

Protocolos

```
struct Ponto {  
    var x:Double  
    var y:Double  
  
    func area() -> Double {  
        return 0  
    }  
    func igual(a outro:Ponto) -> Bool {  
        return self.x == outro.x && self.y == outro.y  
    }  
    ...  
}
```

```
struct Retangulo {  
    var altura:Double  
    var largura:Double  
    var centro:Ponto  
  
    func area() -> Double {  
        return altura * largura  
    }  
    func igual(a outro:Retangulo) -> Bool {  
        return self.altura == outro.altura  
            && self.largura == outro.largura  
            && self.centro.igual(a:outro.centro)  
    }  
    ...  
}
```

```
struct Circulo {  
    var raio:Double  
    var centro:Ponto  
  
    func area() -> Double {  
        return raio*raio*Double.pi  
    }  
    func igual(a outro:Circulo) -> Bool {  
        return self.raio == outro.raio  
            && self.centro.igual(a:outro.centro)  
    }  
    ...  
}
```

```
var pt1 = Ponto(x:14.0, y:27.0)
var pt2 = Ponto(x:4.0, y:9.0)
var pt3 = Ponto(x:27.0, y:-5.3)
var ret = Circulo(raio:10, centro:pt1)
var circ = Retangulo(altura:7, largura:9, centro:pt2)
var formas:????? = [ret, circ, pt3] // ERRO
```

```
var pt1 = Ponto(x:14.0, y:27.0)
var pt2 = Ponto(x:4.0, y:9.0)
var pt3 = Ponto(x:27.0, y:-5.3)
var circ = Circulo(raio:10, centro:pt1)
var ret = Retangulo(altura:7, largura:9, centro:pt2)
var formas:????? = [ret, circ, pt3] // ERRO
```

```
var areaTotal = 0
for f in formas {
    // Calcula a área. Poderia também desenhar na tela.
    areaTotal += f.area()
}
```

**Protocolos: agrupam tipos
com base em funcionalidades
e propriedades em comum**

*"A protocol defines a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality. The protocol can then be adopted by a class, structure, or enumeration to provide an actual implementation of those requirements. Any type that satisfies the requirements of a protocol is said to **conform** to that protocol."*

```
protocol Forma {  
    func area() -> Double  
}
```

```
struct Ponto:Forma { ... }
```

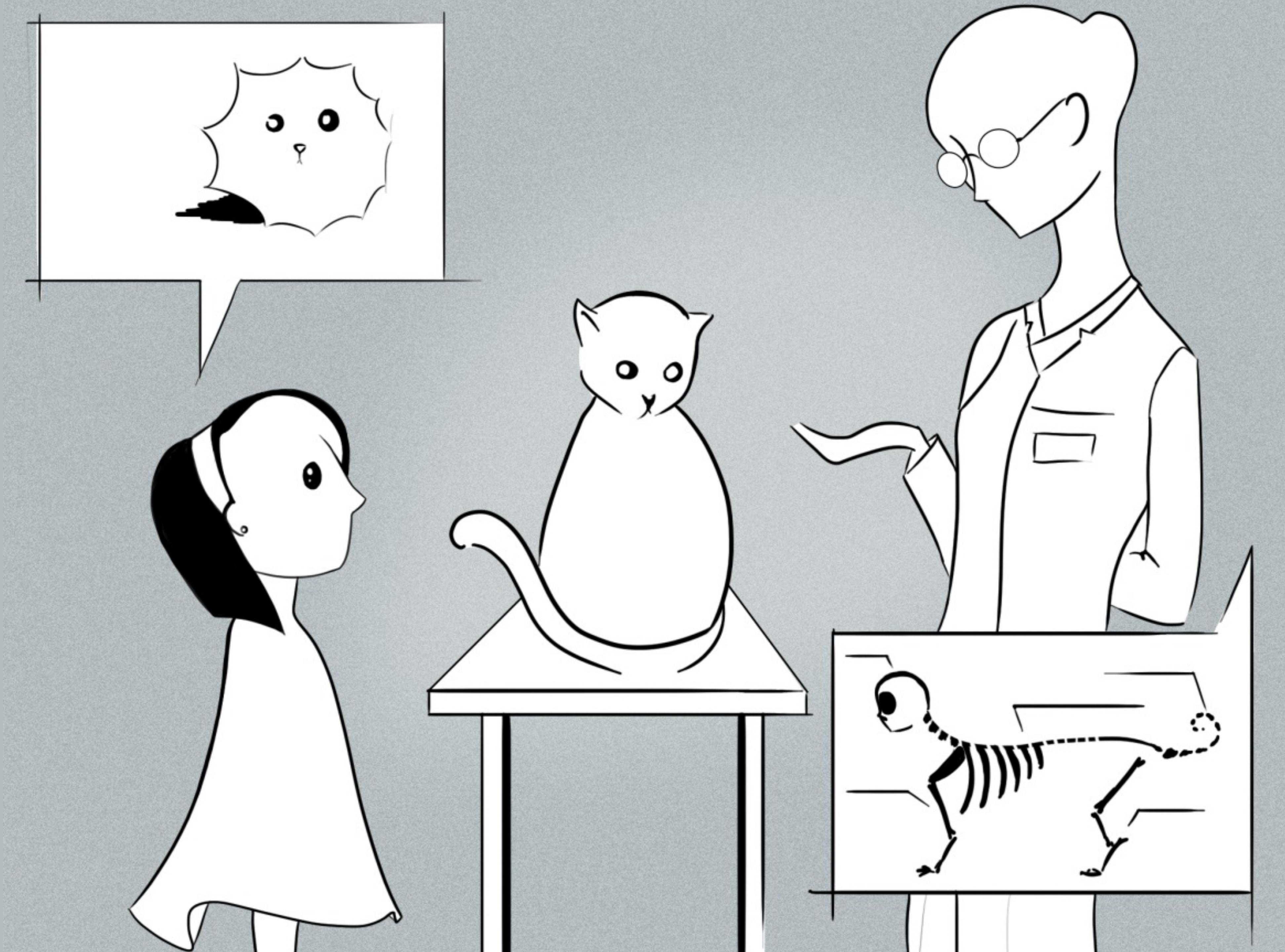
```
struct Retangulo:Forma { ... }
```

```
struct Circulo:Forma { ... }
```

```
var pt1 = Ponto(x:14.0, y:27.0)
var pt2 = Ponto(x:4.0, y:9.0)
var pt3 = Ponto(x:27.0, y:-5.3)
var circ = Circulo(raio:10, centro:pt1)
var ret = Retangulo(altura:7, largura:9, centro:pt2)
// FUNCIONA!
var formas: [Forma] = [ret, circ, pt3]

var areaTotal = 0
for f in formas {
    areaTotal = areaTotal + f.area()
}
```

Protocolos são um **mecanismo de abstração**. Não precisamos saber como cada tipo implementa a rea: .



ABSTRAÇÃO

Várias maneiras de descrever o mesmo objeto



Protocolos também podem incluir propriedades, armazenadas ou computadas.

```
protocol Forma {  
    func area() -> Double  
    func igual(a outra:Forma) -> Bool  
}
```

```
struct Ponto:Forma {  
    ...  
  
    func igual(a outra:Forma) -> Bool {  
        return self.x == outra.x && self.y == outra.y  
    }  
}
```



```
struct Ponto:Forma {  
    ...  
  
    func igual(a outra:Forma) -> Bool {  
        if outra is Ponto {  
            let pto = outra as! Ponto  
            return self.x == pto.x && self.y == pto.y  
        } else {  
            return false  
        }  
    }  
}
```

Faça com que Retangulo e
Circulo também estejam em
conformidade com Forma.

Adicione uma função
mover: dx: dy: a cada forma
geométrica. Essa função move
cada forma em dx unidades
horizontais e dy verticais.



```
struct Ponto:Forma {  
    ...  
  
    mutating func mover(dx:Double, dy:Double) {  
        self.x += dx // Essas linhas só são aceitas se  
        self.y += dy // a função for mutating.  
    }  
}
```

```
protocol Forma {  
    func area() -> Double  
    func igual(a outra:Forma) -> Bool  
    mutating func mover(dx:Double, dy:Double)  
}
```

Adicione a seguinte função a Forma e a Retangulo, Circulo e Ponto:

```
func incluiPonto(_ ponto:Ponto) -> Bool
```

Essa função deve verificar se, dado um ponto, esse ponto está dentro ou fora da superfície da forma geométrica.

Adicione a seguinte função a Forma e a Retangulo, Circulo e Ponto:

```
func incluiForma(_ outra:Forma) -> Bool
```

Essa função deve verificar se a forma recebida como argumento está inclusa na forma alvo da função.