# **OOP in Swift**

Defining a class

Initializing a class (constructors)

Creating class properties

* Stored properties
* Computed properties

Adding class functions

Inheritance

**Object Oriented Programming in Swift**

* Swift provides a variety of tools to enable you to develop applications using object oriented programming
* Examples:
  + Structs *lightweight data structures, similar to C style structs*
  + Classes *traditional “class” from most oop languages*
  + Protocols *similar to interfaces, but offers more flexibility than a traditional interface*
  + Enumerations
* Many of these structures work in a similar way to their counterparts in other object oriented programming languages, but with extra features

**Classes**

Defining the class

class Dog {

// properties

// initialize the class

// functions (methods)

}

Defining the object’s attributes:

class Dog {

// properties

// 1. stored property (this is the traidtional "property" that you're used to seeing)

// 2. computed property (this is unique to swift/kotlin)

// 1. stored properties

// --------------------

var name:String

var age:Int

var breed:String

var color:String

var isSleeping:Bool

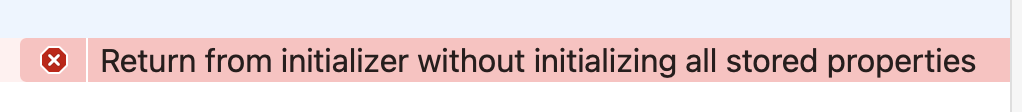
// initialize the class

// functions (methods)

}

Defining the class initializer

* This is the same as a constructor in other languages



One of the requirements of a class in Swift, is that all of your class properties must be assigned an initial value by the time the object is created.

* When is the object created? After the constructor runs (initializer finishes executing, you have a new object)

To solve this issue, you must ensure that all your class *stored* properties are assigned a default value

OPTION 1: Set the default value in the initializer

class Dog {

// properties

// 1. stored property (this is the traidtional "property" that you're used to seeing)

// 2. computed property (this is unique to swift/kotlin)

// 1. stored properties

// --------------------

var name:String

var age:Int

var breed:String

var isSleeping:Bool

// 2. initialize the class

init() {

self.name = "Polo"

self.age = 5

self.breed = "Poodle"

self.isSleeping = false

}

// functions (methods)

}

OPTION 2:

class Dog {

// properties

// 1. stored property (this is the traidtional "property" that you're used to seeing)

// 2. computed property (this is unique to swift/kotlin)

// 1. stored properties

// --------------------

var name:String = "Polo"

var age:Int = 5

var breed:String = "Poodle"

var isSleeping:Bool = false

// 2. initialize the class

init() {

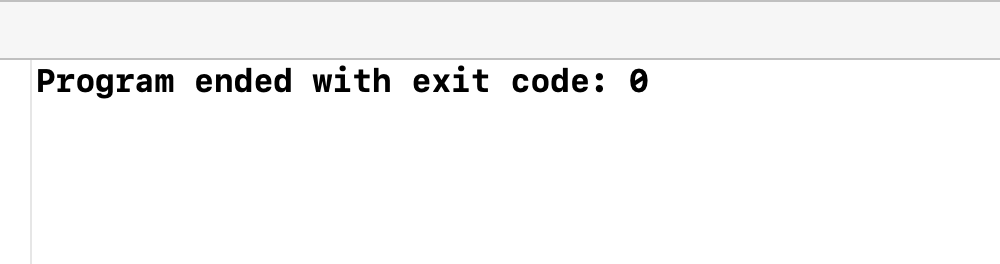
}

// functions (methods)

}

Expected result of both option 1 and option 2:

* Program compiles with no errors



# Creating a new instance of a dog and accessing its properties

There is no “new” keyword!

let d1 = Dog()

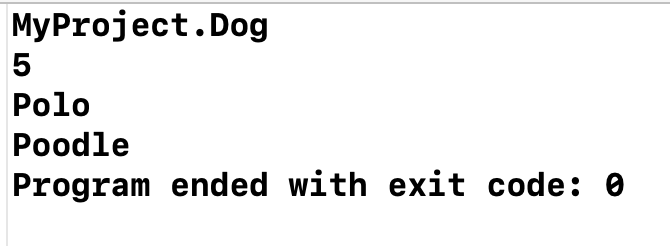
print(d1)

print(d1.age)

print(d1.name)

print(d1.breed)

Expected result:



# 

# OOP - Encapsulation, Abstraction

*Marking variables as private -> getters and setters*

class Dog {

// properties

// 1. stored property (this is the traidtional "property" that you're used to seeing)

// 2. computed property (this is unique to swift/kotlin)

// 1. stored properties

// --------------------

private var name:String = "Polo"

var age:Int = 5

var breed:String = "Poodle"

var isSleeping:Bool = false

// 2. initialize the class

init() {

}

// functions (methods)

}

// lets write our main program

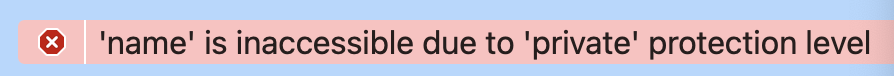
let d1 = Dog()

print(d1)

print(d1.age)

print(d1.name)

print(d1.breed)



Custom initializers

class Dog {

// properties

// 1. stored property (this is the traidtional "property" that you're used to seeing)

// 2. computed property (this is unique to swift/kotlin)

// 1. stored properties

// --------------------

var name:String = "Polo"

var age:Int = 5

var breed:String = "Poodle"

var isSleeping:Bool = false

// 2. initialize the class

init() {

}

// custom initializer

init(n:String, b:String) {

self.name = n

self.breed = b

}

init(name:String, age:Int, breed:String, isSleeping:Bool) {

self.name = name

self.age = age

self.breed = breed

self.isSleeping = isSleeping

}

// functions (methods)

}

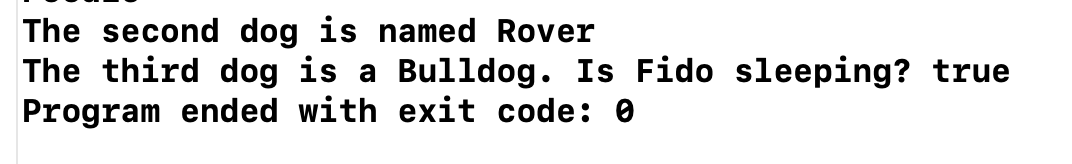
let d2 = Dog(n: "Rover", b: "Golden Retriever")

print("The second dog is named \(d2.name)")

let d3 = Dog(name: "Fido", age: 4, breed: "Bulldog", isSleeping: true)

print("The third dog is a \(d3.breed). Is \(d3.name) sleeping? \(d3.isSleeping)")

Expected result:



# Class functions

To add a class function, define the function within the class

class Dog {

// properties

// 1. stored property (this is the traidtional "property" that you're used to seeing)

// 2. computed property (this is unique to swift/kotlin)

// 1. stored properties

// --------------------

var name:String = "Polo"

var age:Int = 5

var breed:String = "Poodle"

var isSleeping:Bool = false

// 2. initialize the class

init() {

}

// custom initializer

init(n:String, b:String) {

self.name = n

self.breed = b

}

init(name:String, age:Int, breed:String, isSleeping:Bool) {

self.name = name

self.age = age

self.breed = breed

self.isSleeping = isSleeping

}

// functions (methods)

func sayHello() {

print("WOOF! WOOF!")

}

// functions accept parameters

func eatingDinner(favFood:String) {

print("\(self.name) is eating \(favFood)")

}

// function that returns a value

func wantsToPlay() -> Bool {

if (self.isSleeping == false) {

return true

}

else {

return false

}

}

}

let d3 = Dog(name: "Fido", age: 4, breed: "Bulldog", isSleeping: true)

print("The third dog is a \(d3.breed). Is \(d3.name) sleeping? \(d3.isSleeping)")

d3.sayHello()

d3.eatingDinner(favFood: "steak")

d3.eatingDinner(favFood: "apple")

d3.eatingDinner(favFood: "cheese")

d3.eatingDinner(favFood: "dog snacks")

d3.eatingDinner(favFood: "yogurt")

if (d3.wantsToPlay()) {

print("Let's take \(d3.name) to the park!")

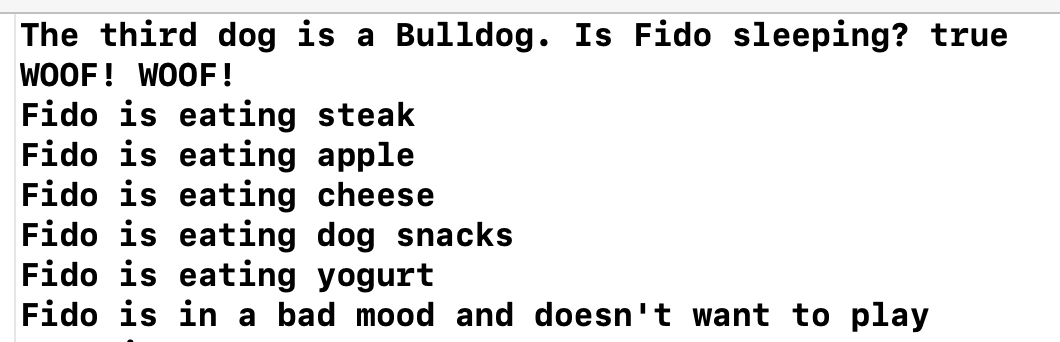
}

else {

print("\(d3.name) is in a bad mood and doesn't want to play")

}

Expected result:



What is a computed property?

* A property that performs a calculation and returns the result
* Does NOT store a value
* (stores a calculation!)

Traditional OOP tools:

Example: Create a function to calculate and return the dog’s age in human years

* 1 dog year = 15 human years

func getAgeInHumanYears() -> Int {

// 1 dog year = 7 human years

return self.age \* 7

}

let d3 = Dog(name: "Fido", age: 4, breed: "Bulldog", isSleeping: true)

print("How old is Fido in human years? \(d3.getAgeInHumanYears())")

How to do it the Swift way:

class Dog {

// properties

// 1. stored property (this is the traidtional "property" that you're used to seeing)

// 2. computed property (this is unique to swift/kotlin)

// 1. stored properties

// --------------------

var name:String = "Polo"

var age:Int = 5

var breed:String = "Poodle"

var isSleeping:Bool = false

// 2. Computed property

// ----------------------

var ageInHumanYears:Int {

// this is the value that will be returned when someone tries to access this property

get {

return self.age \* 7

}

}

// 2. initialize the class

init() {

}

// custom initializer

init(n:String, b:String) {

self.name = n

self.breed = b

}

init(name:String, age:Int, breed:String, isSleeping:Bool) {

self.name = name

self.age = age

self.breed = breed

self.isSleeping = isSleeping

}

// functions (methods)

func sayHello() {

print("WOOF! WOOF!")

}

// functions accept parameters

func eatingDinner(favFood:String) {

print("\(self.name) is eating \(favFood)")

}

// function that returns a value

func wantsToPlay() -> Bool {

// OPTION 1: Using an if-else statement

if (self.isSleeping == false) {

return true

}

else {

return false

}

// OPTION 2: you can just do this

// return !self.isSleeping

}

// func getAgeInHumanYears() -> Int {

// // 1 dog year = 7 human years

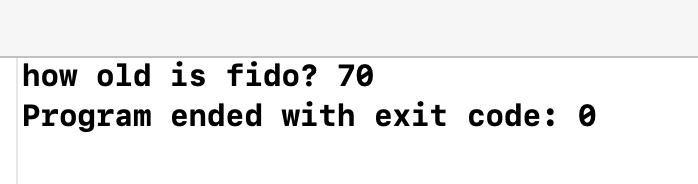
// return self.age \* 7

// }

}

print("how old is fido? \(d3.ageInHumanYears)")

Expected result:



Traditional definition of a Circle

// - diameter

class Circle {

var radius:Double

init(r:Double) {

self.radius = r

}

func getArea() -> Double {

// r \* r \* pi

return self.radius \* self.radius \* Double.pi

}

func getDiameter() -> Double {

// 2r

return 2 \* self.radius

}

func getCircumference() -> Double {

// 2 \* pi \* ri

return 2 \* Double.pi \* self.radius

}

}

var c1 = Circle(r: 5)

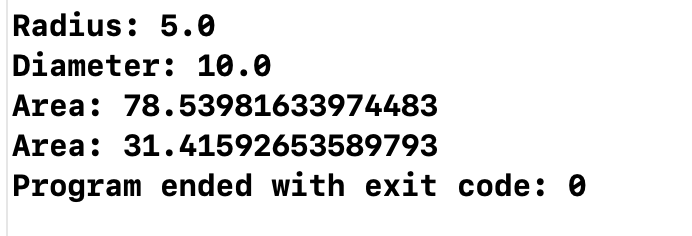
print("Radius: \(c1.radius)")

print("Diameter: \(c1.getDiameter())")

print("Area: \(c1.getArea())")

print("Circumferece: \(c1.getCircumference())")

Expected result:



Rewriting the class using computed properties

class Circle {

// stored property

var radius:Double

// computed property

var diameter:Double {

return 2 \* self.radius

}

var area:Double {

return self.radius \* self.radius \* Double.pi

}

// this syntax is the "long version" of the above syntax

var circumference:Double {

get {

return 2 \* Double.pi \* self.radius

}

}

init(r:Double) {

self.radius = r

}

}

var c1 = Circle(r: 5)

print("Radius: \(c1.radius)")

print("Diameter: \(c1.diameter)")

print("Area: \(c1.area)")

print("Circumference: \(c1.circumference)")

