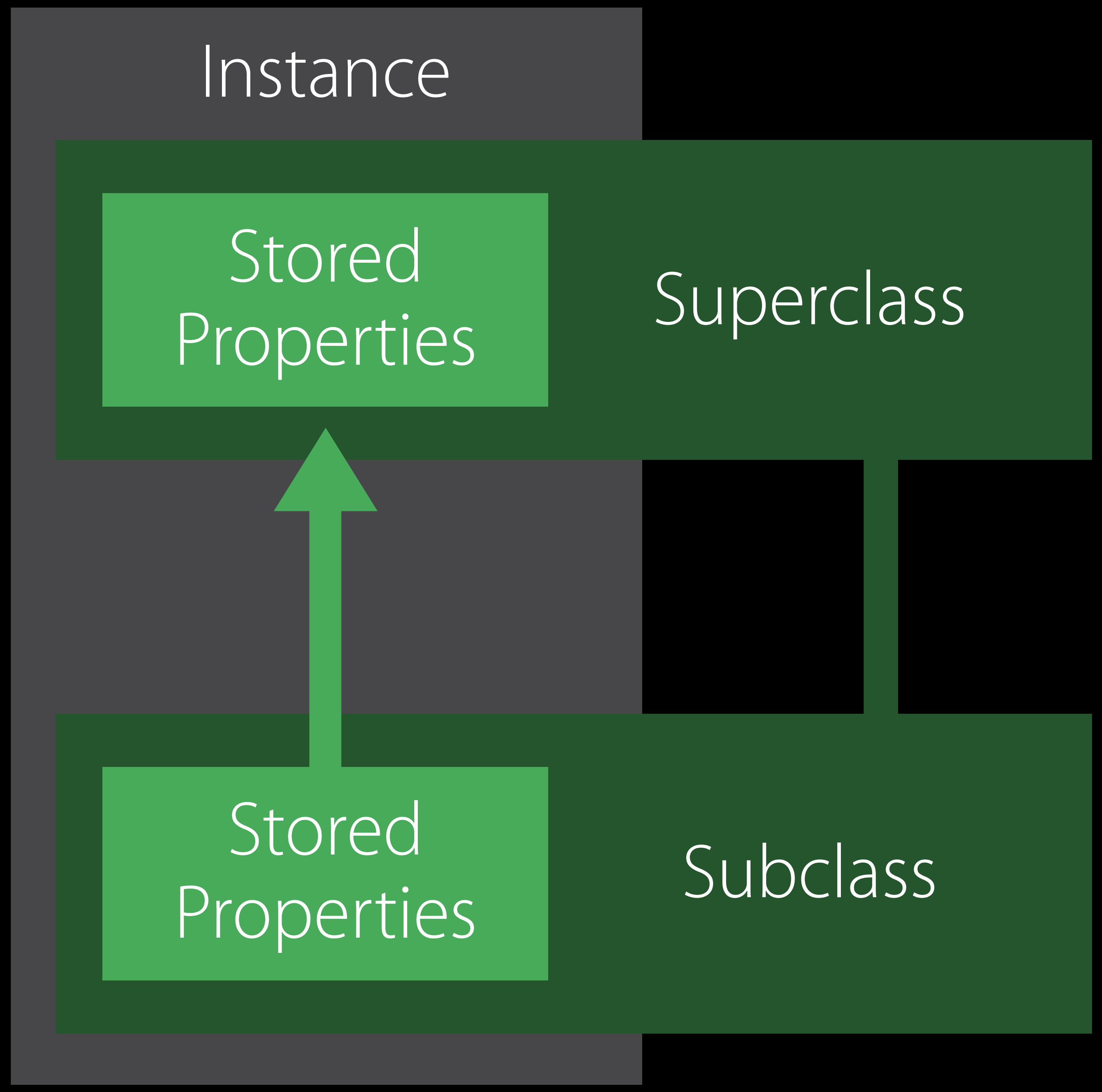
One superclass — choose well!

Single Inheritance weight gain

No retroactive modeling

Superclass may have stored properties

- You must accept them
- Initialization burden
- Don’t break superclass invariants!



## 2. Inheritance All Up In Your Business

One superclass — choose well!

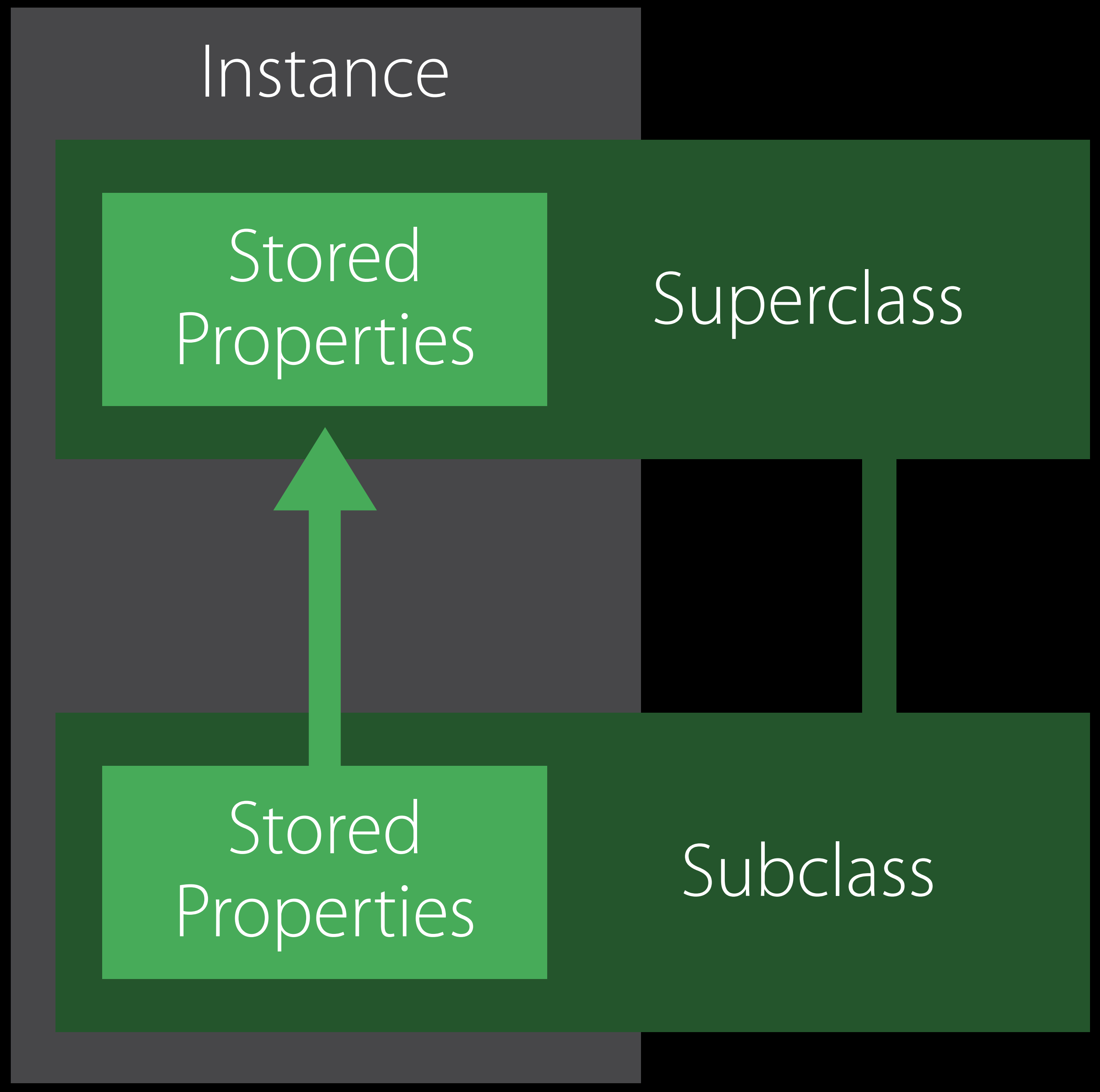
Single Inheritance weight gain

No retroactive modeling

Superclass may have stored properties

- You must accept them
- Initialization burden
- Don't break superclass invariants!

Know what/how to override (and when not to)



## This is not news.

More and more, we promote delegation.

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool
}

func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {
  var lo = 0, hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool {   }
}

func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {
  var lo = 0, hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

# 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool {   }
}

func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {
  var lo = 0, hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}
```

```
class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < other.value
  }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}
```

```
class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < other.value
  }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}
```

```
class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < other.value
  }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}
```

```
class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < other.value
  }
}
```

'Ordered' does not have a member named 'value'

# 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}


class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < other.value
  }
}
```

'Ordered' does not have a member named 'value'

# 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}

class Label : Ordered { var text: String = "" ... }

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < other.value
  }
}
```

'Ordered' does not have a member named 'value'

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}

class Label : Ordered { var text: String = "" ... }

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < other.value
  }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}

class Label : Ordered { var text: String = "" ... }

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < (other as! Number).value
  }
}
```

## 3. Lost Type Relationships

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}

class Label : Ordered { var text: String = "" ... }

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return value < (other as! Number).value
  }
}
```

```
as! ASubclass
```

A sign that a type relationship was lost

Usually due to using classes for abstraction

## A Better Abstraction Mechanism

Supports value types (and classes)

Supports static type relationships (and dynamic dispatch)

Non-monolithic

Supports retroactive modeling

Doesn’t impose instance data on models

Doesn’t impose initialization burdens on models

Makes clear what to implement

## Swift Is a Protocol-Oriented Programming Language

## Start with a Protocol

Your first stop for new abstractions

## Starting Over with Protocols

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

## Starting Over with Protocols

```
class Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

## Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

## Starting Over with Protocols

error: protocol methods may not have bodies

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool { fatalError("implement me!") }
}

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

## Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

# Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value

}
```

error: method does not override any method from its superclass

# Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

class Number : Ordered {
  var value: Double = 0
  override func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

## Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

## Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

## Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Ordered) -> Bool {
    return self.value < (other as! Number).value
  }
}
```

## Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Number) -> Bool {
    return self.value < other.value
  }
}
```

# Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Number) -> Bool {
    return self.value < other.value
  }
}
```

protocol requires function 'precedes' with type '(Ordered) -> Bool'
candidate has non-matching type '(Number) -> Bool'

# Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Ordered) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Number) -> Bool {
    return self.value < other.value
  }
}
```

## Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Number) -> Bool {
    return self.value < other.value
  }
}
```

# Starting Over with Protocols

“Self” requirement

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Number) -> Bool {
    return self.value < other.value
  }
}
```

## Starting Over with Protocols

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}

struct Number : Ordered {
  var value: Double = 0
  func precedes(other: Number) -> Bool {
    return self.value < other.value
  }
}
```

## Using Our Protocol

```
func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {
  var lo = 0
  var hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

## Using Our Protocol

```
func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {
  var lo = 0
  var hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

# Using Our Protocol

```
func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {
  var lo = 0
  var hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

protocol 'Ordered' can only be used as a generic constraint because it has Self or associated type requirements

## Using Our Protocol

```
func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {
  var lo = 0
  var hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

## Using Our Protocol

```
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int {
  var lo = 0
  var hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

## Using Our Protocol

```
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int {
  var lo = 0
  var hi = sortedKeys.count
  while hi > lo {
    let mid = lo + (hi - lo) / 2
    if sortedKeys[mid].precedes(k) { lo = mid + 1 }
    else { hi = mid }
  }
  return lo
}
```

## Two Worlds of Protocols

| Without Self Requirement | With Self Requirement |
|---|---|
| `func precedes(other: Ordered) -> Bool` | `func precedes(other: Self) -> Bool` |

## Two Worlds of Protocols

| Without Self Requirement | With Self Requirement |
|---|---|
| `func precedes(other: Ordered) -> Bool` | `func precedes(other: Self) -> Bool` |
| Usable as a type | Only usable as a generic constraint |
| `func sort(inout a: [Ordered])` | `func sort<T : Ordered>(inout a: [T])` |

# Two Worlds of Protocols

| Without Self Requirement | With Self Requirement |
|---|---|
| `func precedes(other: Ordered) -> Bool` | `func precedes(other: Self) -> Bool` |
| Usable as a type | Only usable as a generic constraint |
| `func sort(inout a: [Ordered])` | `func sort<T : Ordered>(inout a: [T])` |
| Think "heterogeneous" | Think "homogeneous" |
| 多种多样的；混杂的 | 同性质的, 同类的 |

## Two Worlds of Protocols

| Without Self Requirement | With Self Requirement |
|---|---|
| `func precedes(other: Ordered) -> Bool` | `func precedes(other: Self) -> Bool` |
| Usable as a type<br>`func sort(inout a: [Ordered])` | Only usable as a generic constraint<br>`func sort<T : Ordered>(inout a: [T])` |
| Think "heterogeneous" | Think "homogeneous" |
| Every model must deal with all others | Models are free from interaction |

# Two Worlds of Protocols

| Without Self Requirement | With Self Requirement |
|---|---|
| `func precedes(other: Ordered) -> Bool` | `func precedes(other: Self) -> Bool` |
| Usable as a type<br>`func sort(inout a: [Ordered])` | Only usable as a generic constraint<br>`func sort<T : Ordered>(inout a: [T])` |
| Think "heterogeneous" | Think "homogeneous" |
| Every model must deal with all others | Models are free from interaction |
| Dynamic dispatch | Static dispatch |

# Two Worlds of Protocols

| Without Self Requirement | With Self Requirement |
|---|---|
| `func precedes(other: Ordered) -> Bool` | `func precedes(other: Self) -> Bool` |
| Usable as a type<br>`func sort(inout a: [Ordered])` | Only usable as a generic constraint<br>`func sort<T : Ordered>(inout a: [T])` |
| Think "heterogeneous" | Think "homogeneous" |
| Every model must deal with all others | Models are free from interaction |
| Dynamic dispatch | Static dispatch |
| Less optimizable | More optimizable |

## A Challenge for Crusty

Prove it!

# A Primitive “Renderer”

```
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }

  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }

  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
      print("arcAt(\(center), radius: \(radius),"
        + " startAngle: \(startAngle), endAngle: \(endAngle))")
  }
}
```

## Drawable

```
protocol Drawable {
  func draw(renderer: Renderer)
}
```

# Polygon

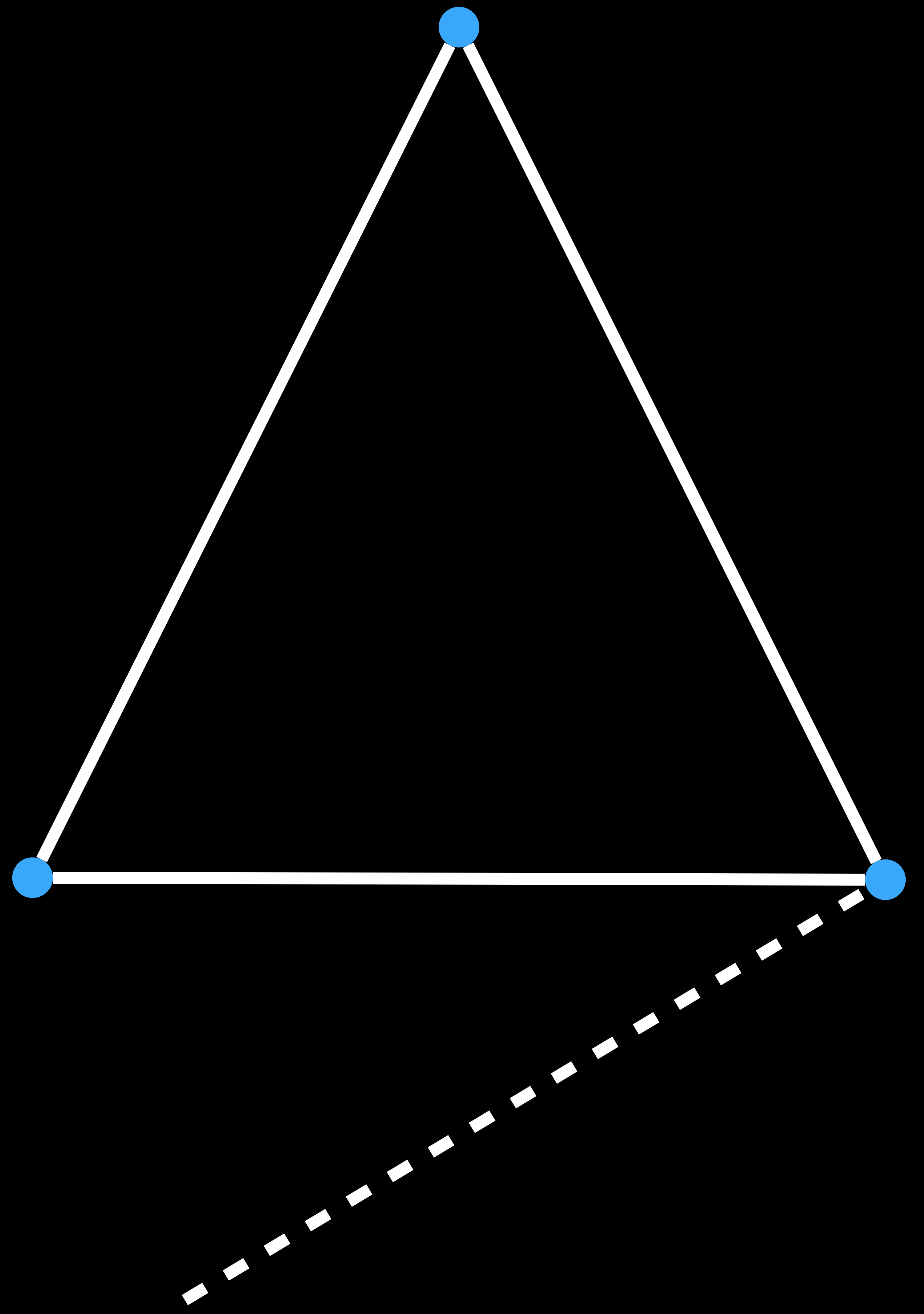
```
protocol Drawable {
  func draw(renderer: Renderer)
}

struct Polygon : Drawable {
  func draw(renderer: Renderer) {
    renderer.moveTo(corners.last!)
    for p in corners {
      renderer.lineTo(p)
    }
  }
  var corners: [CGPoint] = []
}
```

## Polygon

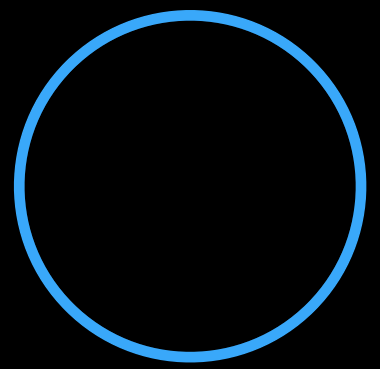
```
protocol Drawable {
  func draw(renderer: Renderer)
}

struct Polygon : Drawable {
  func draw(renderer: Renderer) {
    renderer.moveTo(corners.last!)
    for p in corners {
      renderer.lineTo(p)
    }
  }
  var corners: [CGPoint] = []
}
```

## Polygon

```
protocol Drawable {
  func draw(renderer: Renderer)
}

struct Polygon : Drawable {
  func draw(renderer: Renderer) {
    renderer.moveTo(corners.last!)
    for p in corners {
      renderer.lineTo(p)
    }
  }
  var corners: [CGPoint] = []
}
```

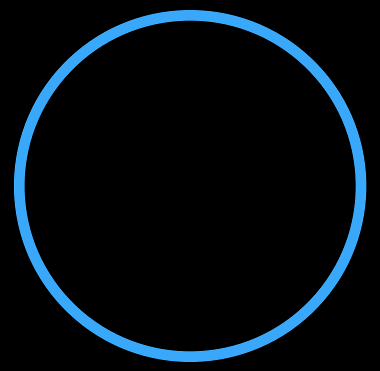


# Circle



```
protocol Drawable {
  func draw(renderer: Renderer)
}

struct Circle : Drawable {
  func draw(renderer: Renderer) {
    renderer.arcAt(center, radius: radius,
      startAngle: 0.0, endAngle: twoPi)
  }
  var center: CGPoint
  var radius: CGFloat
}
```

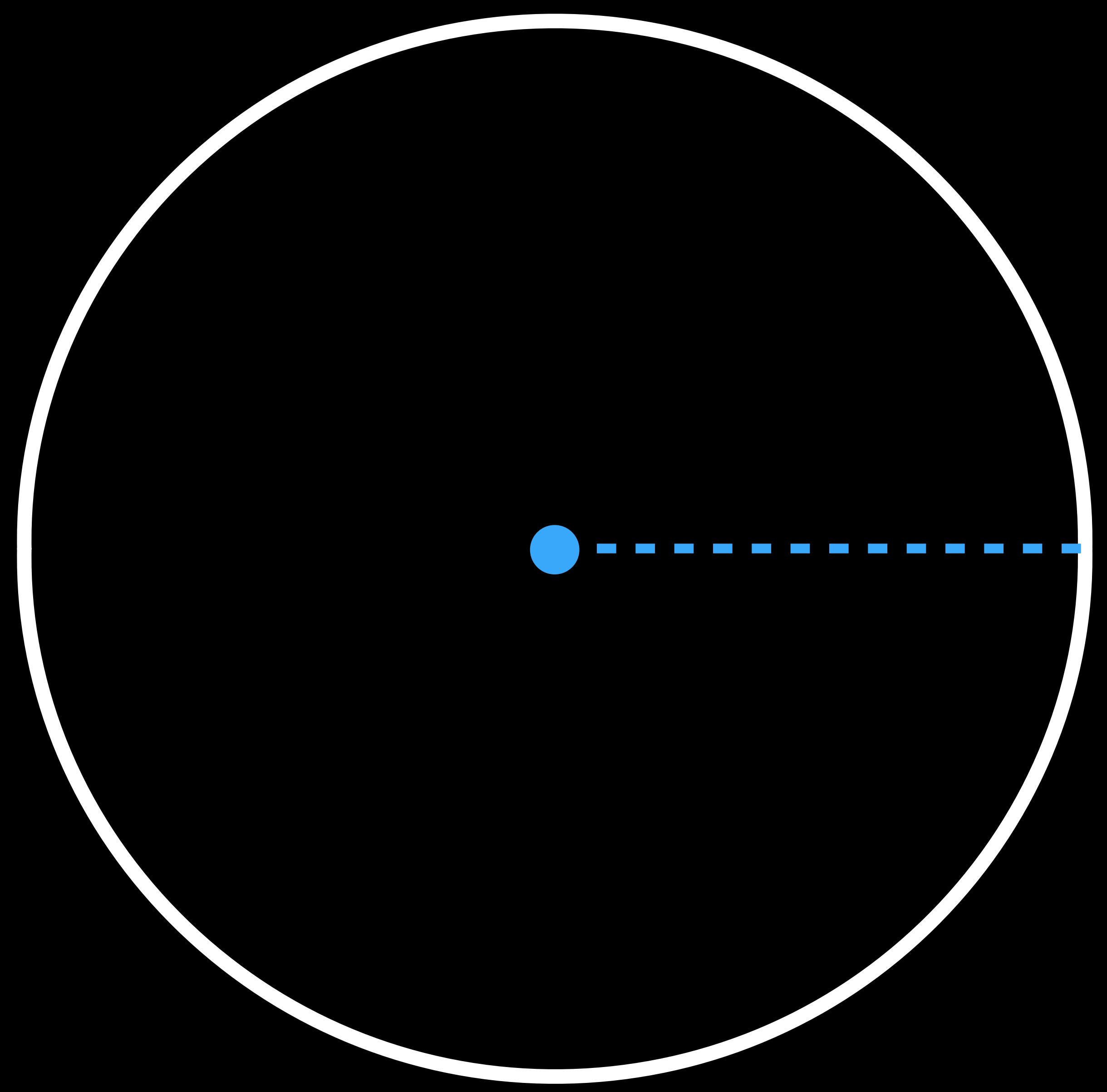
# Circle



```
protocol Drawable {
  func draw(renderer: Renderer)
}

struct Circle : Drawable {
  func draw(renderer: Renderer) {
    renderer.arcAt(center, radius: radius,
      startAngle: 0.0, endAngle: twoPi)
  }
  var center: CGPoint
  var radius: CGFloat
}
```

# Circle

```
protocol Drawable {
  func draw(renderer: Renderer)
}

struct Circle : Drawable {
  func draw(renderer: Renderer) {
    renderer.arcAt(center, radius: radius,
      startAngle: 0.0, endAngle: twoPi)
  }
  var center: CGPoint
  var radius: CGFloat
}
```



## Diagram

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) {
    for f in elements {
      f.draw(renderer)
    }
  }
  var elements: [Drawable] = []
}
```

## Diagram

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) {
    for f in elements {
      f.draw(renderer)
    }
  }
  var elements: [Drawable] = []
}
```

## Diagram

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) {
    for f in elements {
      f.draw(renderer)
    }
  }
  var elements: [Drawable] = []
}
```

## Diagram

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) {
    for f in elements {
      f.draw(renderer)
    }
  }
  var elements: [Drawable] = []
}
```

## Test It!

```
var circle = Circle(center:
  CGPoint(x: 187.5, y: 333.5),
  radius: 93.75)
```

## Test It!

```
var circle = Circle(center:
  CGPoint(x: 187.5, y: 333.5),
  radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])
```

# Test It!

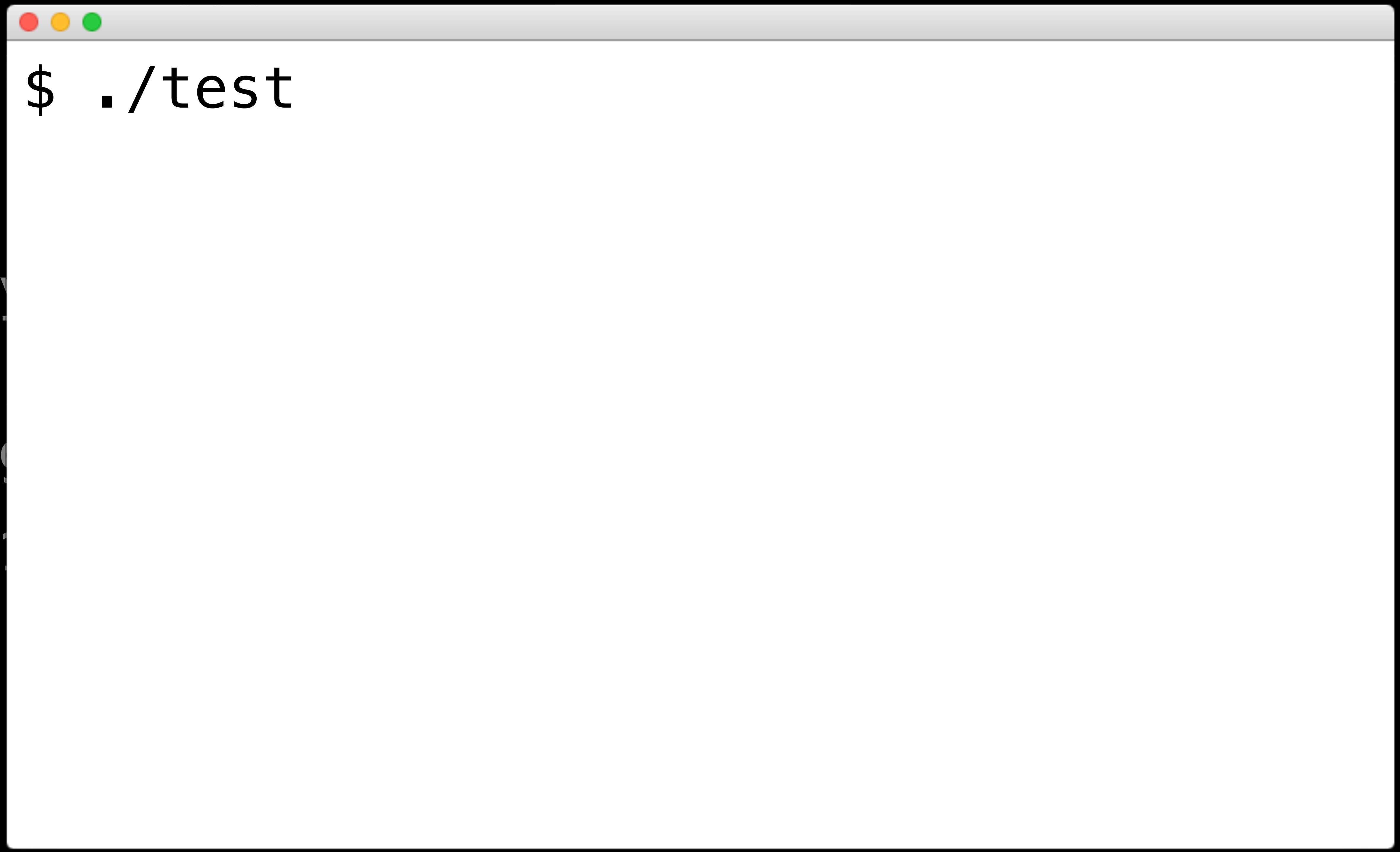
```
var circle = Circle(center:
  CGPoint(x: 187.5, y: 333.5),
  radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])
```

## Test It!

```
var circle = Circle(center:
  CGPoint(x: 187.5, y: 333.5),
  radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])

diagram.draw(Renderer())
```

## Test It!

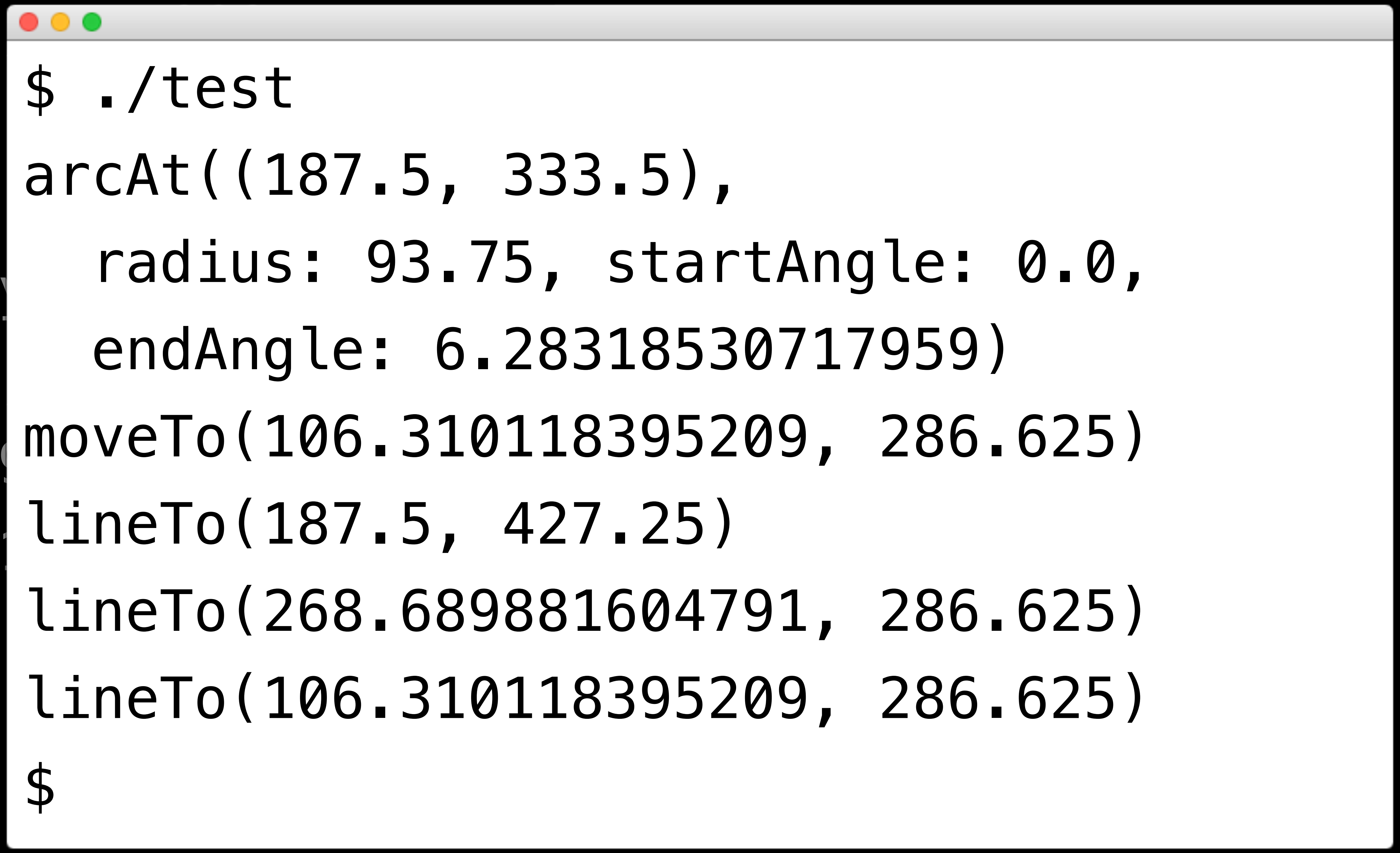
```
var circle = Circle(center:
  CGPoint(x: 187.5
  radius: 93.75)

var triangle = Poly
  CGPoint(x: 187.5
  CGPoint(x: 268.6
  CGPoint(x: 106.3

var diagram = Diag

diagram.draw(Renderer())
```



```
$ ./test
```

## Test It!

```
var circle = Circle(center:
  CGPoint(x: 187.5
  radius: 93.75)

var triangle = Poly
  CGPoint(x: 187.5
  CGPoint(x: 268.6
  CGPoint(x: 106.3

var diagram = Diag

diagram.draw(Renderer())
```



```
$ ./test
arcAt((187.5, 333.5),
  radius: 93.75, startAngle: 0.0,
  endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
$
```

## Renderer as a Protocol

```
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
      print("arcAt(\(center), radius: \(radius),"
        + " startAngle: \(startAngle), endAngle: \(endAngle))")
  }
}
```

# Renderer as a Protocol

```
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
      print("arcAt(\(center), radius: \(radius),"
        + " startAngle: \(startAngle), endAngle: \(endAngle))")
  }
}
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

## Renderer as a Protocol

```
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
      print("arcAt(\(center), radius: \(radius),"
        + " startAngle: \(startAngle), endAngle: \(endAngle))")
  }
}
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

# Renderer as a Protocol

```
protocol Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
      print("arcAt(\(center), radius: \(radius),"
        + " startAngle: \(startAngle), endAngle: \(endAngle))")
  }
}
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

## Renderer as a Protocol

```
protocol Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
      print("arcAt(\(center), radius: \(radius),"
        + " startAngle: \(startAngle), endAngle: \(endAngle))")
  }
}
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

## Renderer as a Protocol

```
protocol Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
      print("arcAt(\(center), radius: \(radius),"
        + " startAngle: \(startAngle), endAngle: \(endAngle))")
  }
}
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

# Renderer as a Protocol

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

```
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

# Renderer as a Protocol

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

```
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

# Renderer as a Protocol

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

```
struct Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

# Renderer as a Protocol

```swift
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

```swift
struct TestRenderer : Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

# Renderer as a Protocol

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

```
struct TestRenderer : Renderer {
  func moveTo(p: CGPoint) { print("moveTo(\(p.x), \(p.y))") }
  func lineTo(p: CGPoint) { print("lineTo(\(p.x), \(p.y))") }
  func arcAt(center: CGPoint, radius: CGFloat,
```

## Rendering with CoreGraphics

### Retroactive modeling

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

## Rendering with CoreGraphics

### Retroactive modeling

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

## Rendering with CoreGraphics

### Retroactive modeling

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

## Rendering with CoreGraphics

### Retroactive modeling

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}
```

# Rendering with CoreGraphics

## Retroactive modeling

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}

extension CGContext : Renderer {
  func moveTo(p: CGPoint) {  }
  func lineTo(p: CGPoint) {  }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {  }
}
```

## Rendering with CoreGraphics

### Retroactive modeling

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}

extension CGContext : Renderer {
  func moveTo(p: CGPoint) {  }
  func lineTo(p: CGPoint) {  }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {  }
}
```

## Rendering with CoreGraphics

### Retroactive modeling

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}

extension CGContext : Renderer {
  func moveTo(p: CGPoint) {  }
  func lineTo(p: CGPoint) {  }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {  }
}
```

# Rendering with CoreGraphics

## Retroactive modeling

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat)
}

extension CGContext : Renderer {
  func moveTo(p: CGPoint) {  }
  func lineTo(p: CGPoint) {  }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {  }
}
```

## Rendering with CoreGraphics

### Retroactive modeling

```
extension CGContext : Renderer {
  func moveTo(p: CGPoint) {

  }
  func lineTo(p: CGPoint) {

  }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {



  }
}
```

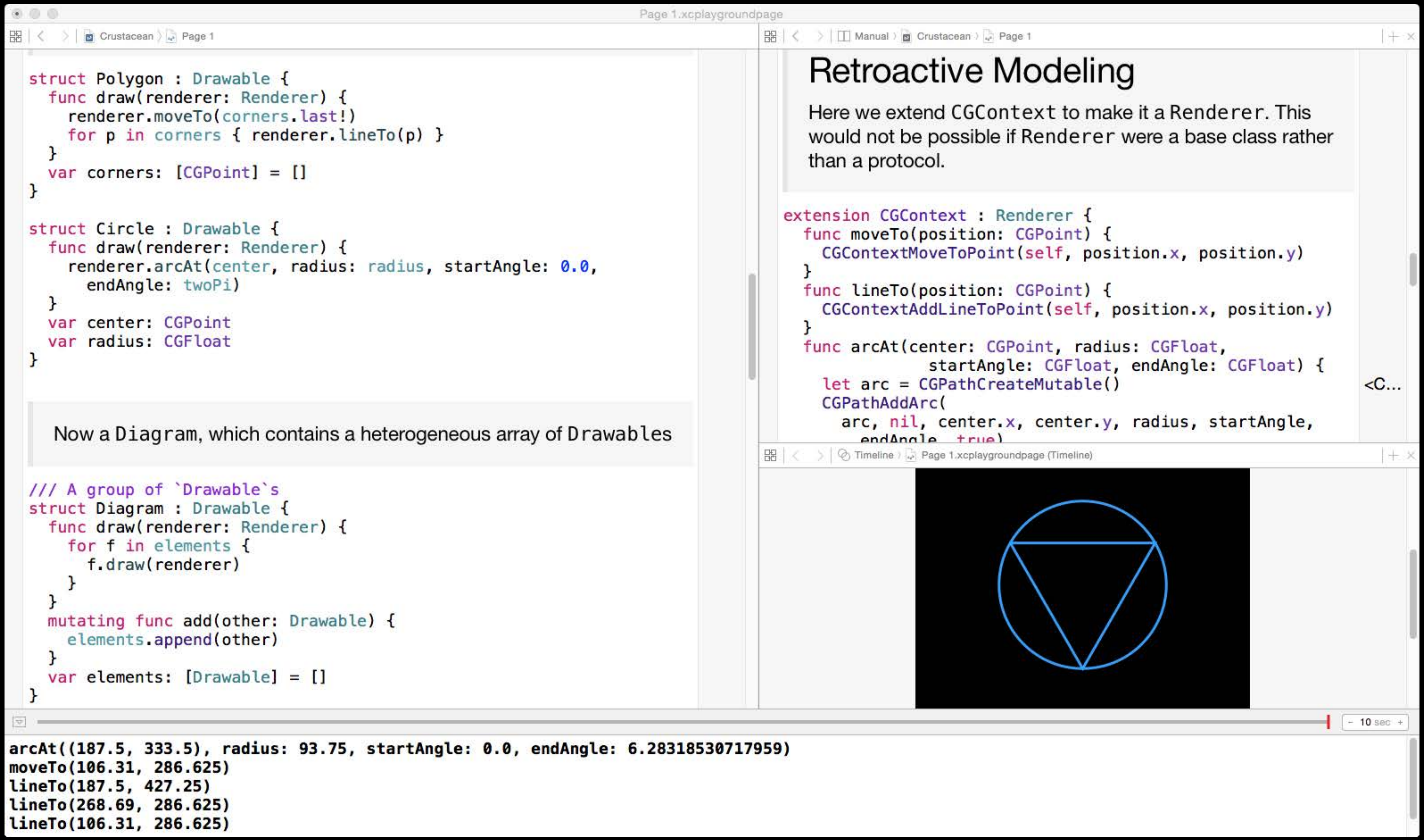
# Rendering with CoreGraphics

## Retroactive modeling

```
extension CGContext : Renderer {
  func moveTo(p: CGPoint) {
    CGContextMoveToPoint(self, position.x, position.y)
  }
  func lineTo(p: CGPoint) {
    CGContextAddLineToPoint(self, position.x, position.y)
  }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
    let arc = CGPathCreateMutable()
    CGPathAddArc(arc, nil, c.x, c.y, radius, startAngle, endAngle, true)
    CGContextAddPath(self, arc)
  }
}
```

# Crustacean: The Playground

https://developer.apple.com/sample-code/wwdc/2015/


Page 1.xcplaygroundpage
Crustacean 〉 Page 1
Manual 〉 Crustacean 〉 Page 1
Timeline 〉 Page 1.xcplaygroundpage (Timeline)
<C...
10 sec

```swift
struct Polygon : Drawable {
  func draw(renderer: Renderer) {
    renderer.moveTo(corners.last!)
    for p in corners { renderer.lineTo(p) }
  }
  var corners: [CGPoint] = []
}

struct Circle : Drawable {
  func draw(renderer: Renderer) {
    renderer.arcAt(center, radius: radius, startAngle: 0.0,
      endAngle: twoPi)
  }
  var center: CGPoint
  var radius: CGFloat
}
```

Now a `Diagram`, which contains a heterogeneous array of `Drawables`

```swift
/// A group of `Drawable`s
struct Diagram : Drawable {
  func draw(renderer: Renderer) {
    for f in elements {
      f.draw(renderer)
    }
  }
  mutating func add(other: Drawable) {
    elements.append(other)
  }
  var elements: [Drawable] = []
}
```

## Retroactive Modeling

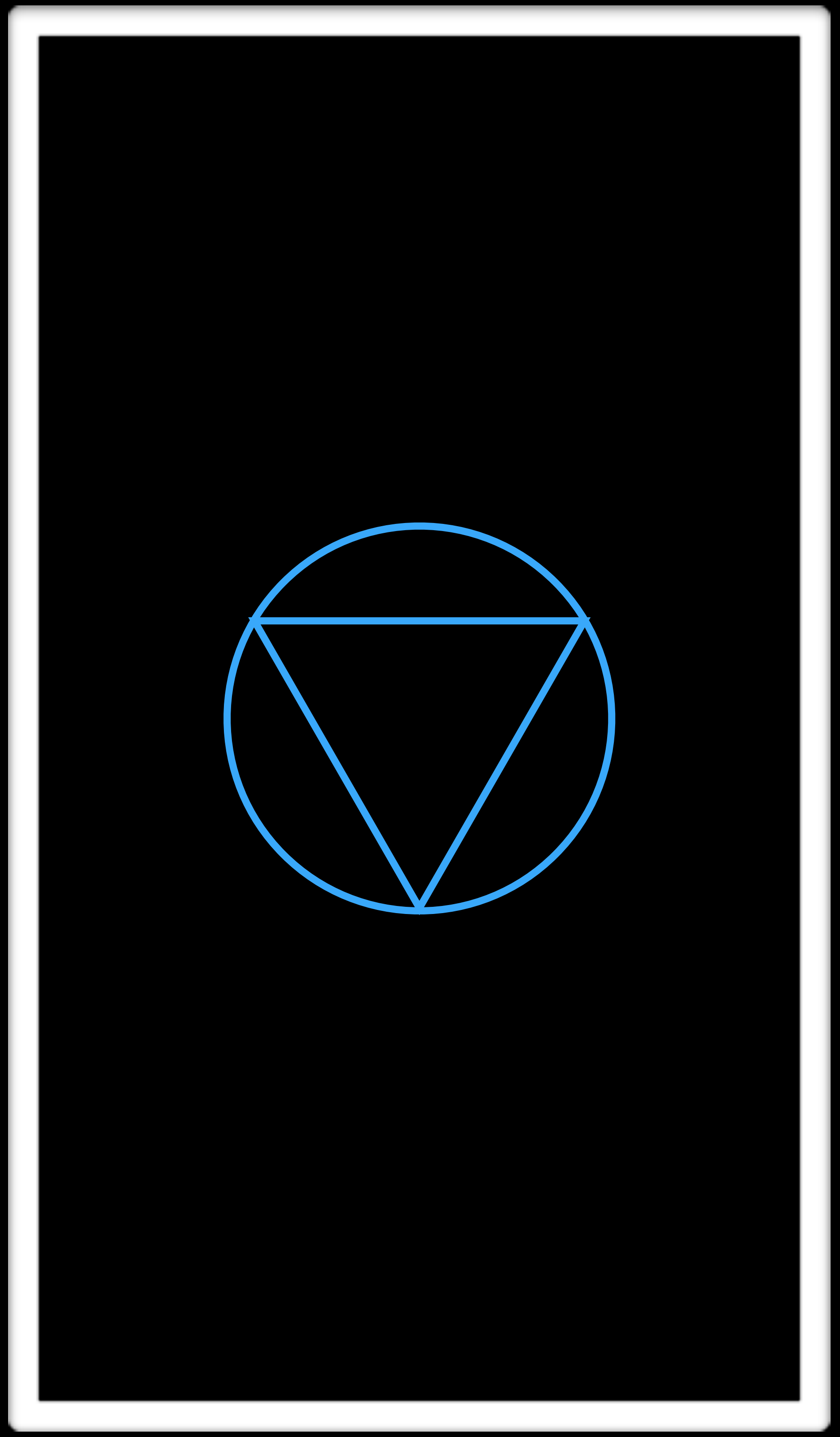
Here we extend `CGContext` to make it a `Renderer`. This would not be possible if `Renderer` were a base class rather than a protocol.

```swift
extension CGContext : Renderer {
  func moveTo(position: CGPoint) {
    CGContextMoveToPoint(self, position.x, position.y)
  }
  func lineTo(position: CGPoint) {
    CGContextAddLineToPoint(self, position.x, position.y)
  }
  func arcAt(center: CGPoint, radius: CGFloat,
             startAngle: CGFloat, endAngle: CGFloat) {
    let arc = CGPathCreateMutable()
    CGPathAddArc(
      arc, nil, center.x, center.y, radius, startAngle,
        endAngle, true)
```

```
arcAt((187.5, 333.5), radius: 93.75, startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.31, 286.625)
lineTo(187.5, 427.25)
lineTo(268.69, 286.625)
lineTo(106.31, 286.625)
```

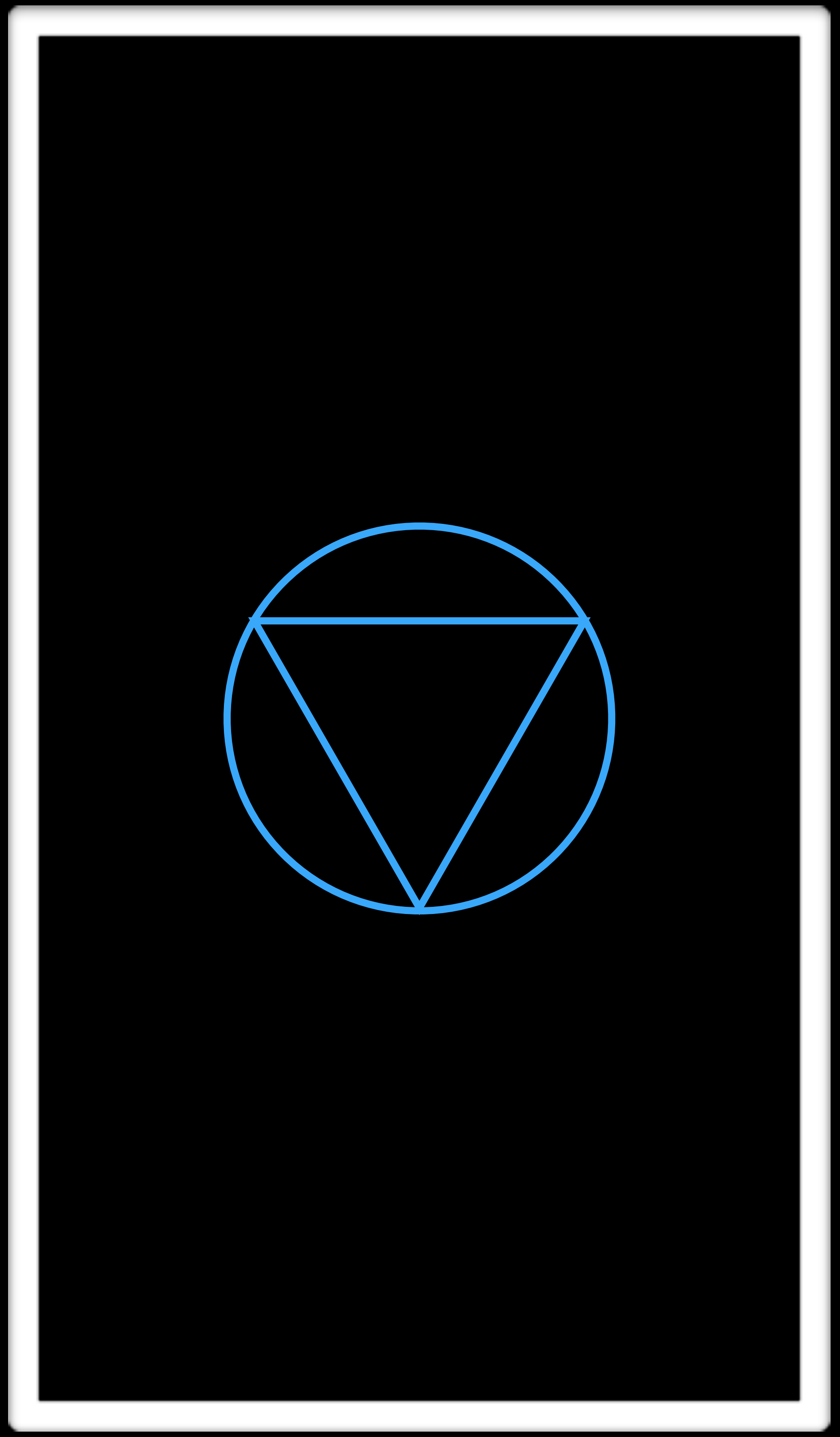
# Nested Diagram

```
var circle = Circle(center: CGPoint(x: 187.5, y: 333.5), radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])
```



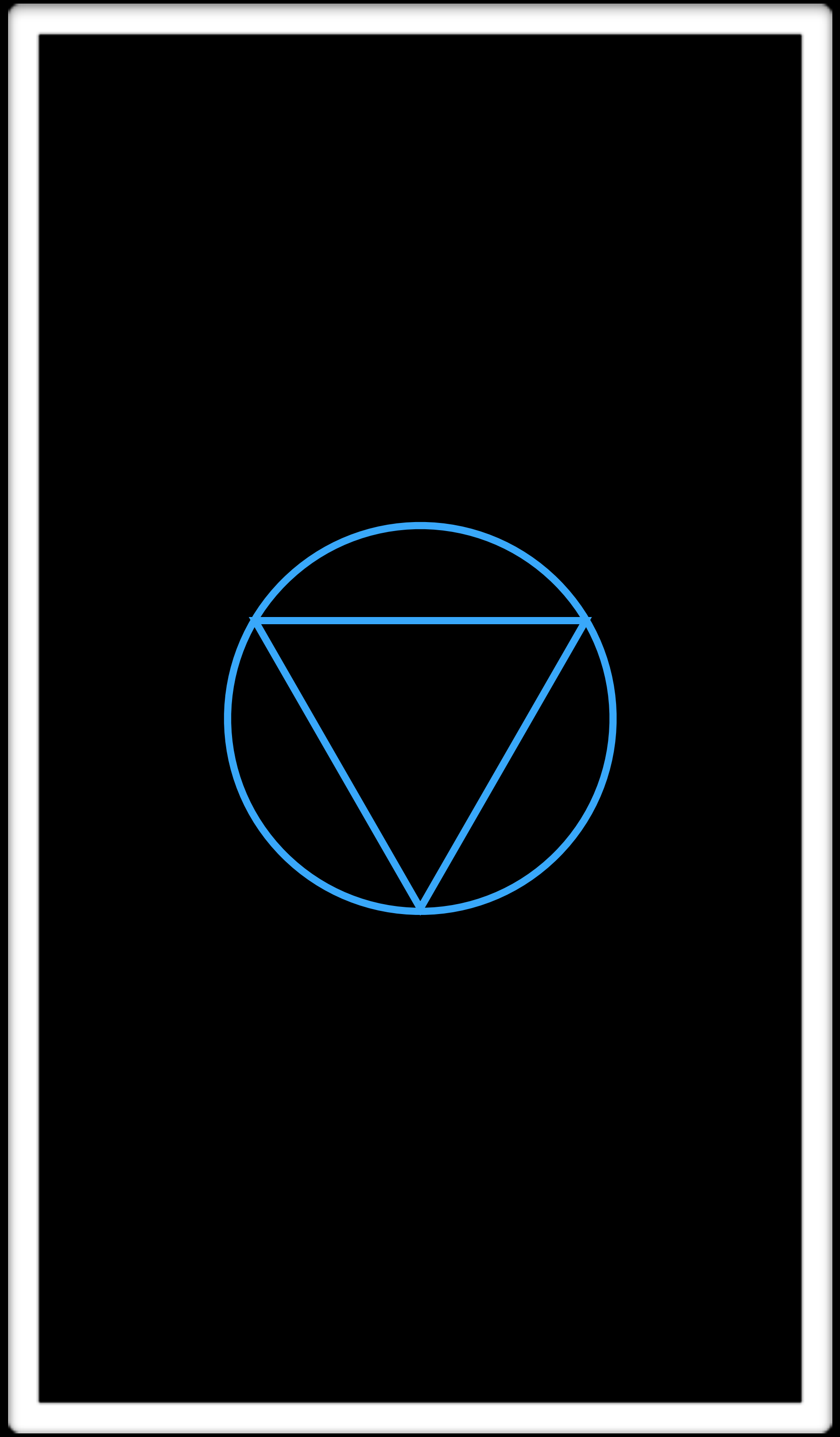
# Nested Diagram

```
var circle = Circle(center: CGPoint(x: 187.5, y: 333.5), radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])

diagram.elements.append(diagram)
```



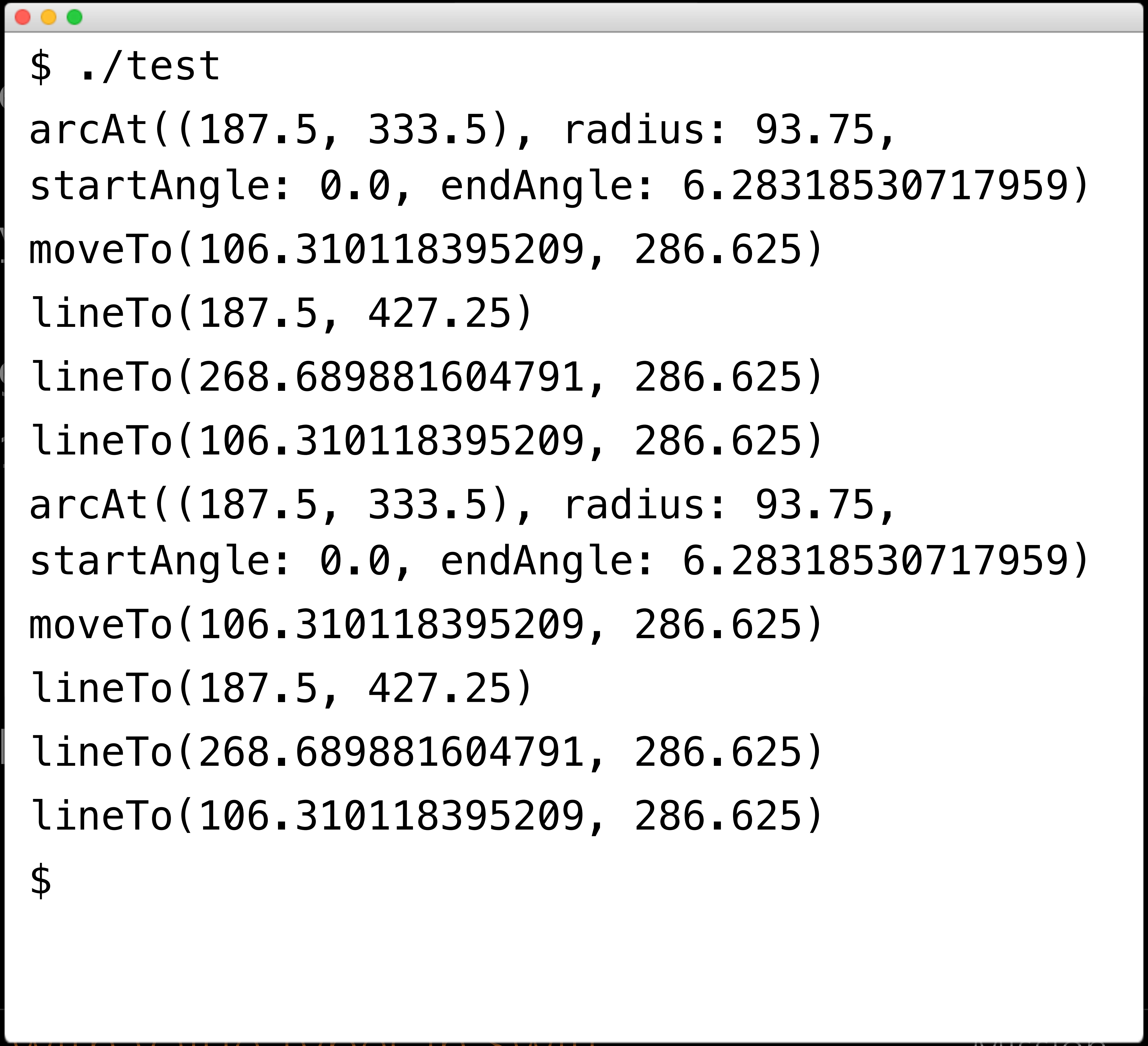
# Nested Diagram

```
var circle = Circle(center: CGPoint(x: 187.5, y: 333.5), radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])

diagram.elements.append(diagram)
```

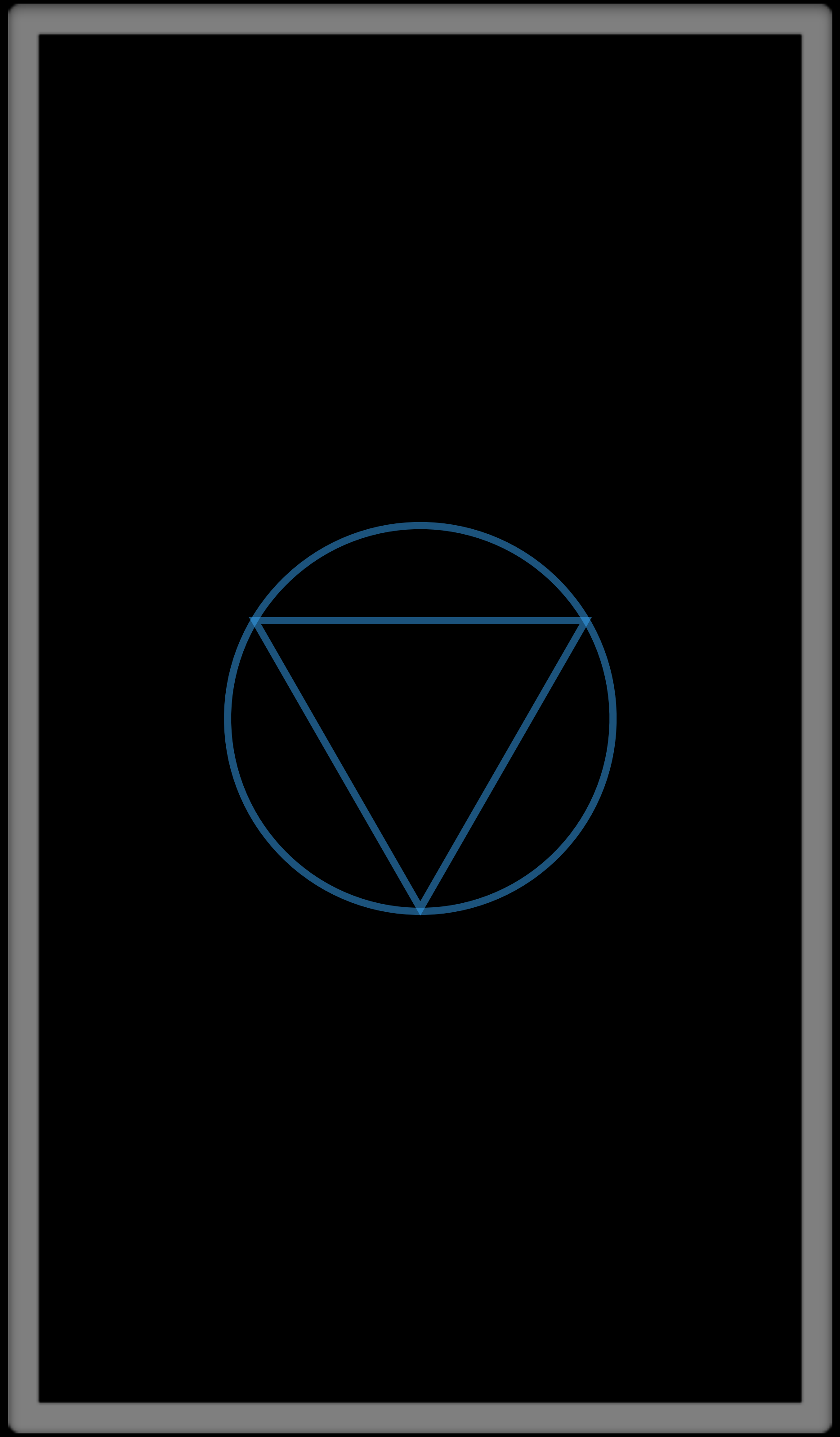


# Nested Diagram

```
var circle = Circl

var triangle = Pol
  CGPoint(x: 187.5
  CGPoint(x: 268.6
  CGPoint(x: 106.3

var diagram = Diag

diagram.elements.a
```



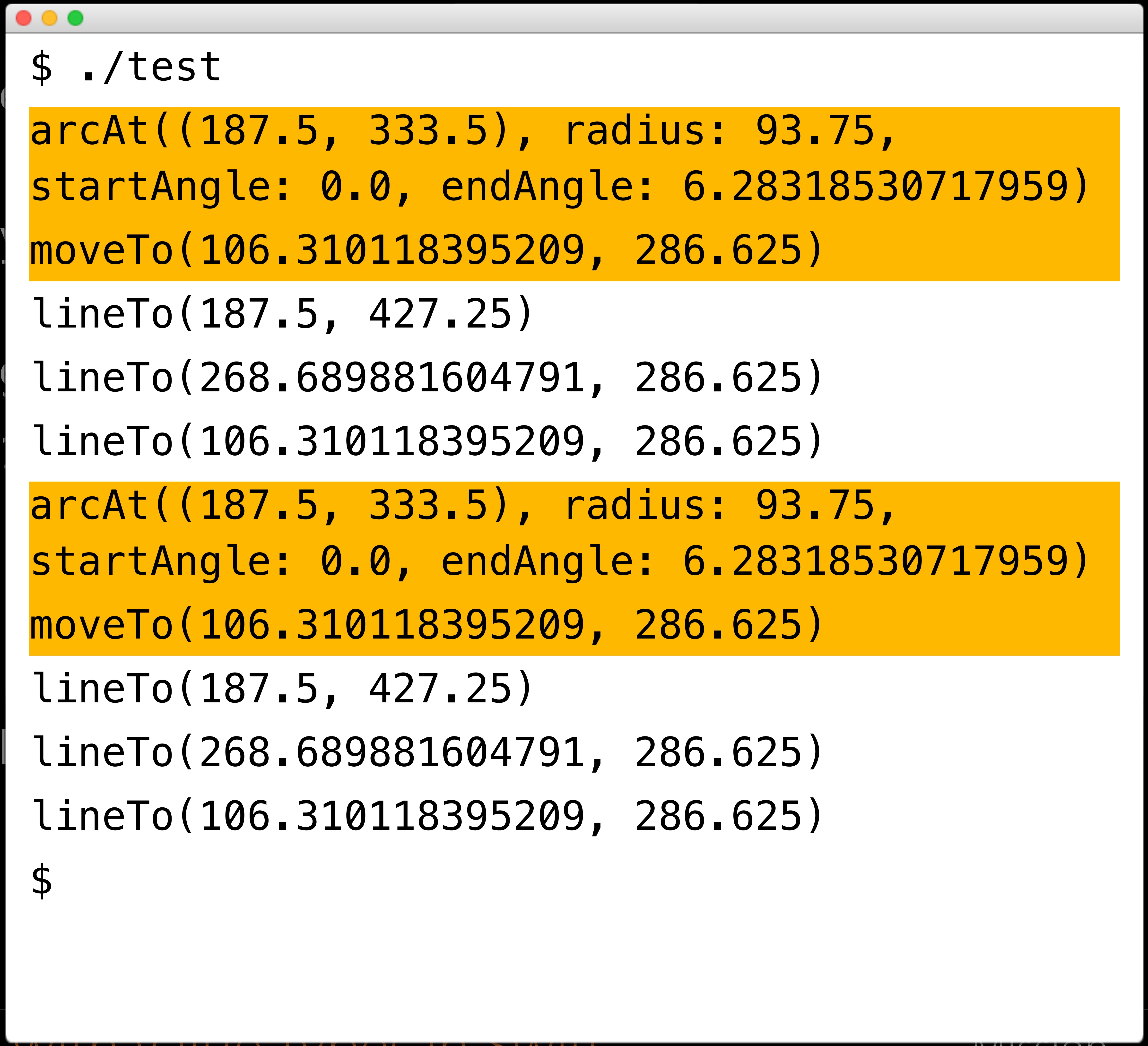
```
$ ./test
arcAt((187.5, 333.5), radius: 93.75,
startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
arcAt((187.5, 333.5), radius: 93.75,
startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
$
```



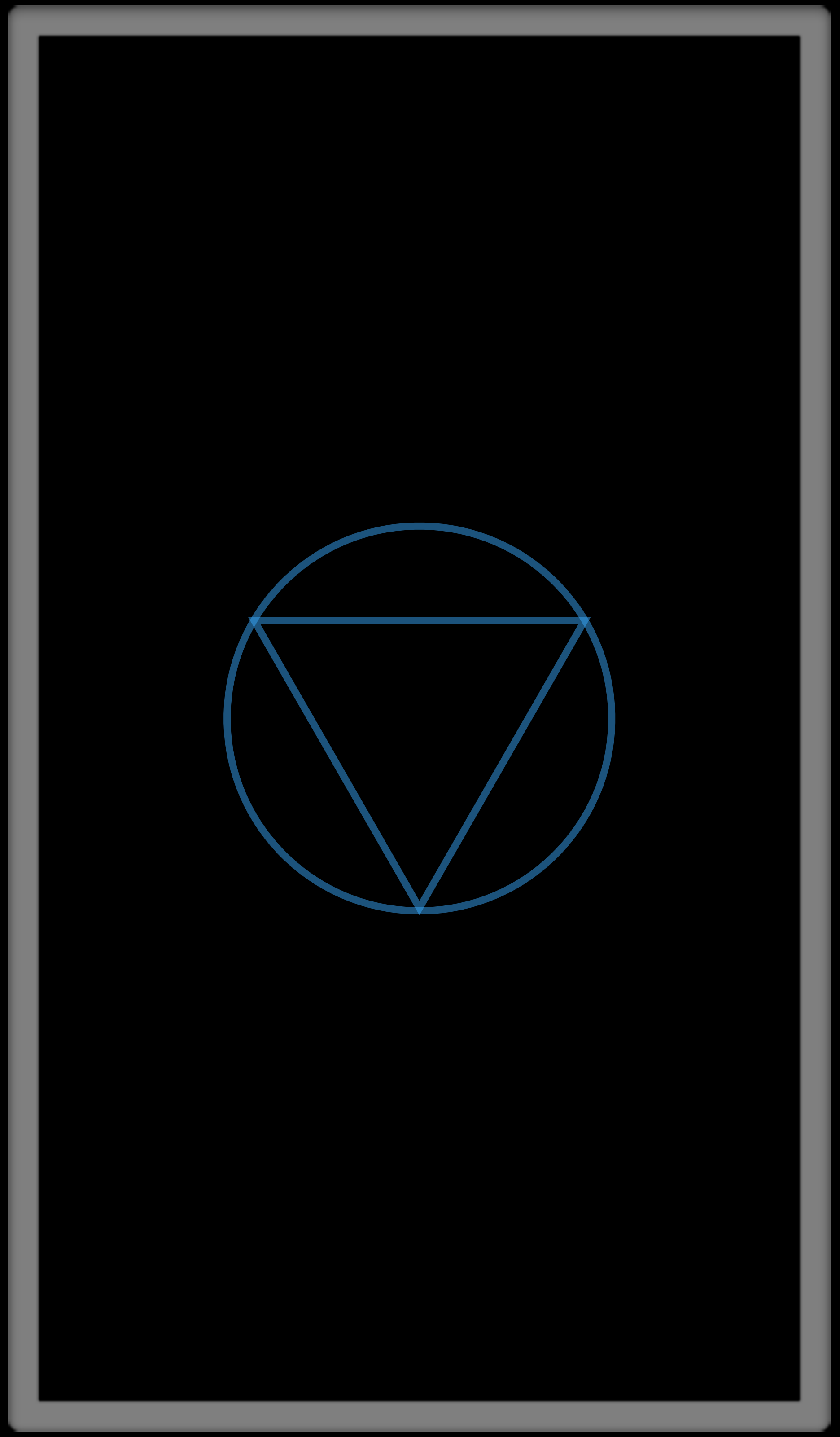
Building Better Apps with Value Types in Swift Mission Friday 2:30PM

# Nested Diagram

```
var circle = Circl

var triangle = Pol
  CGPoint(x: 187.5
  CGPoint(x: 268.6
  CGPoint(x: 106.3

var diagram = Diag

diagram.elements.a
```



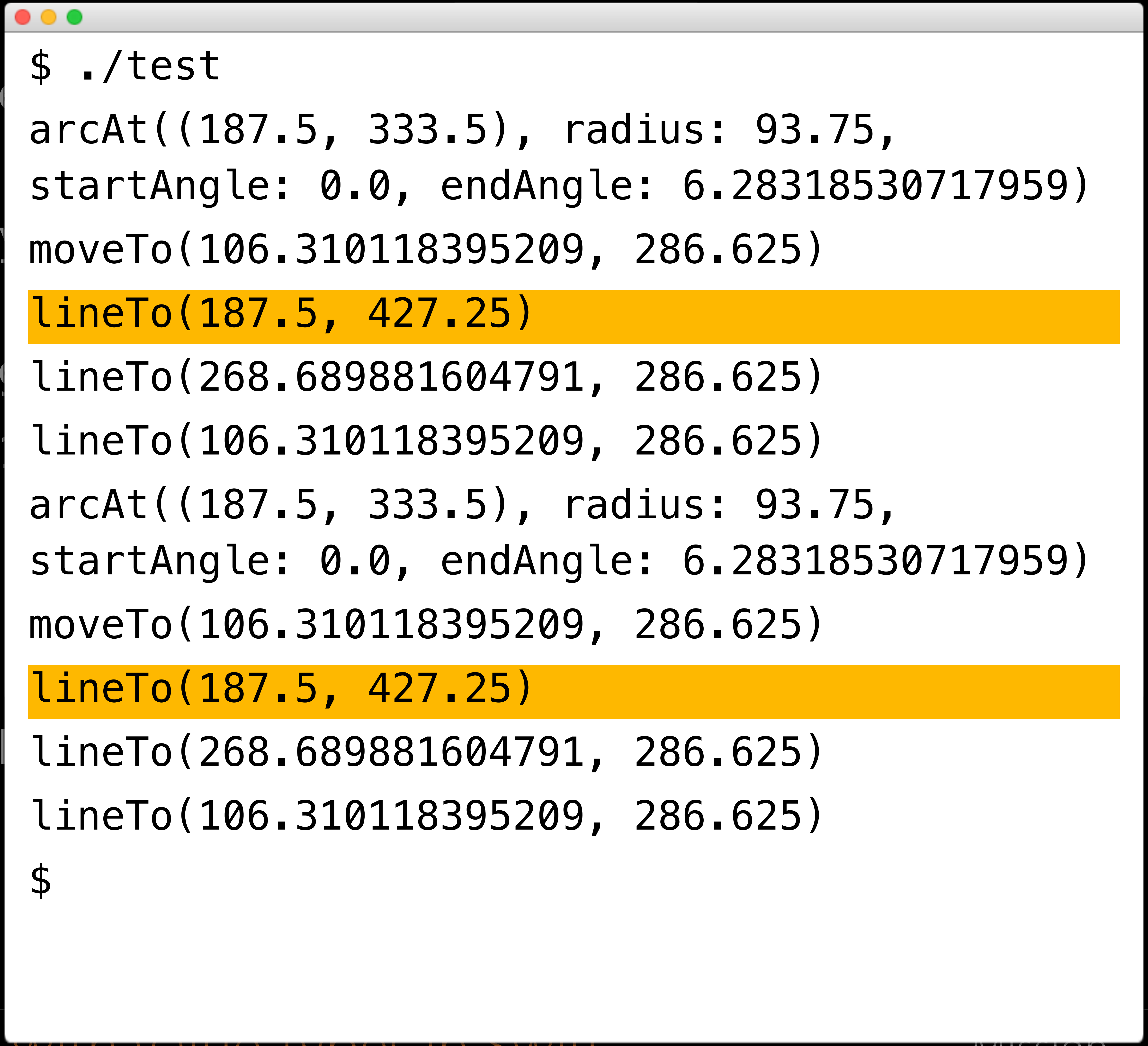
```
$ ./test
arcAt((187.5, 333.5), radius: 93.75,
startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
arcAt((187.5, 333.5), radius: 93.75,
startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
$
```



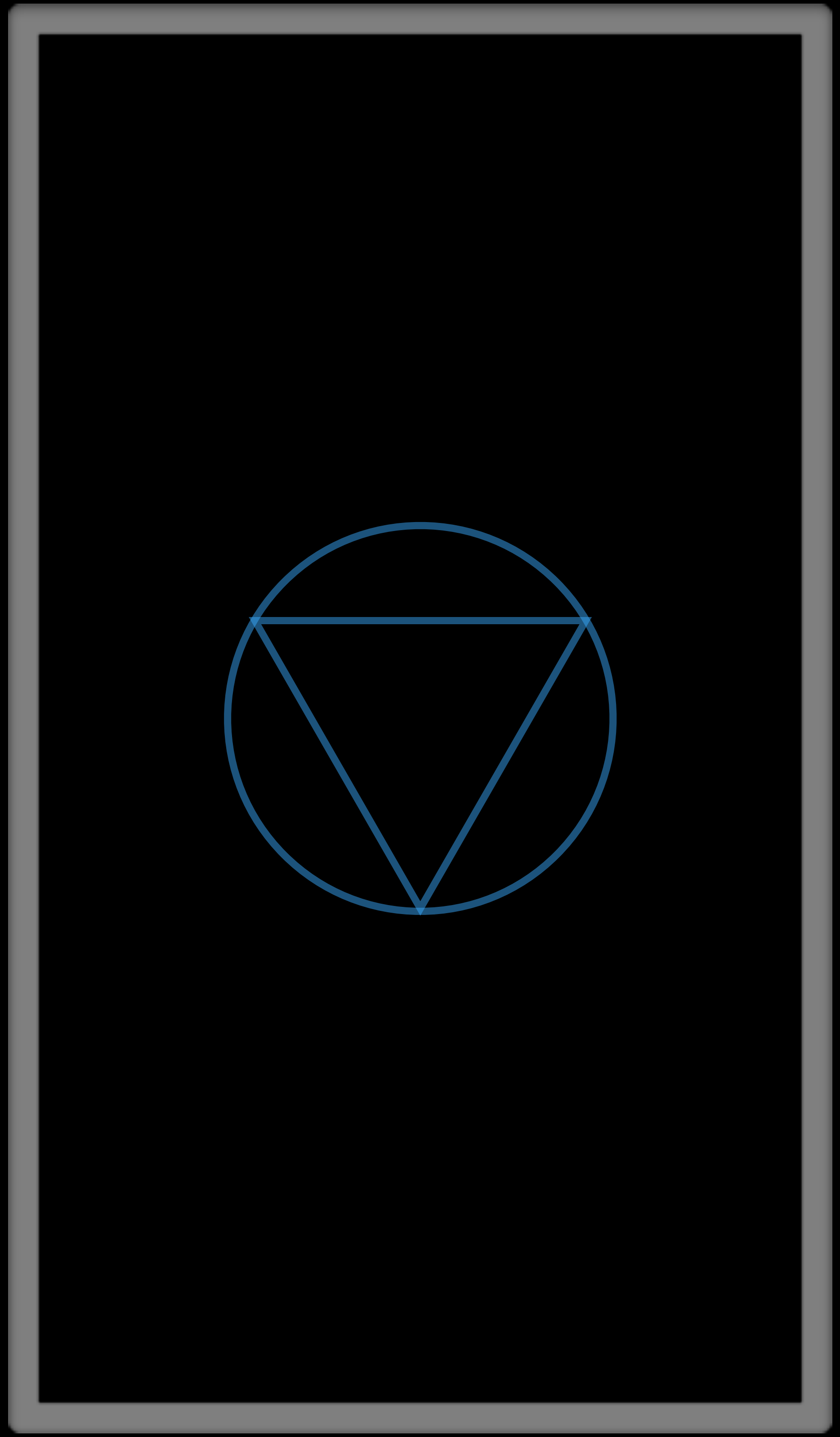
Building Better Apps with Value Types in Swift | Mission | Friday 2:30PM

# Nested Diagram

```
var circle = Circl

var triangle = Pol
  CGPoint(x: 187.5
  CGPoint(x: 268.6
  CGPoint(x: 106.3

var diagram = Diag

diagram.elements.a
```

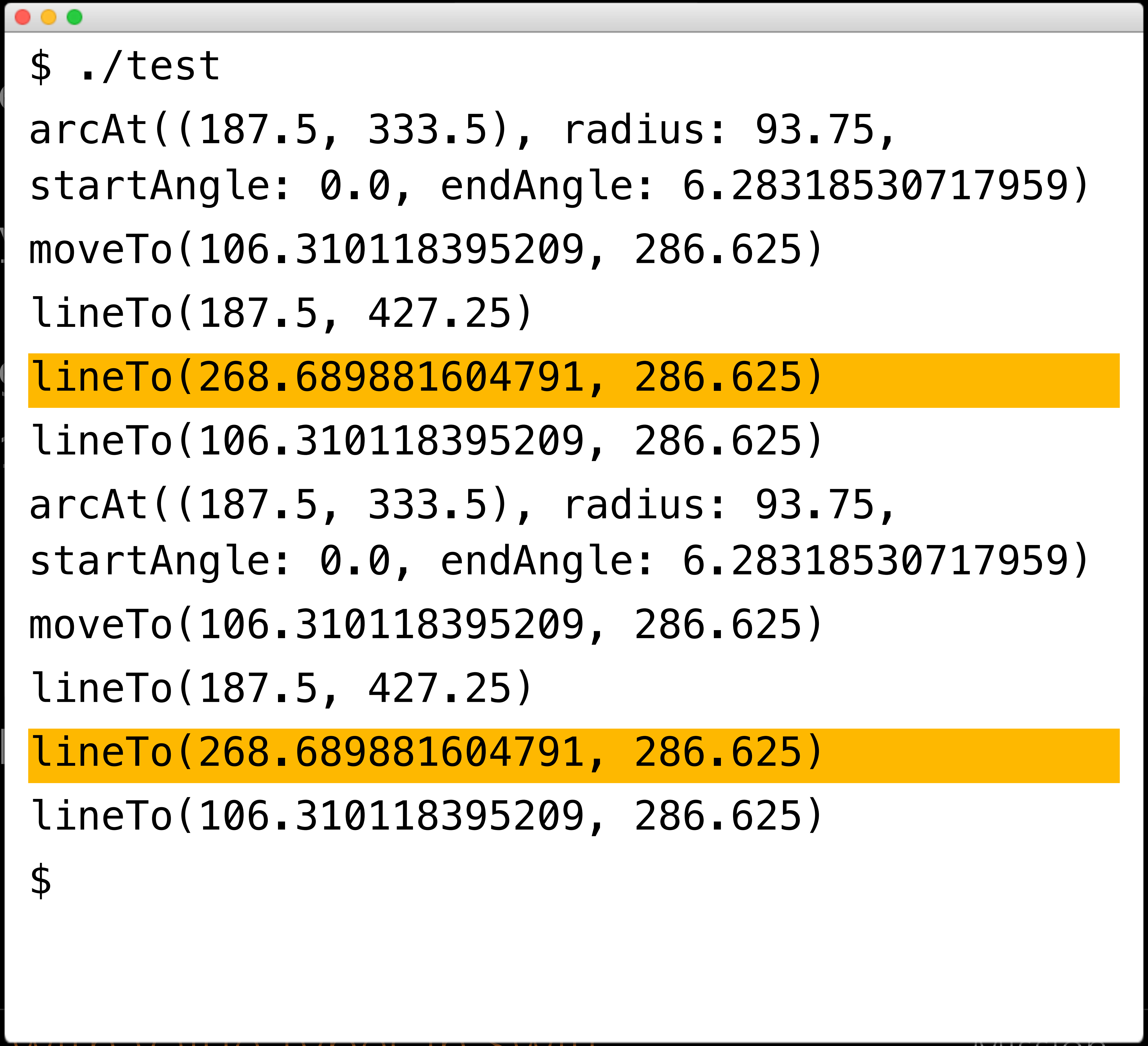


```
$ ./test
arcAt((187.5, 333.5), radius: 93.75,
startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
arcAt((187.5, 333.5), radius: 93.75,
startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
$
```

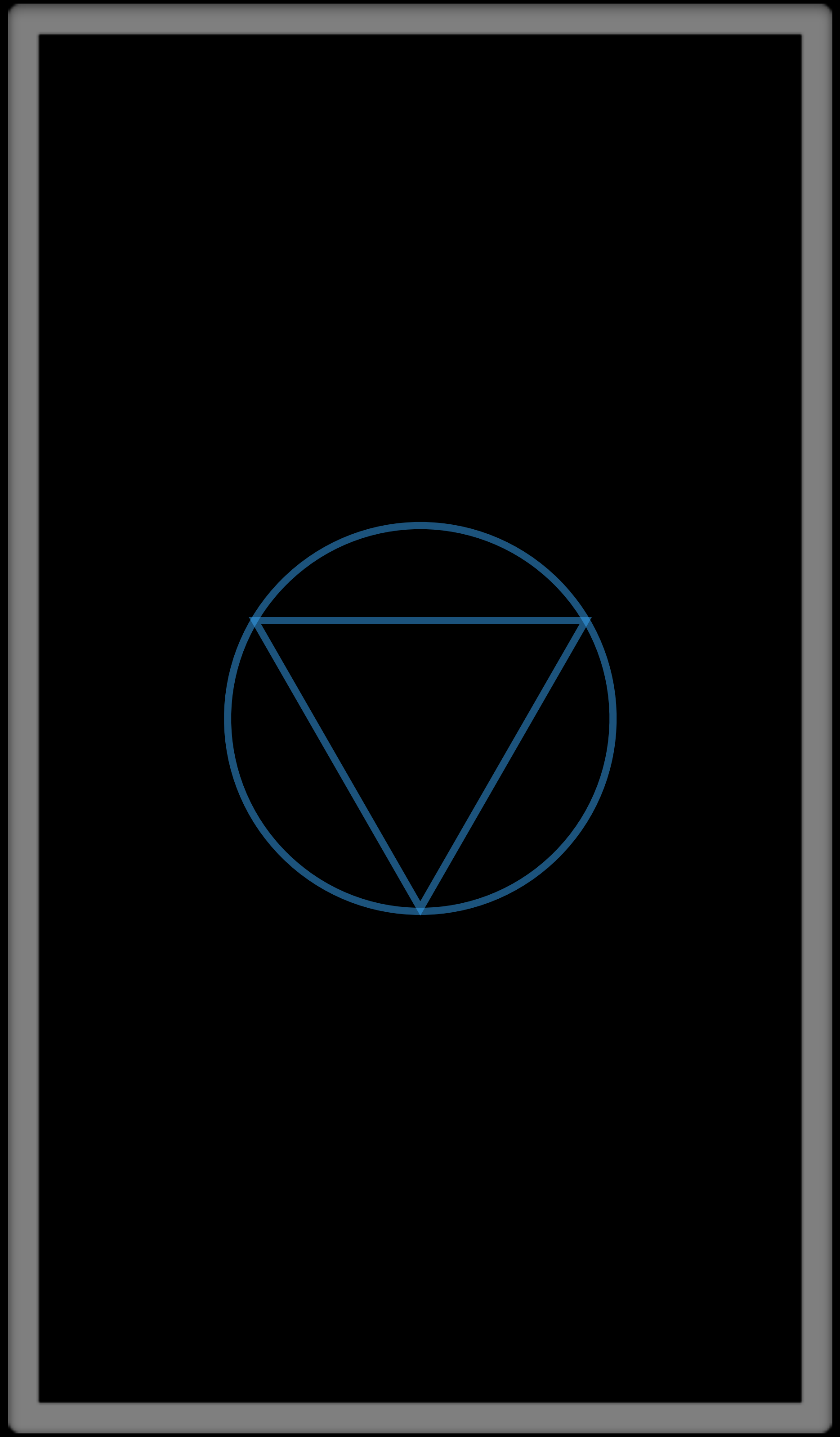


# Nested Diagram

```
var circle = Circl

var triangle = Pol
  CGPoint(x: 187.5
  CGPoint(x: 268.6
  CGPoint(x: 106.3

var diagram = Diag

diagram.elements.a
```

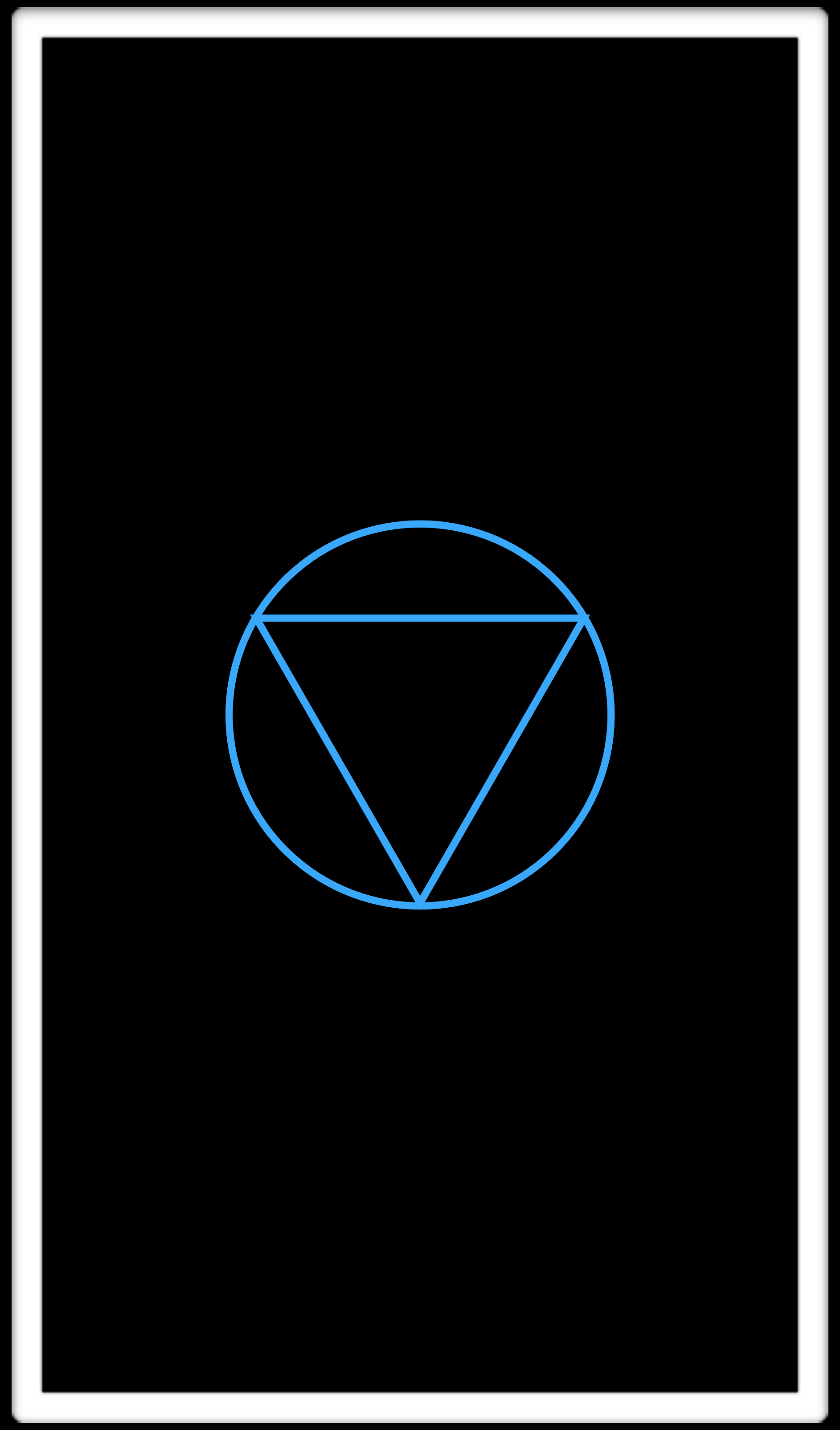


```
$ ./test
arcAt((187.5, 333.5), radius: 93.75,
startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
arcAt((187.5, 333.5), radius: 93.75,
startAngle: 0.0, endAngle: 6.28318530717959)
moveTo(106.310118395209, 286.625)
lineTo(187.5, 427.25)
lineTo(268.689881604791, 286.625)
lineTo(106.310118395209, 286.625)
$
```



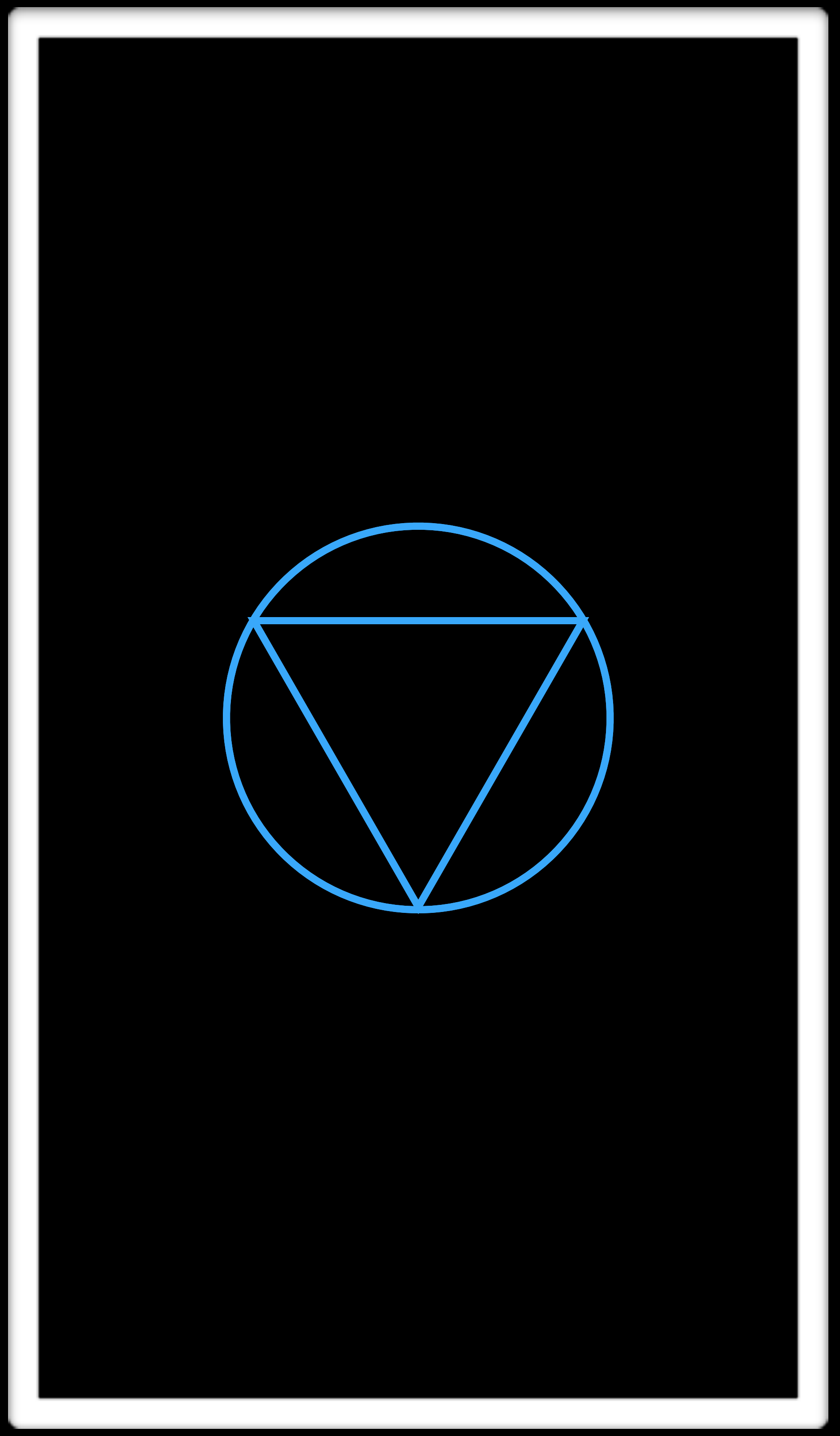
# Nested Diagram

```
var circle = Circle(center: CGPoint(x: 187.5, y: 333.5), radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])

diagram.elements.append(diagram)
```



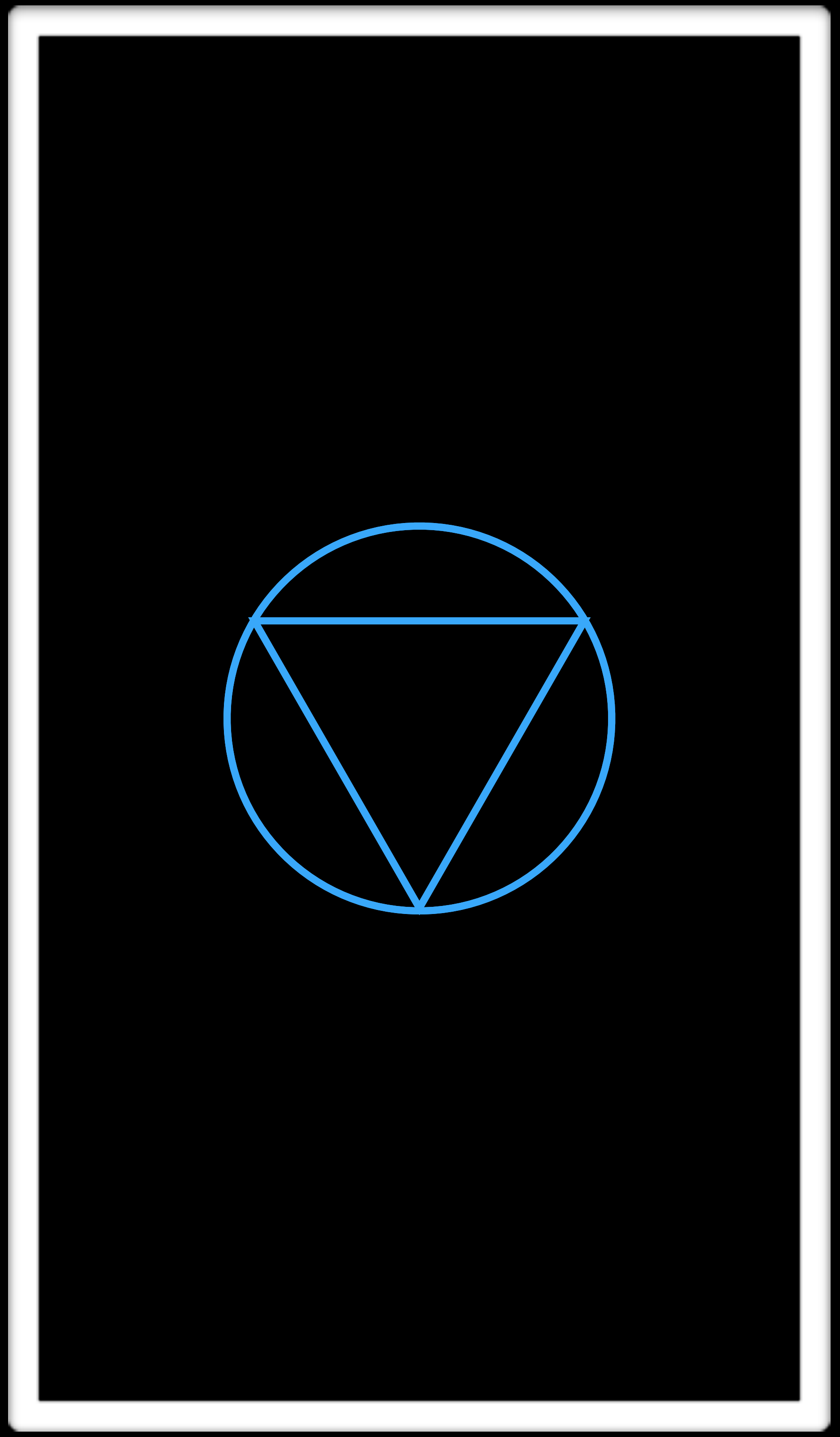
# Nested Diagram

```
var circle = Circle(center: CGPoint(x: 187.5, y: 333.5), radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])

diagram.elements.append(diagram)
```



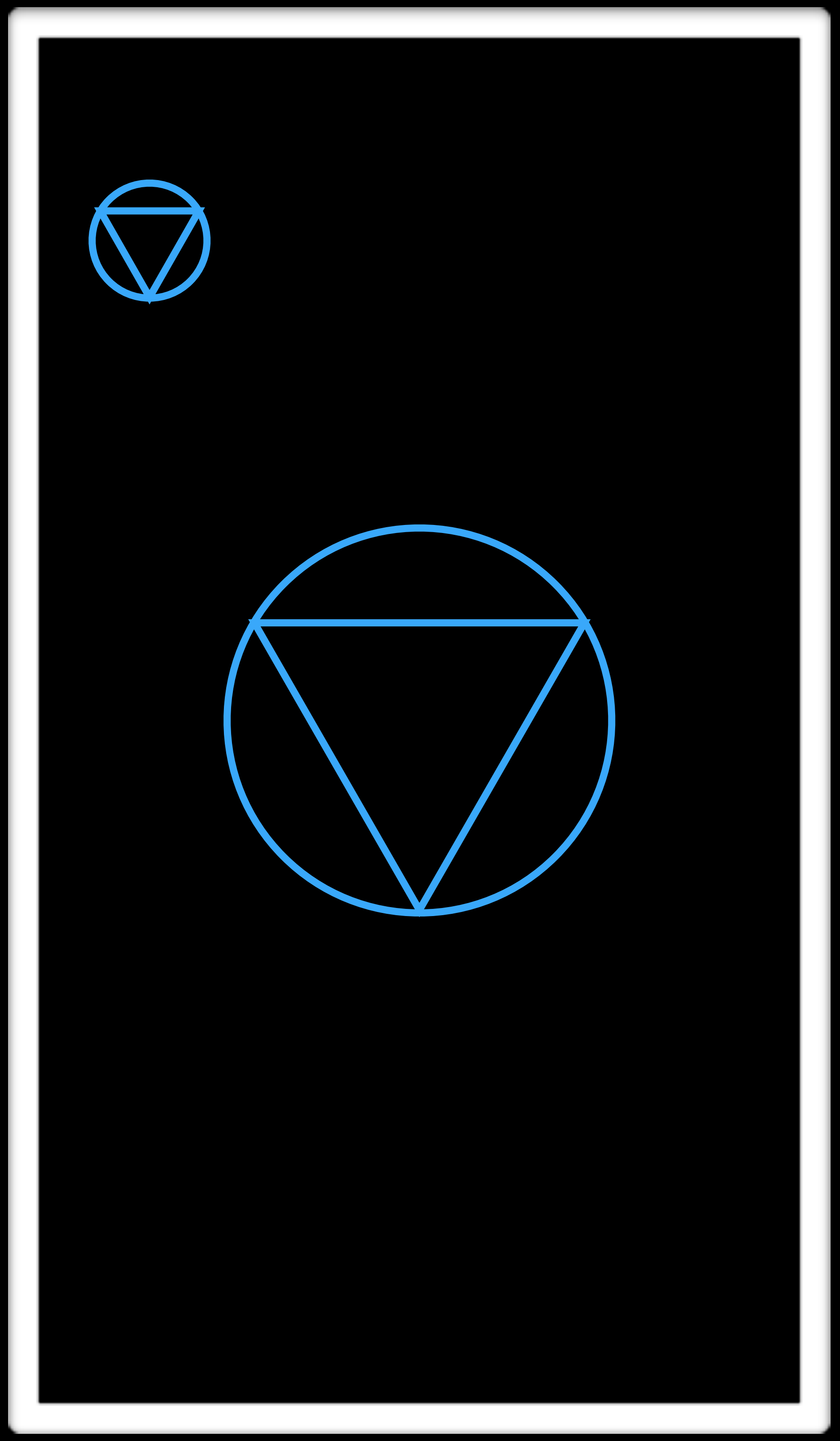
# Nested Diagram

```
var circle = Circle(center: CGPoint(x: 187.5, y: 333.5), radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])

diagram.elements.append(
  Scaled(scale: 0.3, subject: diagram))
```



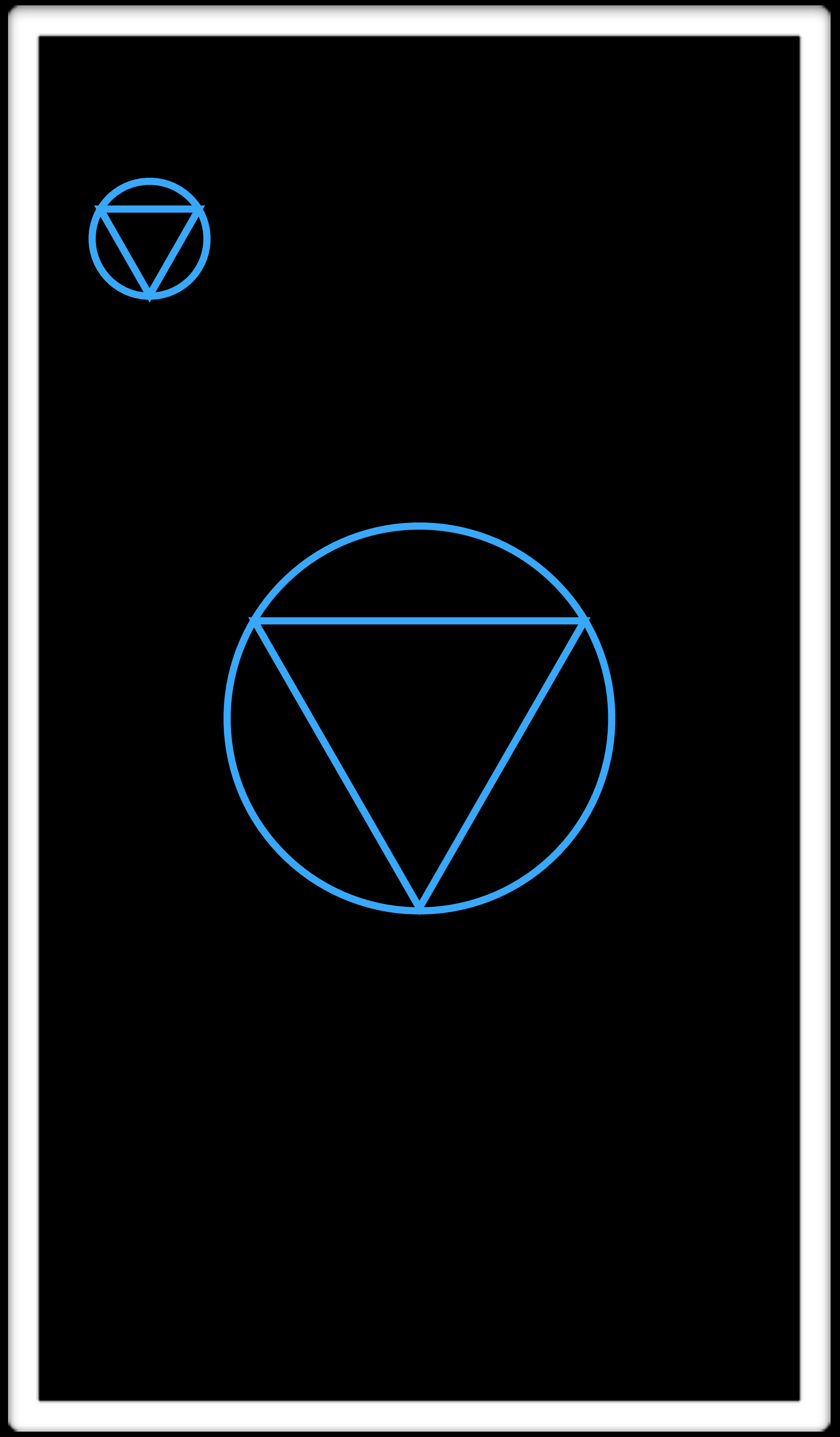
## Nested Diagram

```
var circle = Circle(center: CGPoint(x: 187.5, y: 333.5), radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])

diagram.elements.append(
  Scaled(scale: 0.3, subject: diagram))
```



# Nested Diagram

```
var circle = Circle(center: CGPoint(x: 187.5, y: 333.5), radius: 93.75)

var triangle = Polygon(corners: [
  CGPoint(x: 187.5, y: 427.25),
  CGPoint(x: 268.69, y: 286.625),
  CGPoint(x: 106.31, y: 286.625)])

var diagram = Diagram(elements: [circle, triangle])

diagram.elements.append(
  Scaled(scale: 0.3, subject: diagram))
```

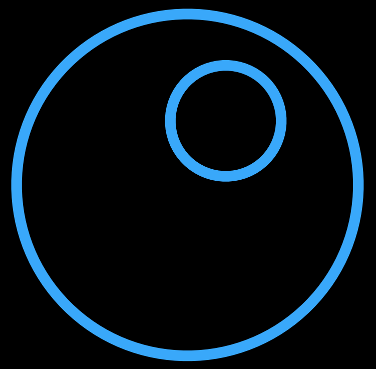


## Protocols and Generics for Testability

So much better than mocks

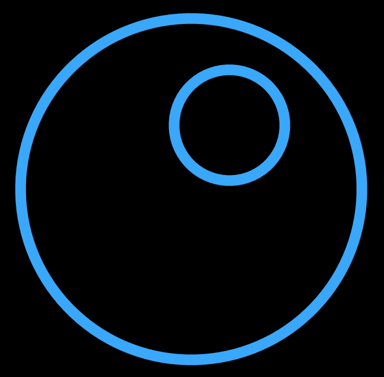
Disciplined decoupling is a beautiful thing.

## Bubble



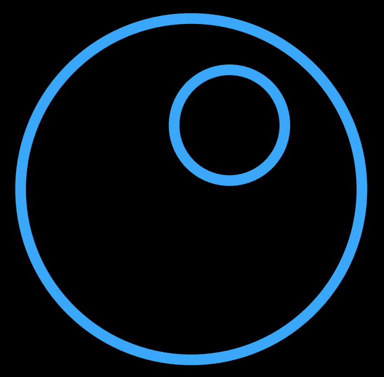
```
struct Bubble : Drawable {
  func draw(r: Renderer) {
    r.arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
    r.arcAt(highlightCenter, radius: highlightRadius,
        startAngle: 0, endAngle: twoPi)
  }
}
```

## Bubble



```
struct Bubble : Drawable {
  func draw(r: Renderer) {
    r.arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
    r.arcAt(highlightCenter, radius: highlightRadius,
        startAngle: 0, endAngle: twoPi)
  }
}
struct Circle : Drawable {
  func draw(r: Renderer) {
    r.arcAt(center, radius: radius, startAngle: 0.0, endAngle: twoPi)
  }
}
```

## Bubble



```
struct Bubble : Drawable {
  func draw(r: Renderer) {
    r.circleAt(center, radius: radius)
    r.circleAt(highlightCenter, radius: highlightRadius)
  }
}

struct Circle : Drawable {
  func draw(r: Renderer) {
    r.circleAt(center, radius: radius)
  }
}
```

## Adding a Circle Primitive

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)

  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}
```

## Adding a Circle Primitive

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)

  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}
```

## Adding a Circle Primitive

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}
```

## Adding a Circle Primitive

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}
```

New requirement

## Implementing the Requirement

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension TestRenderer {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

New requirement

# Implementing the Requirement… Again!

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension CGContext {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

# Implementing the Requirement… Again!

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension CGContext {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

Duplicate implementation

✗

## Implementing the Requirement… Again!

```swift
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension CGContext {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

# Protocol Extensions

NEW

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

## Protocol Extensions

NEW

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

## Protocol Extensions

NEW

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

Shared implementation

✓

## Protocol Extensions

NEW

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

✓

## Protocol Extensions

### Requirements create customization points

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) {
    arcAt(center, radius: radius, startAngle: 0, endAngle: twoPi)
  }
}
```

## Protocol Extensions

Requirements create customization points

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }

}
```

# Protocol Extensions

## Requirements create customization points

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }

}
```

# Protocol Extensions

## Requirements create customization points

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}
```

## Protocol Extensions

Requirements create customization points

```
protocol Renderer {
  func moveTo(p: CGPoint)
  func lineTo(p: CGPoint)
  func circleAt(center: CGPoint, radius: CGFloat)
  func arcAt(
    center: CGPoint, radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat)
}

extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}
```

## Protocol Extensions

### Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}
```

```
extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}
```

# Protocol Extensions

## Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}
```

```
let r = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## Protocol Extensions

### Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

# Protocol Extensions

## Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r: Renderer = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

# Protocol Extensions

## Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r: Renderer = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r: Renderer = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

# Protocol Extensions

## Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r: Renderer = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

Provides the requirement

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r: Renderer = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## Protocol Extensions

Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

let r: Renderer = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## Protocol Extensions

### Requirements create customization points

```
extension Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }
  func rectangleAt(edges: CGRect) { ... }
}

extension TestRenderer : Renderer {
  func circleAt(center: CGPoint, radius: CGFloat) { ... }

}

let r: Renderer = TestRenderer()
r.circleAt(origin, radius: 1);
r.rectangleAt(edges);
```

## More Protocol Extension Tricks

Scenes from the standard library and beyond

## More Protocol Extension Tricks

### Constrained extensions

```
extension CollectionType {
  public func indexOf(element: Generator.Element) -> Index? {
    for i in self.indices {
      if self[i] == element {
        return i
      }
    }
    return nil
  }
}
```

## More Protocol Extension Tricks

### Constrained extensions

```
extension CollectionType {
  public func indexOf(element: Generator.Element) -> Index? {
    for i in self.indices {
      if self[i] == element {
        return i
      }
    }
    return nil
  }
}
```

binary operator '==' cannot be applied to two Generator.Element operands

## More Protocol Extension Tricks

### Constrained extensions

```
extension CollectionType {
  public func indexOf(element: Generator.Element) -> Index? {
    for i in self.indices {
      if self[i] == element {
        return i
      }
    }
    return nil
  }
}
```

binary operator '==' cannot be applied to two Generator.Element operands

## More Protocol Extension Tricks

NEW

### Constrained extensions

```
extension CollectionType where Generator.Element : Equatable {
  public func indexOf(element: Generator.Element) -> Index? {
    for i in self.indices {
      if self[i] == element {
        return i
      }
    }
    return nil
  }
}
```

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }
```

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }
```

```
let position = binarySearch([2, 3, 5, 7], forKey: 5)
```

# More Protocol Extension Tricks

## Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }
```

cannot invoke 'binarySearch' with an argument list of type '([Int], forKey: Int)'

```
let position = binarySearch([2, 3, 5, 7], forKey: 5)
```

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Int : Ordered {
  func precedes(other: Int) -> Bool { return self < other }
}




let position = binarySearch([2, 3, 5, 7], forKey: 5)
```

## More Protocol Extension Tricks

Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Int : Ordered {
  func precedes(other: Int) -> Bool { return self < other }
}
```

```
let position = binarySearch(["2", "3", "5", "7"], forKey: "5")
```

# More Protocol Extension Tricks

## Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Int : Ordered {
  func precedes(other: Int) -> Bool { return self < other }
}
```

cannot invoke 'binarySearch' with an argument list of type '([String], forKey: String)'

```
let position = binarySearch(["2", "3", "5", "7"], forKey: "5")
```

# More Protocol Extension Tricks

## Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Int : Ordered {
  func precedes(other: Int) -> Bool { return self < other }
}
extension String : Ordered {
  func precedes(other: String) -> Bool { return self < other }
}

let position = binarySearch(["2", "3", "5", "7"], forKey: "5")
```

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}


let position = binarySearch(["2", "3", "5", "7"], forKey: "5")
```

## More Protocol Extension Tricks

Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}
extension Double : Ordered {}

let position = binarySearch(["2", "3", "5", "7"], forKey: "5")
```

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}


let position = binarySearch(["2", "3", "5", "7"], forKey: "5")
```

# More Protocol Extension Tricks

## Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}
let truth = 3.14.precedes(98.6)   // Compiles

let position = binarySearch(["2", "3", "5", "7"], forKey: "5")
```

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}
let truth = 3.14.precedes(98.6)   // Compiles

let position = binarySearch([2.0, 3.0, 5.0, 7.0], forKey: 5.0)
```

# More Protocol Extension Tricks

## Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}
let truth = 3.14.precedes(98.6)   // Compiles

let position = binarySearch([2.0, 3.0, 5.0, 7.0], forKey: 5.0)
```

cannot invoke 'binarySearch' with an argument list of type '([Double], forKey: Double)'

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}
let truth = 3.14.precedes(98.6)   // Compiles
```

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Ordered where Self : Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}
let truth = 3.14.precedes(98.6)
```

## More Protocol Extension Tricks

### Retroactive adaptation

```
protocol Ordered {
  func precedes(other: Self) -> Bool
}
func binarySearch<T : Ordered>(sortedKeys: [T], forKey k: T) -> Int { ... }

extension Ordered where Self : Comparable {
  func precedes(other: Self) -> Bool { return self < other }
}
extension Int : Ordered {}
extension String : Ordered {}
let truth = 3.14.precedes(98.6)
```

'Double' does not have a member named 'precedes'

## More Protocol Extension Tricks

### Generic beautification

```
func binarySearch<
  C : CollectionType where C.Index == RandomAccessIndexType,
  C.Generator.Element : Ordered
>(sortedKeys: C, forKey k: C.Generator.Element) -> Int {

  ...

}

let pos = binarySearch([2, 3, 5, 7, 11, 13, 17], forKey: 5)
```

## More Protocol Extension Tricks

Generic beautification

```
extension CollectionType where Index == RandomAccessIndexType,
Generator.Element : Ordered {

  func binarySearch(forKey: Generator.Element) -> Int {
    ...
  }

}

let pos = [2, 3, 5, 7, 11, 13, 17].binarySearch(5)
```

# More Protocol Extension Tricks

## Generic beautification

```
extension CollectionType where Index == RandomAccessIndexType,
Generator.Element : Ordered {

  func binarySearch(forKey: Generator.Element) -> Int {
    ...
  }

}

let pos = [2, 3, 5, 7, 11, 13, 17].binarySearch(5)
```

## More Protocol Extension Tricks

### Generic beautification

```
extension CollectionType where Index == RandomAccessIndexType,
Generator.Element : Ordered {

  func binarySearch(forKey: Generator.Element) -> Int {
    ...
  }

}

let pos = [2, 3, 5, 7, 11, 13, 17].binarySearch(5)
```

# More Protocol Extension Tricks

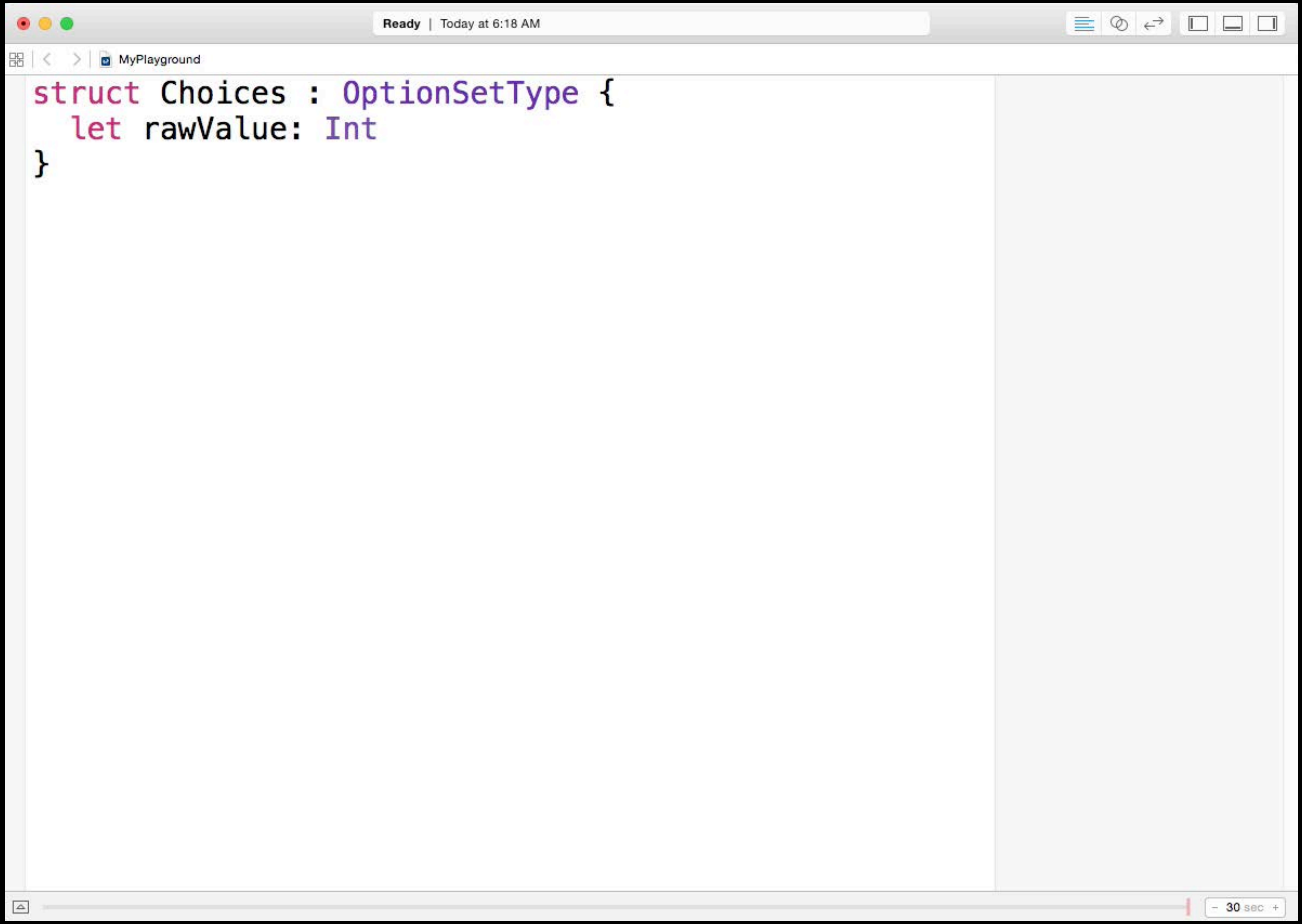
## Generic beautification

```
extension CollectionType where Index == RandomAccessIndexType,
Generator.Element : Ordered {

  func binarySearch(forKey: Generator.Element) -> Int {
    ...
  }

}

let pos = [2, 3, 5, 7, 11, 13, 17].binarySearch(5)
```

# More Protocol Extension Tricks

## Interface generation

NEW

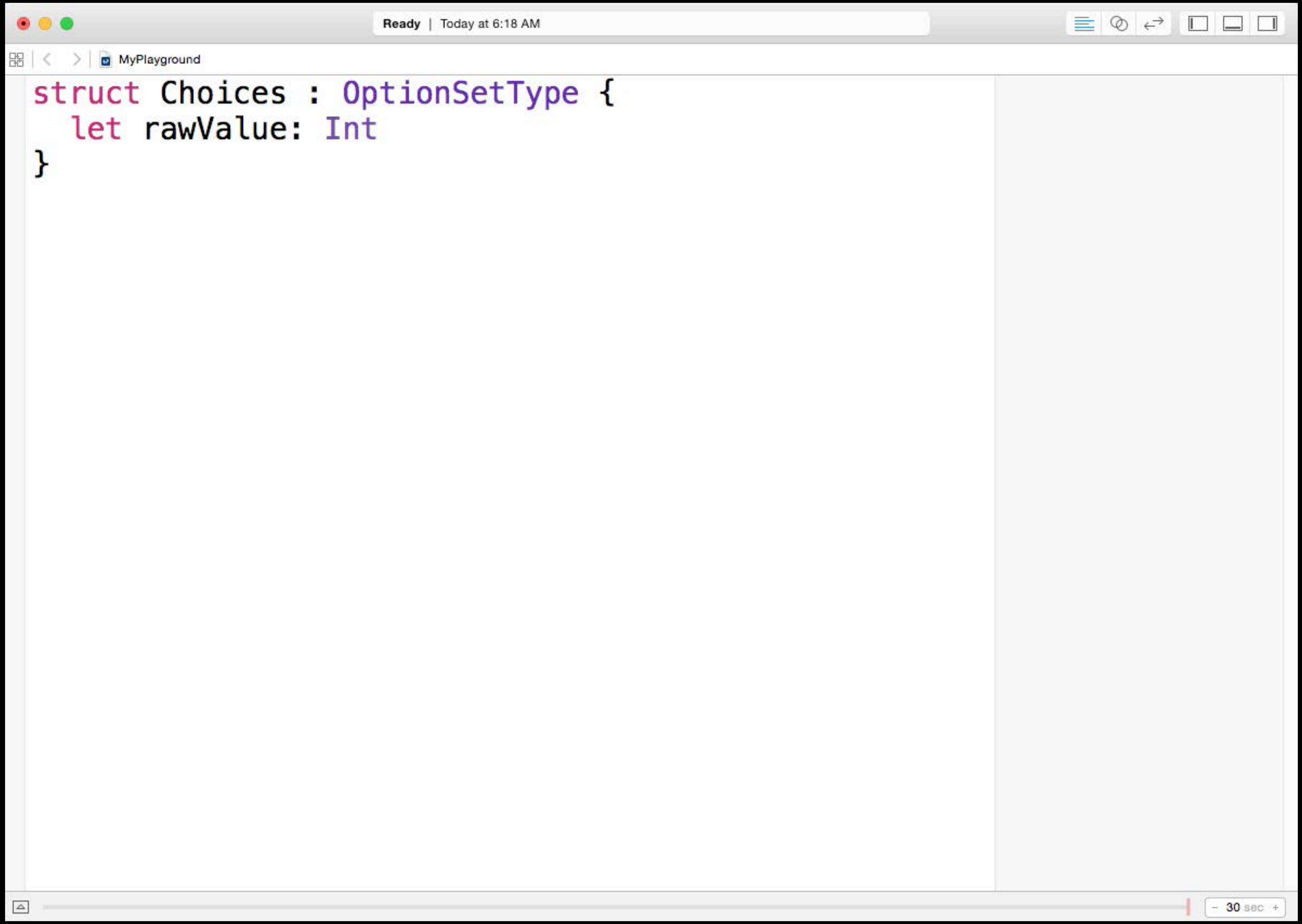


```
struct Choices : OptionSetType {
  let rawValue: Int
}
```

# More Protocol Extension Tricks

NEW

## Interface generation



```swift
struct Choices : OptionSetType {
  let rawValue: Int
}
```

## Make All Value Types `Equatable`

```
func == (lhs: Polygon, rhs: Polygon) -> Bool {
  return lhs.corners == rhs.corners
}
extension Polygon : Equatable {}

func == (lhs: Circle, rhs: Circle) -> Bool {
  return lhs.center == rhs.center
    && lhs.radius == rhs.radius
}
extension Circle : Equatable {}
```

## Make All Value Types `Equatable`

```
func == (lhs: Polygon, rhs: Polygon) -> Bool {
  return lhs.corners == rhs.corners
}
extension Polygon : Equatable {}

func == (lhs: Circle, rhs: Circle) -> Bool {
  return lhs.center == rhs.center
    && lhs.radius == rhs.radius
}
extension Circle : Equatable {}
```

## Make All Value Types `Equatable`

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements == rhs.elements
}
```

## Make All Value Types `Equatable`

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements == rhs.elements
}
```

binary operator '==' cannot be applied to two [Drawable] operands.

## Make All Value Types `Equatable`

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements == rhs.elements
}
```

## Make All Value Types `Equatable`

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { $0 != $1 }
}
```

## Make All Value Types `Equatable`

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { $0 != $1 }
}
```

binary operator '!=' cannot be applied to two Drawable operands.

## Make All Value Types `Equatable`

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { $0 != $1 }
}
```

## Should Every `Drawable` Be `Equatable`?

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { $0 != $1 }
}

protocol Drawable : Equatable {
  func draw()
}
```

## Should Every **Drawable** Be **Equatable**?

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}
```

```
protocol Equatable {
  func == (Self, Self) -> Bool
}
```

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { $0 != $1 }
}

protocol Drawable : Equatable {
  func draw()
}
```

# Should Every Drawable Be Equatable?

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}
```

```
protocol Equatable {
  func == (Self, Self) -> Bool
}
```

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { $0 != $1 }
}
```

```
protocol Drawable : Equatable {
  func draw()
}
```

## Bridge-Building

```
struct Diagram : Drawable, Equatable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}

protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
```

## Bridge-Building

```
struct Diagram : Drawable {
  func draw(renderer: Renderer) { ... }
  var elements: [Drawable] = []
}

func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}

protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
```

## Bridge-Building

```swift
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}
protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
```

## Bridge-Building

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}
protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
extension Drawable where Self : Equatable {
  func isEqualTo(other: Drawable) -> Bool {
    if let o = other as? Self { return self == o }
    return false
  }
}
```

## Bridge-Building

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}
protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
extension Drawable where Self : Equatable {
  func isEqualTo(other: Drawable) -> Bool {
    if let o = other as? Self { return self == o }
    return false
  }
}
```

## Bridge-Building

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}
protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
extension Drawable where Self : Equatable {
  func isEqualTo(other: Drawable) -> Bool {
    if let o = other as? Self { return self == o }
    return false
  }
}
```

# Bridge-Building

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}
protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
extension Drawable where Self : Equatable {
  func isEqualTo(other: Drawable) -> Bool {
    if let o = other as? Self { return self == o }
    return false
  }
}
```

## Bridge-Building

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}
protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
extension Drawable where Self : Equatable {
  func isEqualTo(other: Drawable) -> Bool {
    if let o = other as? Self { return self == o }
    return false
  }
}
```

## Bridge-Building

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}
protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
extension Drawable where Self : Equatable {
  func isEqualTo(other: Drawable) -> Bool {
    if let o = other as? Self { return self == o }
    return false
  }
}
```

## Bridge-Building

```
func == (lhs: Diagram, rhs: Diagram) -> Bool {
  return lhs.elements.count == rhs.elements.count
    && !zip(lhs.elements, rhs.elements).contains { !$0.isEqualTo($1) }
}
protocol Drawable {
  func isEqualTo(other: Drawable) -> Bool
  func draw()
}
extension Drawable where Self : Equatable {
  func isEqualTo(other: Drawable) -> Bool {
    if let o = other as? Self { return self == o }
    return false
  }
}
```

## When to Use Classes

They do have their place…

You *want* implicit sharing when

## When to Use Classes

### They do have their place…

You *want* implicit sharing when

- Copying or comparing instances doesn't make sense (e.g., Window)

## When to Use Classes

They do have their place…

You *want* implicit sharing when

- Copying or comparing instances doesn't make sense (e.g., Window)
- Instance lifetime is tied to external effects (e.g., TemporaryFile)

# When to Use Classes

## They do have their place…

You *want* implicit sharing when

- Copying or comparing instances doesn't make sense (e.g., Window)
- Instance lifetime is tied to external effects (e.g., TemporaryFile)
- Instances are just “sinks”—write-only conduits to external state (e.g., CGContext)

## When to Use Classes

### They do have their place…

You *want* implicit sharing when

- Copying or comparing instances doesn't make sense (e.g., Window)
- Instance lifetime is tied to external effects (e.g., TemporaryFile)
- Instances are just "sinks"—write-only conduits to external state (e.g., CGContext)

```
final class StringRenderer : Renderer {
  var result: String
  ...
}
```

# When to Use Classes

## They do have their place…

You *want* implicit sharing when

- Copying or comparing instances doesn't make sense (e.g., Window)
- Instance lifetime is tied to external effects (e.g., TemporaryFile)
- Instances are just "sinks"—write-only conduits to external state (e.g., CGContext)

```
final class StringRenderer : Renderer {
  var result: String
  ...
}
```

Still a protocol!

## When to Use Classes

They do have their place…

## When to Use Classes

### They do have their place…

Don't fight the system

- If a framework expects you to subclass, or to pass an object, do!

## When to Use Classes

They do have their place…

Don't fight the system

- If a framework expects you to subclass, or to pass an object, do!

On the other hand, be circumspect

- Nothing in software should grow too large
- When factoring something out of a class, consider a non-class

## Summary

## Summary

Protocols > Superclasses

## Summary

Protocols > Superclasses

Protocol extensions = magic (almost)

## Summary

Protocols > Superclasses

Protocol extensions = magic (almost)

Go see the value types talk on Friday!

## Summary

Protocols > Superclasses

Protocol extensions = magic (almost)

Go see the value types talk on Friday!

Eat your vegetables

## Summary

Protocols > Superclasses

Protocol extensions = magic (almost)

Go see the value types talk on Friday!

Eat your vegetables

Be like Crusty…

Think different.

## More Information

Swift Language Documentation
http://developer.apple.com/swift

Apple Developer Forums
http://developer.apple.com/forums

Stefan Lesser
Swift Evangelist
slesser@apple.com

## Related Sessions

| | | |
|---|---|---|
| Building Better Apps with Value Types in Swift | Mission | Friday 2:30PM |

WWDC15