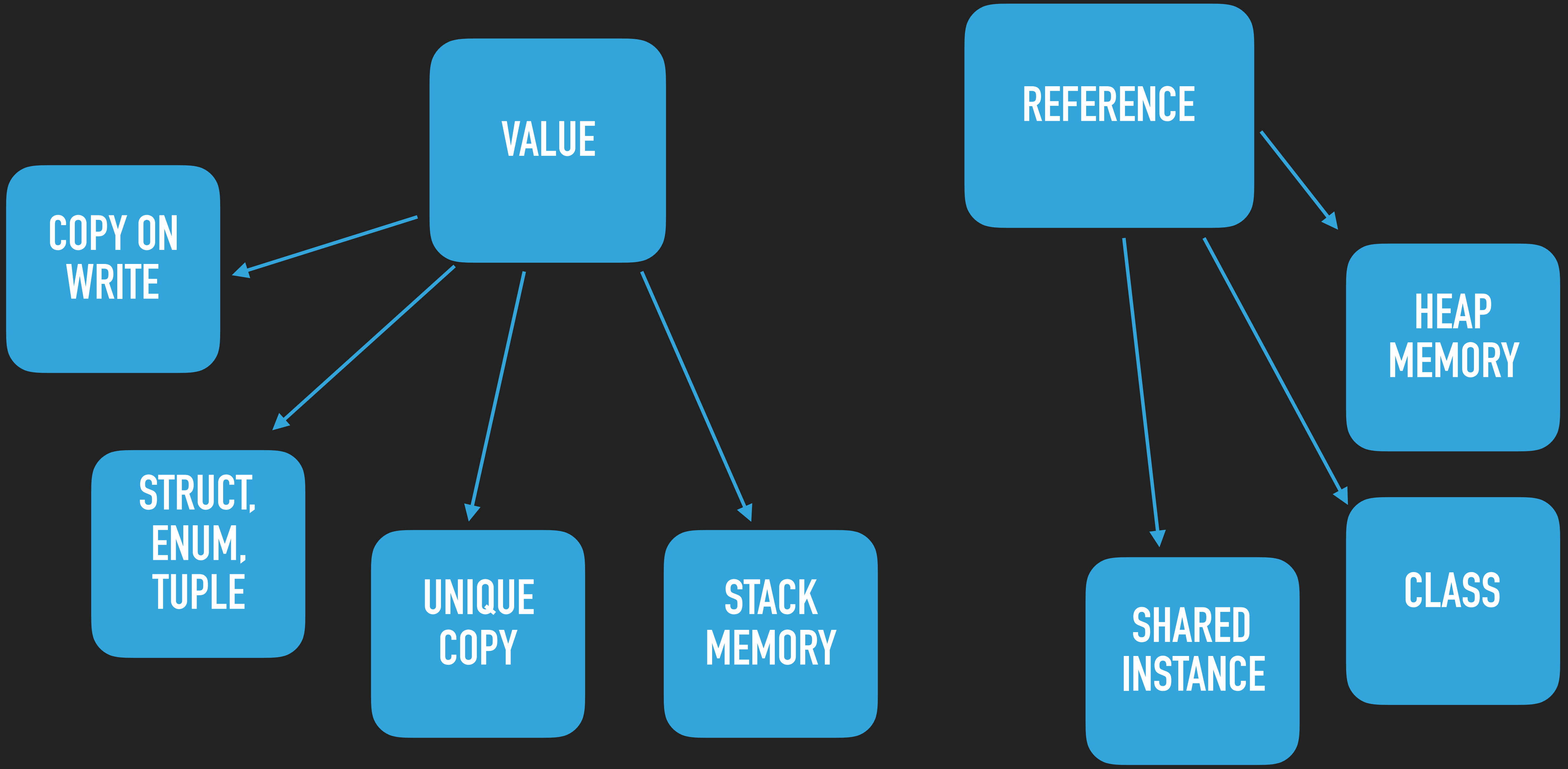
A.STEFANCHUK IT STEP

SWIFT CUSTOM TYPES

CUSTOM TYPES

- Enums
- Structs
- Classes

VALUE VS REFERENCE TYPES



VALUE

COPY ON WRITE

STRUCT, ENUM, TUPLE

UNIQUE COPY

STACK MEMORY

REFERENCE

HEAP MEMORY

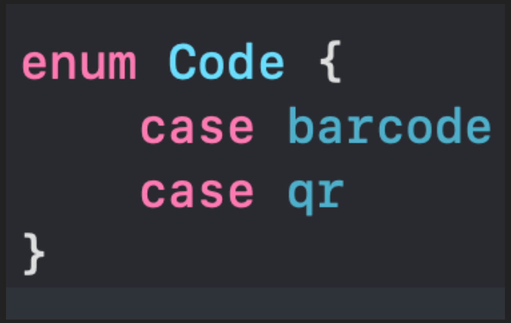
CLASS

SHARED INSTANCE

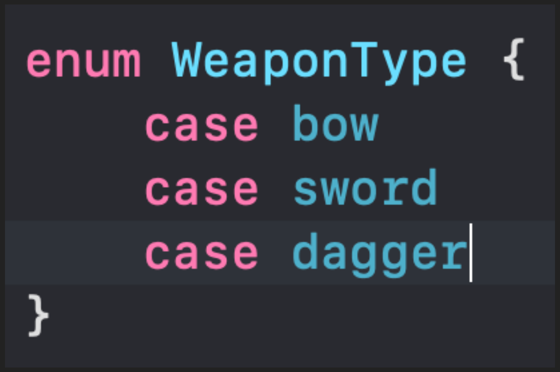
ENUMS

- Nice thing for defining set of fixed named values, not using strings or ints
- Can have associated values
- They have computed properties for additional info
- They have functions
- Can conform to protocols
- Value type
- Can’t store properties

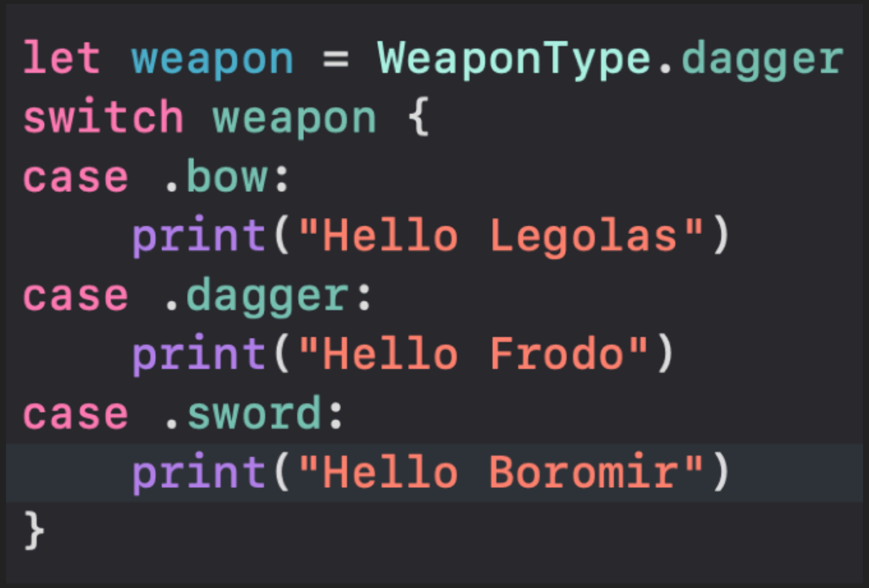
ENUMS



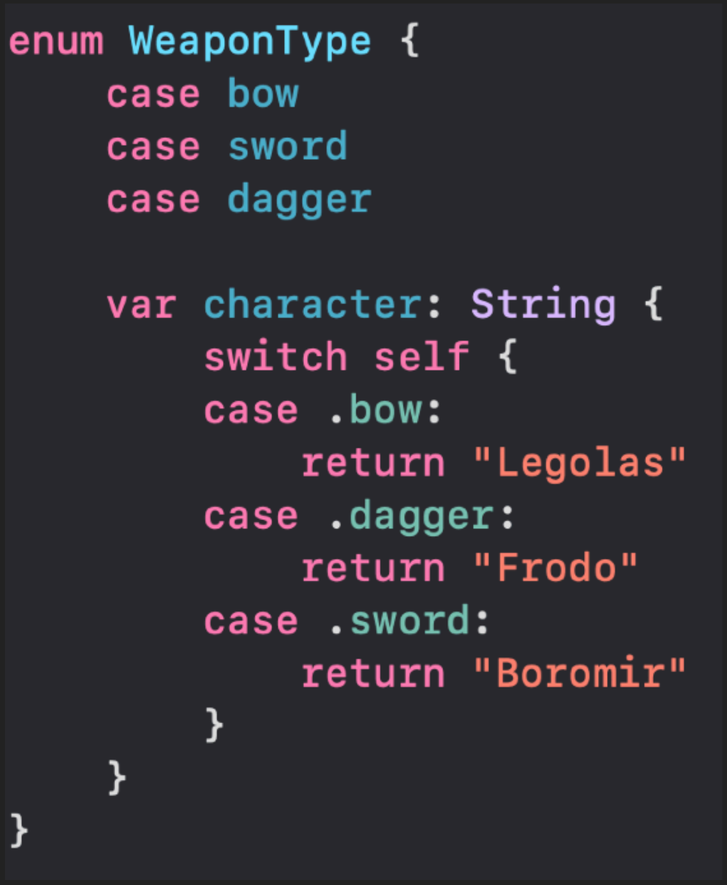
```swift
enum Code {
    case barcode
    case qr
}
```



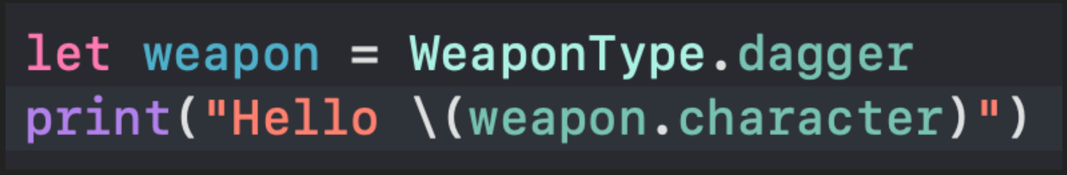
```swift
enum WeaponType {
    case bow
    case sword
    case dagger
}
```



```swift
let weapon = WeaponType.dagger
switch weapon {
case .bow:
    print("Hello Legolas")
case .dagger:
    print("Hello Frodo")
case .sword:
    print("Hello Boromir")
}
```

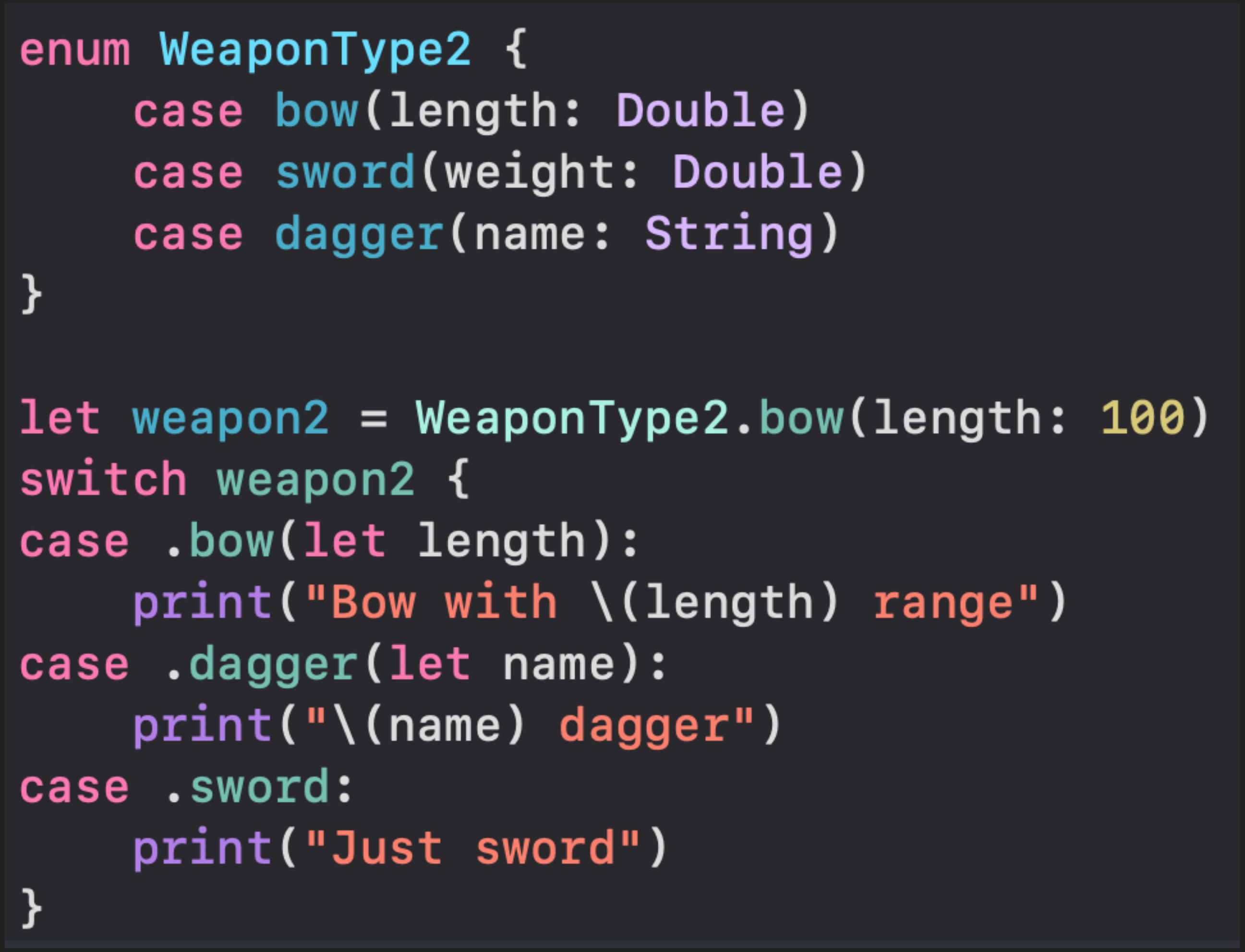


```swift
enum WeaponType {
    case bow
    case sword
    case dagger

    var character: String {
        switch self {
        case .bow:
            return "Legolas"
        case .dagger:
            return "Frodo"
        case .sword:
            return "Boromir"
        }
    }
}
```

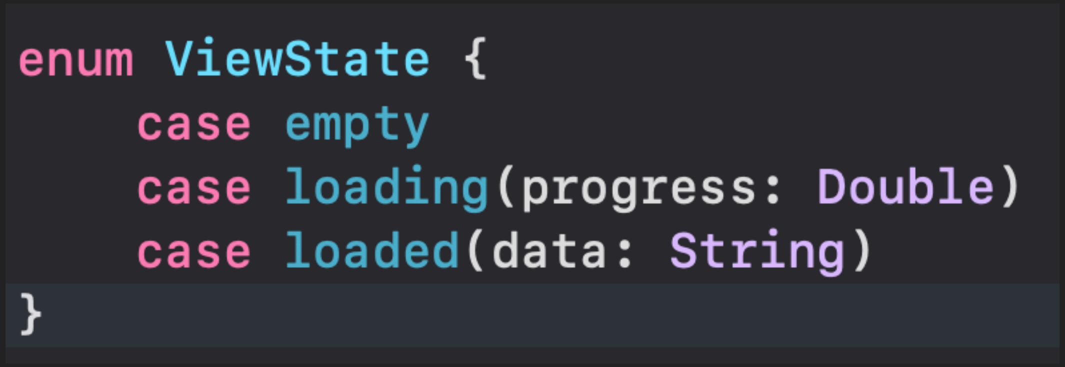


```swift
let weapon = WeaponType.dagger
print("Hello \(weapon.character)")
```

ENUMS ASSOCIATED VALUES



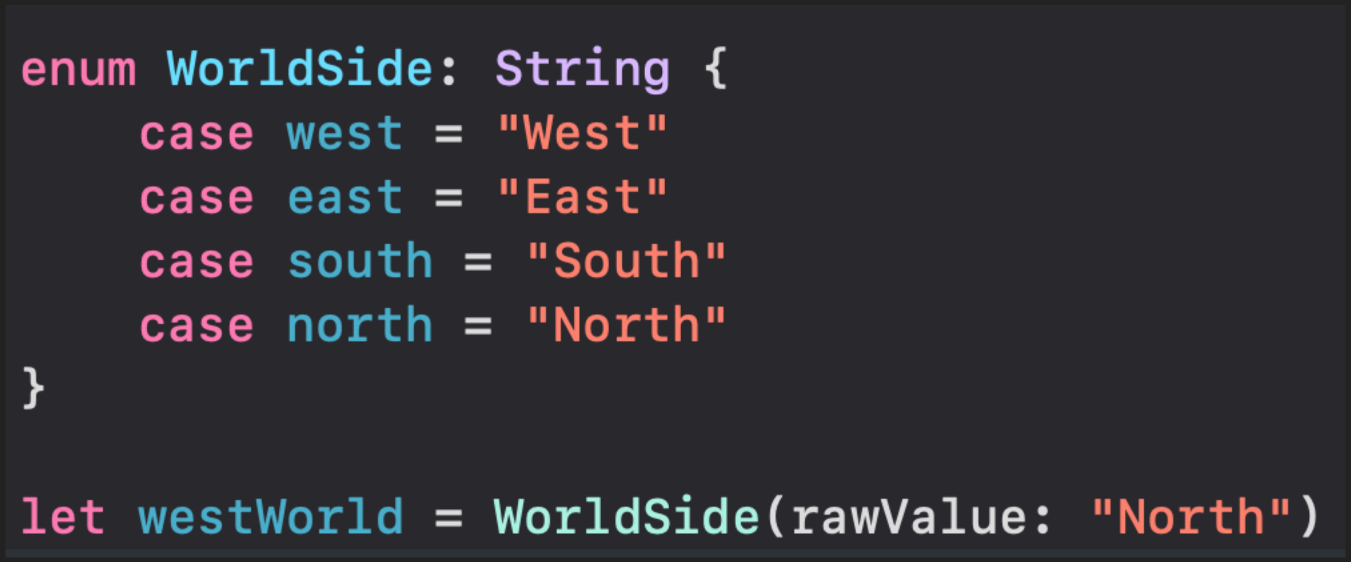
```swift
enum WeaponType2 {
    case bow(length: Double)
    case sword(weight: Double)
    case dagger(name: String)
}

let weapon2 = WeaponType2.bow(length: 100)
switch weapon2 {
case .bow(let length):
    print("Bow with \(length) range")
case .dagger(let name):
    print("\(name) dagger")
case .sword:
    print("Just sword")
}
```



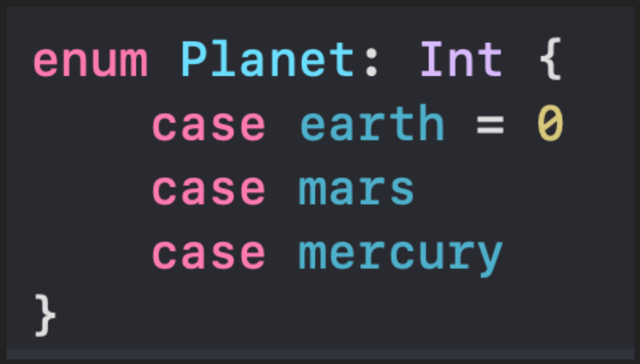
```swift
enum ViewState {
    case empty
    case loading(progress: Double)
    case loaded(data: String)
}
```

ENUMS RAW VALUES

- Each case can have some associated value (string or Int)
- We can init with that value



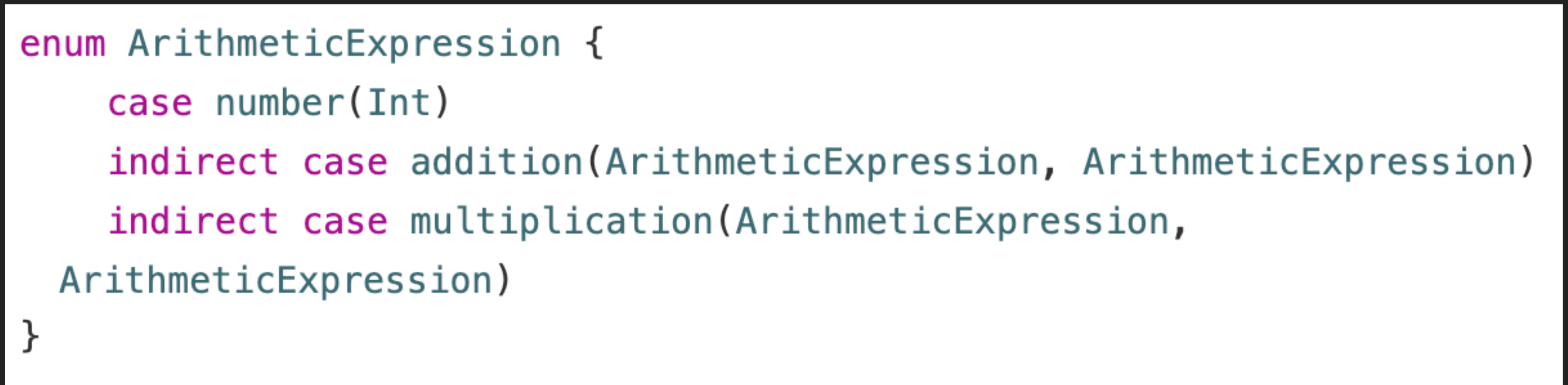
```
enum WorldSide: String {
    case west = "West"
    case east = "East"
    case south = "South"
    case north = "North"
}

let westWorld = WorldSide(rawValue: "North")
```



```
enum Planet: Int {
    case earth = 0
    case mars
    case mercury
}
```

RECURSIVE ENUMS

- Enums that can have associated value of their own type



```
enum ArithmeticExpression {
    case number(Int)
    indirect case addition(ArithmeticExpression, ArithmeticExpression)
    indirect case multiplication(ArithmeticExpression,
  ArithmeticExpression)
}
```

STRUCTS VS CLASS

- Both structs and class have:
- Properties
- Methods
- Subscripts
- Initializers
- Be extended with extensions
- Conform to protocols

STRUCTS VS CLASS - DIFFERENCES

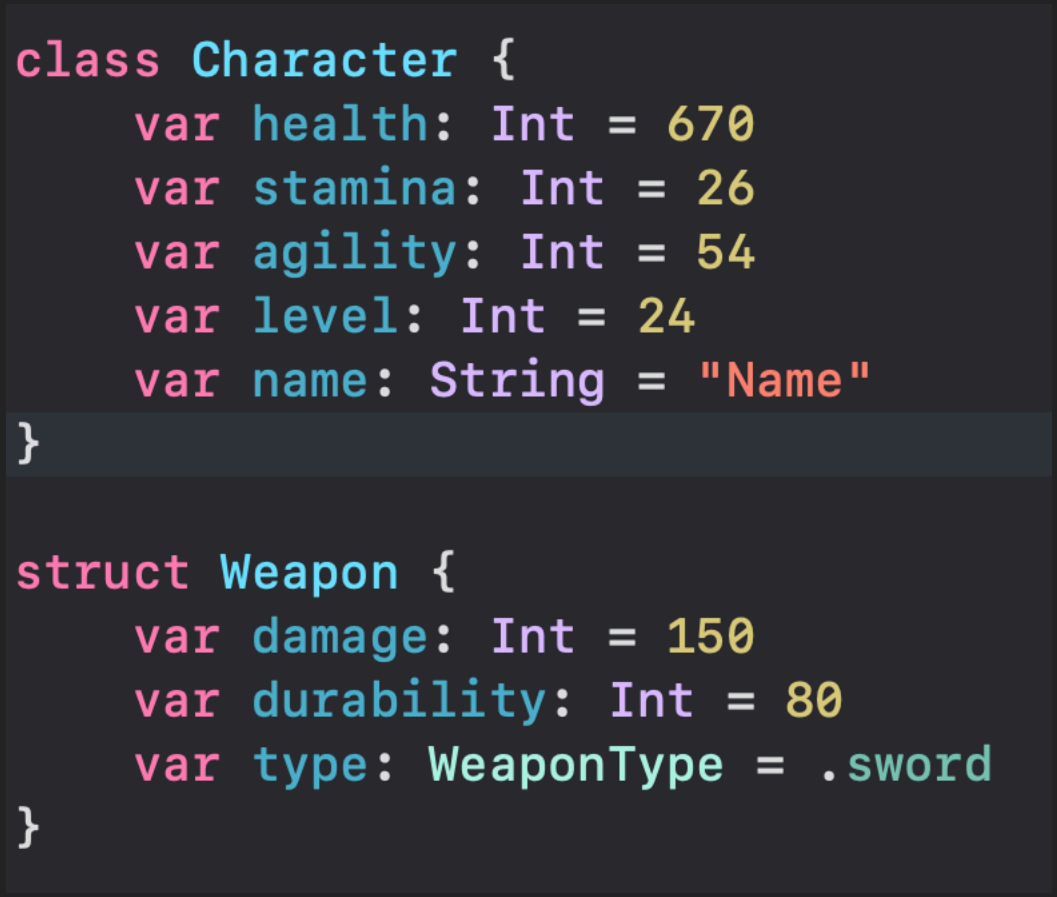
- Struct is **value** type, Class is **reference** type
- Classes have inheritance
- Classes have type casting
- Classes have deinit

Firstly, choose structs for your data models

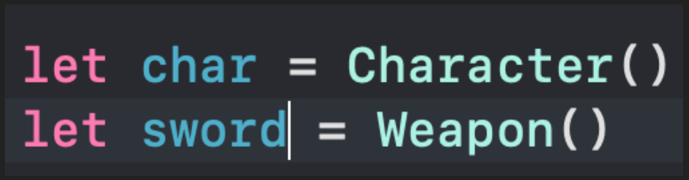
Then switch it class if needed

STRUCTS VS CLASS - CREATION

- Looks pretty much the same



```swift
class Character {
    var health: Int = 670
    var stamina: Int = 26
    var agility: Int = 54
    var level: Int = 24
    var name: String = "Name"
}

struct Weapon {
    var damage: Int = 150
    var durability: Int = 80
    var type: WeaponType = .sword
}
```

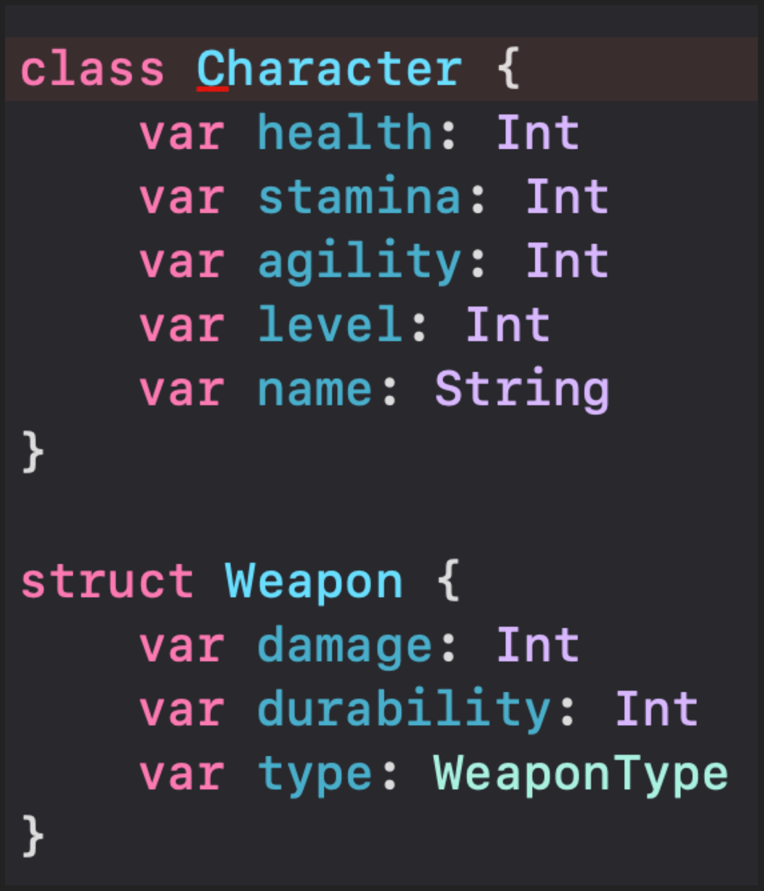


```swift
let char = Character()
let sword = Weapon()
```

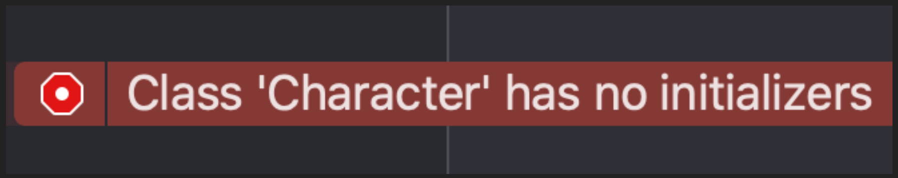
At the moment of creation all variables should have values

STRUCTS VS CLASS - CREATION

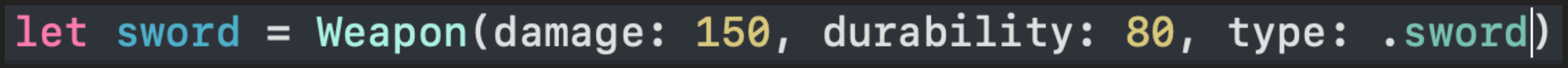
- We get free init for structs



```
class Character {
    var health: Int
    var stamina: Int
    var agility: Int
    var level: Int
    var name: String
}

struct Weapon {
    var damage: Int
    var durability: Int
    var type: WeaponType
}
```



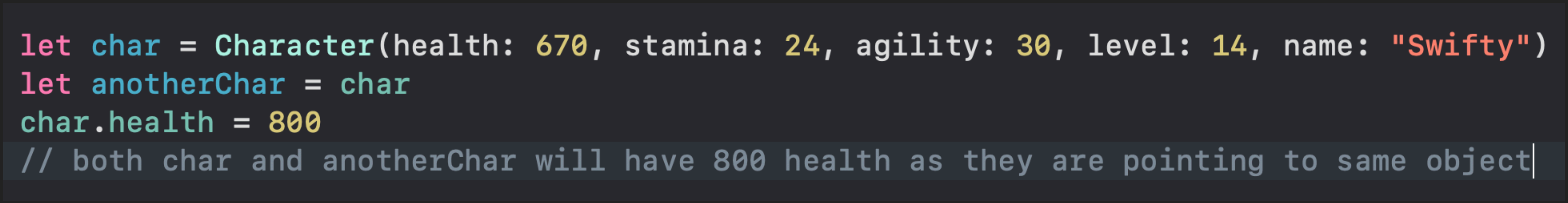
Class 'Character' has no initializers



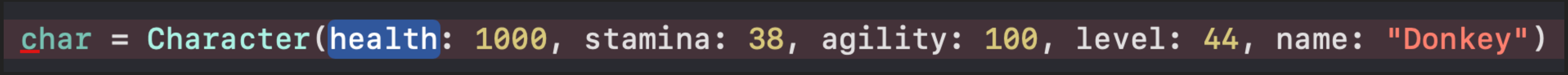
```
let sword = Weapon(damage: 150, durability: 80, type: .sword)
```

REFERENCE TYPE

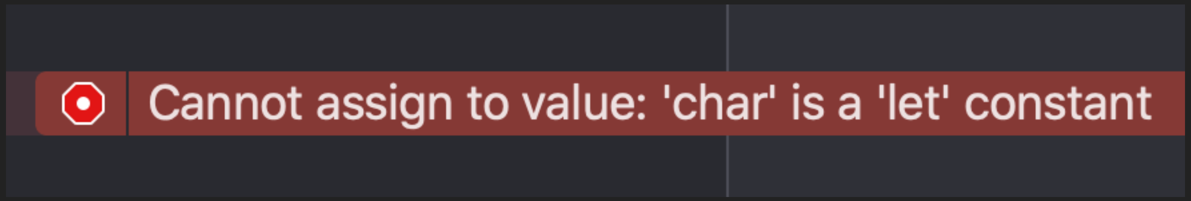
- It creates shared instance



```
let char = Character(health: 670, stamina: 24, agility: 30, level: 14, name: "Swifty")
let anotherChar = char
char.health = 800
// both char and anotherChar will have 800 health as they are pointing to same object
```



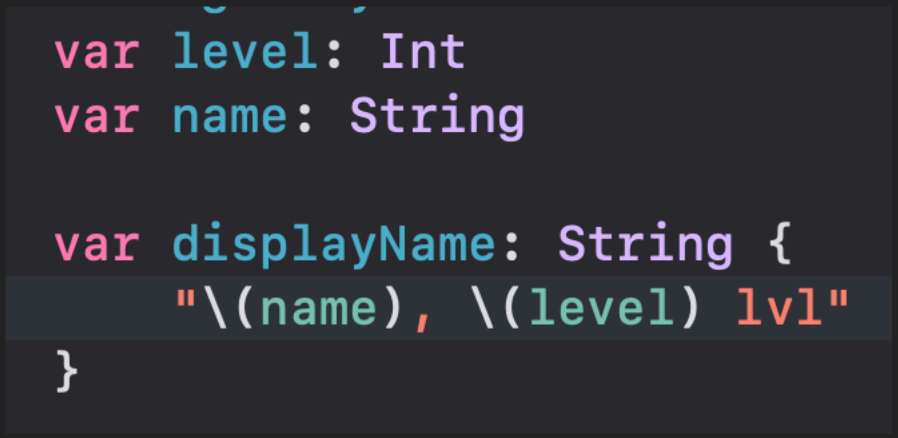
```
char = Character(health: 1000, stamina: 38, agility: 100, level: 44, name: "Donkey")
```



Cannot assign to value: 'char' is a 'let' constant

PROPERTIES

- Stored - store information about object
- Calculated - calculate value based on



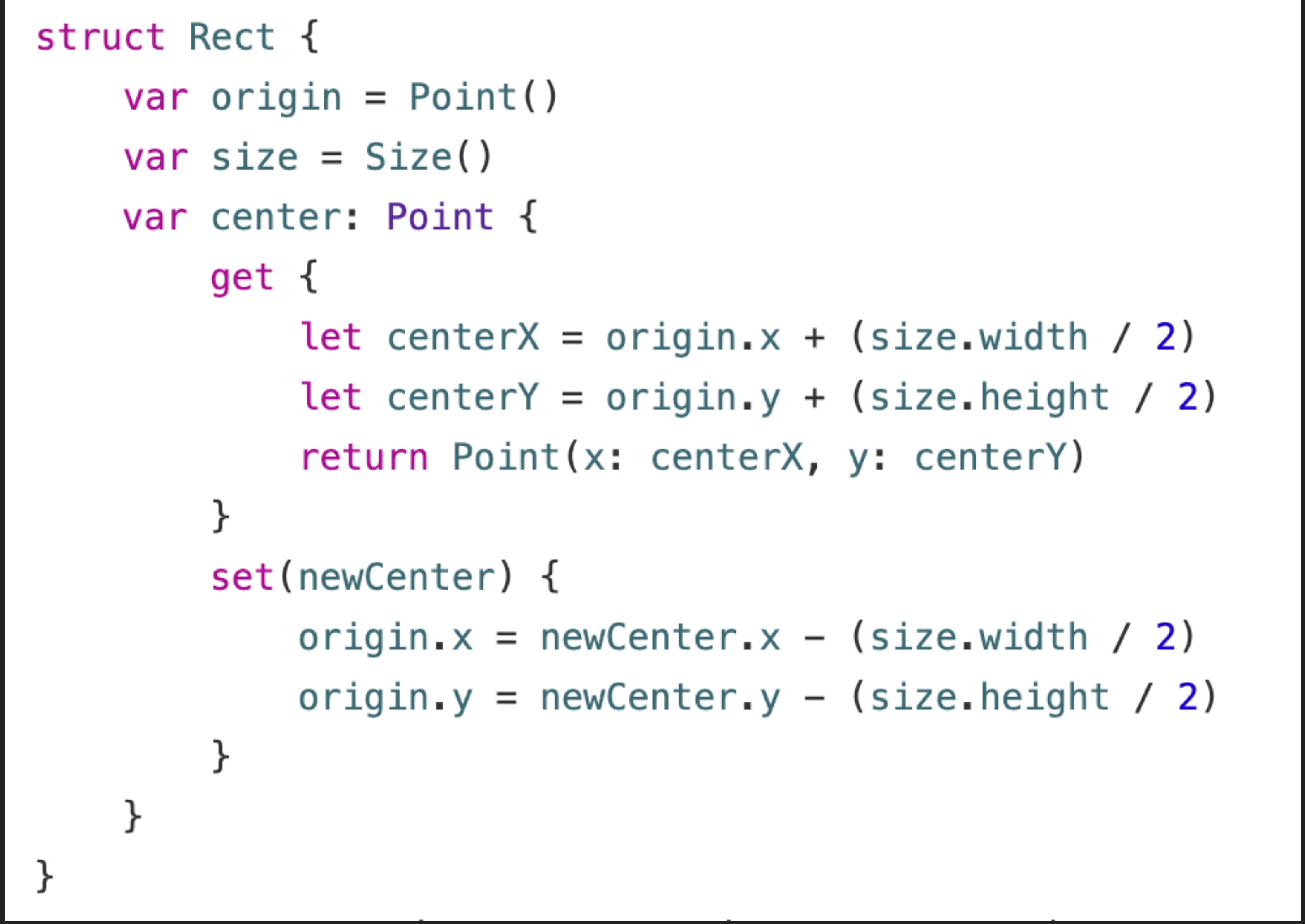
```
var level: Int
var name: String

var displayName: String {
    "\(name), \(level) lvl"
}
```

- lazy - not calculated until you use them for the first time

Use it for something that needs a
lot of resources

PROPERTIES

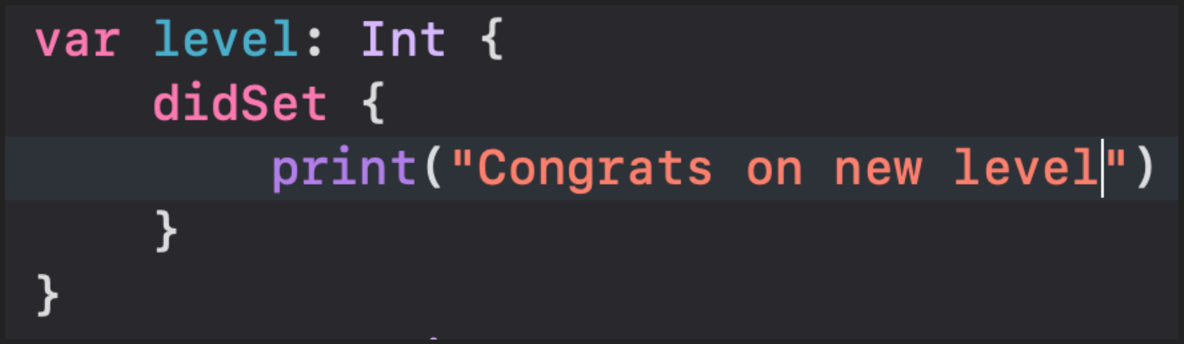
- Calculated properties have getter and setter



```
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set(newCenter) {
            origin.x = newCenter.x - (size.width / 2)
            origin.y = newCenter.y - (size.height / 2)
        }
    }
}
```

PROPERTIES

- You can observe change of property with didSet and willSet



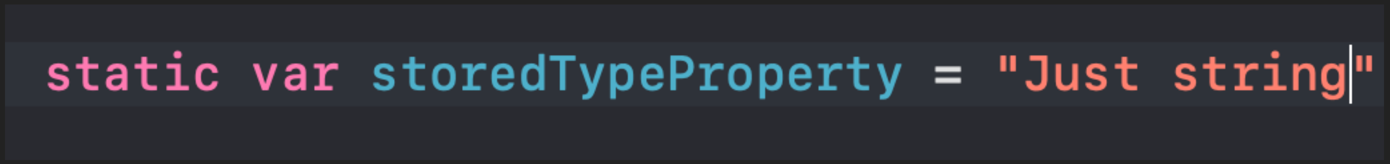
```
var level: Int {
    didSet {
        print("Congrats on new level")
    }
}
```

It’s used a lot

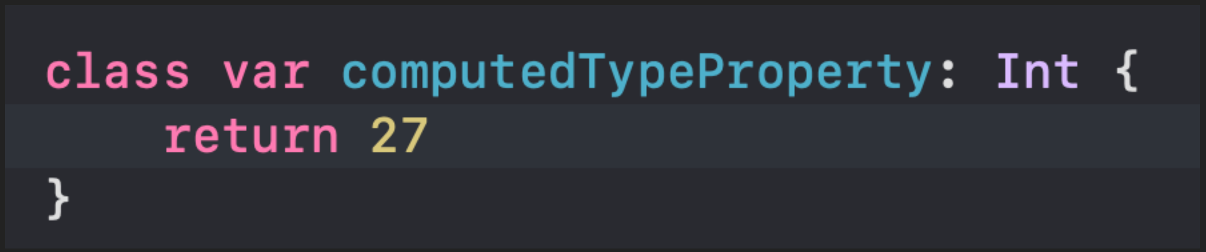
- **Property wrappers** allow us add a layer of separation between code that manages how a property is stored and code that defines a property

PROPERTIES

- Type can have own properties - static properties



```swift
static var storedTypeProperty = "Just string"
```



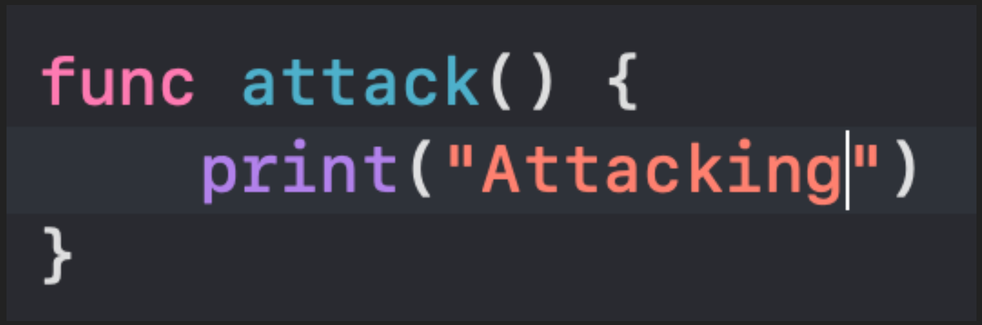
```swift
class var computedTypeProperty: Int {
    return 27
}
```

class properties can be overriden, static - no

Static things are good for constants

METHODS

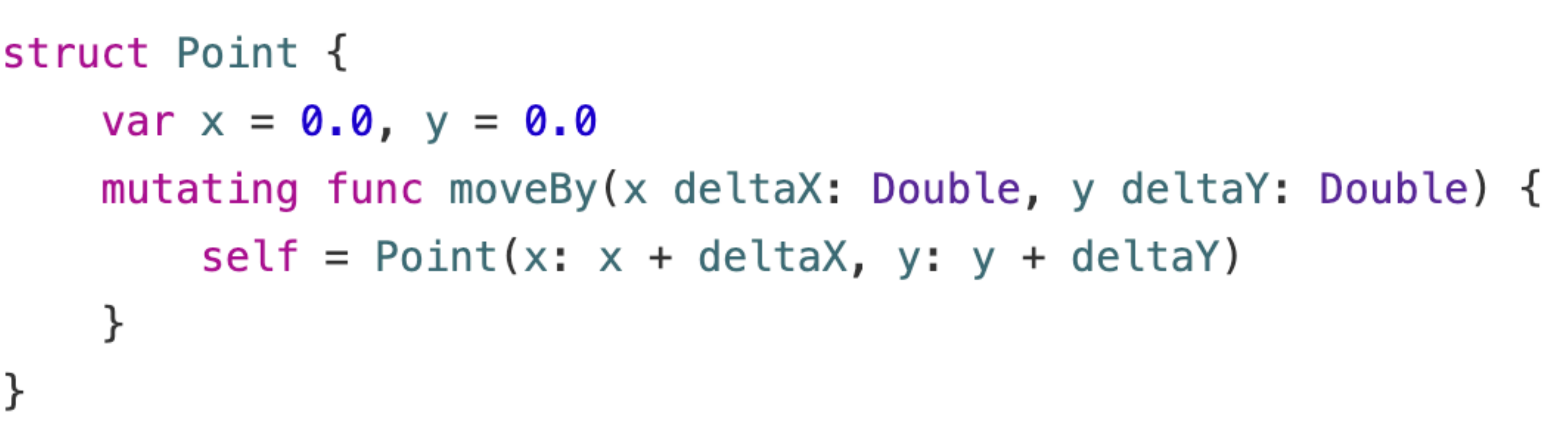
- Methods define class behaviour
- You can also have instance and type methods



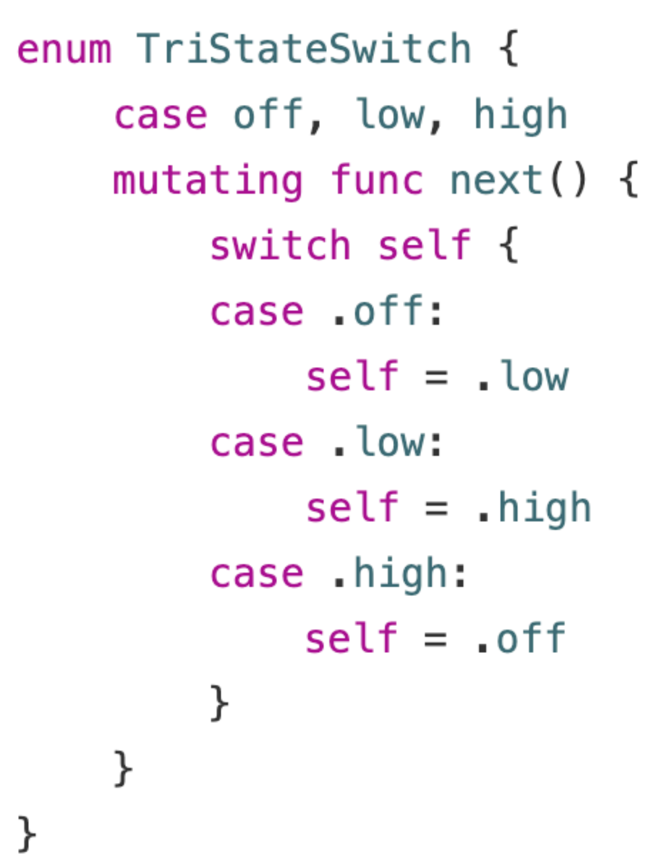
```swift
func attack() {
    print("Attacking")
}
```

If function is modifying property value in struct it should be marked mutating

MUTATING METHODS



```swift
struct Point {
    var x = 0.0, y = 0.0
    mutating func moveBy(x deltaX: Double, y deltaY: Double) {
        self = Point(x: x + deltaX, y: y + deltaY)
    }
}
```



```swift
enum TriStateSwitch {
    case off, low, high
    mutating func next() {
        switch self {
        case .off:
            self = .low
        case .low:
            self = .high
        case .high:
            self = .off
        }
    }
}
```