Protocols

```
protocol MyProtocol {
        //protocol definition here
}
```

Protocols

```
protocol MyProtocol {
        //protocol definition here
}

class myClass: MyProtocol {
        //class implementation here
}
```

Protocols

```
class MyClass: MyProtocol, AnotherProtocol, ThirdProtocol {
        // class implementation here
}
```

Protocols

```
Class MyClass: MySuperClass, MyProtocol, MyProtocol2 {
        // Class implementation here
}
```

Protocols

```
Class MyClass: MySuperClass, MyProtocol, MyProtocol2 {
        // Class implementation here
}
```

Protocols - Properties

```
protocol FullName {
        var firstName: String {get set}
        var lastName: String {get set}
}
```

Protocols - Properties

```
protocol FullName {
        var firstName: String {get}
        var lastName: String {get set}
}
```

Protocols - Properties

```
class Scientist: FullName {
        var firstName = ""
        var lastName = ""
}
```

Protocols - Properties

```
class Scientist: FullName {
        var firstName = ""
        var lastName = ""
}
```

Protocols - Methods

```
protocol FullName {
        var firstName: String {get set}
        var lastName: String {get set}


        func getFullName() -> String
}
```

Protocols - Methods

```
class Scientist: FullName {
        var firstName = ""
        var lastName = ""
        var field = “”

        func getFullName() -> String {
                return "\(firstName) \(lastName) studies \(field)"
        }
}
```

Protocols - Methods

```
struct FootballPlayer: FullName {

    var firstName = ""
    var  lastName  =  ""
    var number = 0

    func getFullName() -> String {
        return "\(firstName) \(lastName) has the number \(number)"
     }
}
```

Protocols - Methods

```
var scientist = Scientist()
scientist.firstName = "Kara”
scientist.lastName = "Hoffman”
scientist.field = "Physics"

var player = FootballPlayer();
player.firstName = "Dan”
player.lastName = "Marino”
player.number = 13

var person: FullName
person = scientist;
print(person.getFullName())
person = player
print(player.getFullName())
```

```
var scientist = Scientist()
scientist.firstName = "Kara”
scientist.lastName = "Hoffman”
scientist.field = "Physics"

var player = FootballPlayer();
player.firstName = "Dan”
player.lastName = "Marino”
player.number = 13

var person: FullName
person = scientist;
print(person.getFullName())
person = player
print(player.getFullName())
```

Kara Hoffman studies Physics
Dan Marino has the number 13

Extensions

```
extension String {
        //add new functionality here
}
```

Extensions

```
extension String {
    var firstLetter: Character? {
        get {
            return self.characters.first
        }
    }

    func reverse() -> String {
        var reverse = ""
        for letter in self.characters {
            reverse = "\(letter)" + reverse
        }
        return reverse
    }
}
```

Extensions

```
var myString = "Learning Swift is fun"

print(myString.reverse())

print(myString.firstLetter)
```

Memory management

```
class MyClass {
  var name = ""
  init(name: String) {
    self.name = name
    print("Initializing class with name \(self.name)")
  }
  deinit {
   print("Releasing class with name \(self.name)")
  }
}
```

Memory management

```
var class1ref1: MyClass? = MyClass(name: "One")
var class2ref1: MyClass? = MyClass(name: "Two")
var class2ref2: MyClass? = class2ref1

print("Setting class1ref1 to nil")
class1ref1 = nil

print("Setting class2ref1 to nil")
class2ref1 = nil

print("Setting class2ref2 to nil")
class2ref2 = nil
```

Memory management

```
var class1ref1: MyClass? = MyClass(name: "One")
var class2ref1: MyClass? = MyClass(name: "Two")
var class2ref2: MyClass? = class2ref1

print("Setting class1ref1 to nil")
class1ref1 = nil

print("Setting class2ref1 to nil")
class2ref1 = nil

print("Setting class2ref2 to nil")
class2ref2 = nil
```

```
Initializing class with name One
Initializing class with name Two
Setting class1ref1 to nil
Releasing class with name One
Setting class2ref1 to nil
Setting class2ref2 to nil
Releasing class with name Two
```

Memory management

```
class MyClass1 {
    var name = ""
    var class2: MyClass2?

  init(name: String) {
      self.name = name
      print("Initializing class with name \(self.name)")
  }
  deinit {
      print("Releaseing class with name \(self.name)")
  }

}

class MyClass2 {
    var name = ""
    var class1: MyClass1?

  init(name: String) {
      self.name = name
      print("Initializing class2 with name \(self.name)")
  }
  deinit {
      print("Releaseing class2 with name \(self.name)")
  }

}
```

Memory management

```
class MyClass1 {
    var name = ""
    var class2: MyClass2?

  init(name: String) {
      self.name = name
      print("Initializing class with name \(self.name)")
  }
  deinit {
      print("Releaseing class with name \(self.name)")
  }

}

class MyClass2 {
    var name = ""
    var class1: MyClass1?

  init(name: String) {
      self.name = name
      print("Initializing class2 with name \(self.name)")
  }
  deinit {
      print("Releaseing class2 with name \(self.name)")
  }

}
```

Memory management

```
var class1: MyClass1? = MyClass1(name: "Class1")
var class2: MyClass2? = MyClass2(name: "Class2")
//class1 and class2 each have a reference count of 1

class1?.class2 = class2
//Class2 now has a reference count of 2
class2?.class1 = class1
//class1 now has a reference count of 2

print("Setting classes to nil")
class2 = nil
//class2 now has a reference count of 1, not destroyed
class1 = nil
//class1 now has a reference count of 1, not destroyed
```

Memory management

```
class MyClass3 {
    var name = ""
    unowned let class4: MyClass4

    init(name: String, class4: MyClass4) {
        self.name = name
        self.class4 = class4
        print("Initializing class3 with name \(self.name)")
    }
    deinit {
        print("Releasing class3 with name \(self.name)")
    }

}

class MyClass4{
    var name = ""
    var class3: MyClass3?

    init(name: String) {
        self.name = name
        print("Initializing class4 with name \(self.name)")
    }
    deinit {
        print("Releasing class4 with name \(self.name)")
    }
}
```

Memory management

```
var class4 = MyClass4(name: "Class4")
var class3: MyClass3? = MyClass3(name: "class3", class4: class4)

class4.class3 = class3

print("Classes going out of scope")
```

Memory management

```
var class4 = MyClass4(name: "Class4")
var class3: MyClass3? = MyClass3(name: "class3", class4: class4)

class4.class3 = class3

print("Classes going out of scope")
```

```
Initializing class4 with name Class4
Initializing class3 with name class3
Classes going out of scope.
Releasing class4 with name Class4
Releasing class3 with name class3
```

Memory management

```
var class4 = MyClass4(name: "Class4")
var class3: MyClass3? = MyClass3(name: "class3", class4: class4)

class4.class3 = class3

print("Classes going out of scope")
```

```
Initializing class4 with name Class4
Initializing class3 with name class3
Classes going out of scope.
Releasing class4 with name Class4
Releasing class3 with name class3
```

Memory management

```
class MyClass5 {
  var name = ""
  var class6: MyClass6?
  init(name: String) {
    self.name = name
    print("Initializing class5 with name \(self.name)")
  }
  deinit {
    print("Releasing class5 with name \(self.name)")
  }
}

class MyClass6 {
  var name = ""
  weak var class5: MyClass5?
  init(name: String) {
    self.name = name
    print("Initializing class6 with name \(self.name)")
  }
  deinit {
    print("Releasing class6 with name \(self.name)")
  }
}
```

Memory management

```
var class5: MyClass5? = MyClass5(name: "Class5")
var class6: MyClass6? = MyClass6(name: "Class6")

class5?.class6 = class6
class6?.class5 = class5

print("Classes going out of scope ")
```

Memory management

```
var class5: MyClass5? = MyClass5(name: "Class5")
var class6: MyClass6? = MyClass6(name: "Class6")

class5?.class6 = class6
class6?.class5 = class5

print("Classes going out of scope "
```

```
Initializing class5 with name Class5
Initializing class6 with name Class6
Classes going out of scope.
Releasing class5 with name Class5
Releasing class6 with name Class6
```

Thursday

- Protocols and Extensions
- No need for Xcode as yet.

Protocol Oriented Programming

- How protocols are used as a type
- How to implement polymorphism in Swift using protocols
- How to use protocol extensions
- Why we would want to use protocol extensions

Protocol as Types

```
protocol PersonProtocol {
    var firstName: String {get set}
    var lastName: String {get set}
    var birthDate: NSDate {get set}
    var profession: String {get}

    init (firstName: String, lastName: String, birthDate: NSDate)
}
```

Protocol as Types

```
func updatePerson(person: PersonProtocol) -> PersonProtocol {
    // Code to update person goes here
    return person
}
```

```
var myPerson: PersonProtocol
```

```
var people: [PersonProtocol] = []
```

Protocol as Types

PersonProtocol protocol:var
myPerson: PersonProtocol

myPerson = SwiftProgrammer(firstName: "Jon", lastName: "Hoffman", birthDate: bDateProgrammer)
print("\(myPerson.firstName) \(myPerson.lastName)")

myPerson = FootballPlayer(firstName: "Dan", lastName: "Marino", birthDate: bDatePlayer)
print("\(myPerson.firstName) \(myPerson.lastName)")

Polymorphism with Protocols

```
for person in people {
    print("\(person.firstName) \(person.lastName):
        \(person.profession)")
}
```

```
Jon Hoffman: Swift Programmer
Dan Marino: Football Player
```

Type casting with Protocols

```
for person in people {
  if person is SwiftProgrammer {
     print("\(person.firstName) is a Swift Programmer")
}
}
```

Type casting with Protocols

```
for person in people {
    switch (person) {
    case is SwiftProgrammer:
        print("\(person.firstName) is a Swift Programmer")
    case is FootballPlayer:
        print("\(person.firstName) is a Football Player")
    default:
        print("\(person.firstName) is an unknown type")
    }
}
```

Type casting with Protocols

```
for person in people where person is SwiftProgrammer {
    print("\(person.firstName) is a Swift Programmer")

}
```

Type casting with Protocols

- as?
- as!

Type casting with Protocols

```
for person in people {
    if let p = person as? SwiftProgrammer {
        print("\(person.firstName) is a Swift Programmer")
    }
}
```

```
for person in people where person is SwiftProgrammer {
  let p = person as! SwiftProgrammer
}
```

Protocol Extensions

```
protocol DogProtocol {
    var name: String {get set}
    var color: String {get set}
}
```

Protocol Extensions

```
struct JackRussel: DogProtocol {
    var name: String
    var color: String
}

class WhiteLab: DogProtocol {
    var name: String
    var color: String

    init(name: String, color: String) {
        self.name = name
        self.color = color
    }
}

struct Mutt: DogProtocol {
    var name: String
    var color: String
}
```

Protocol Extensions

```
protocol DogProtocol {
    var name: String {get set}
    var color: String {get set}
    func speak() -> String
}
```

Protocol Extensions

```
struct JackRussel: DogProtocol {
    var name: String
    var color: String
     func speak() -> String {
        return "Woof Woof"
    }
}

class WhiteLab: DogProtocol {
    var name: String
    var color: String

    init(name: String, color: String) {
        self.name = name
        self.color = color
    }
    func speak() -> String {
        return "Woof Woof"
    }

}

struct Mutt: DogProtocol {
    var name: String
    var color: String
    func speak() -> String {
        return "Woof Woof"
    }

}
```

Protocol Extensions

```
protocol DogProtocol {
    var name: String {get set}
    var color: String {get set}
}

extension DogProtocol {
    func speak() -> String {
        return "Woof Woof"
    }
}
```

Protocol Extensions

```
struct JackRussel: DogProtocol {
    var name: String
    var color: String
}
class WhiteLab: DogProtocol {
    var name: String
    var color: String

    init(name: String, color: String) {
        self.name = name
        self.color = color
    }
}

struct Mutt: DogProtocol {
    var name: String
    var color: String
}
```

Protocol Extensions

```
let dash = JackRussel(name: "Dash", color: "Brown and White")
let lily = WhiteLab(name: "Lily", color: "White")
let buddy = Mutt(name: "Buddy", color: "Brown")
let dSpeak = dash.speak()  // returns "woof woof"
let lSpeak = lily.speak()  // returns "woof woof"
let bSpeak = buddy.speak() // returns "woof woof"
```

Protocol Extensions

```
struct Mutt: DogProtocol {
    var name: String
    var color: String
    func speak() -> String {
        return "I am hungry"
    }
}
```

Protocol Extensions

```
protocol TextValidating {
    var regExMatchingString: String {get}
    var regExFindMatchString: String {get}
    var validationMessage: String {get}

    func validateString(str: String) -> Bool
    func getMatchingString(str: String) -> String?
}
```

Protocol Extensions

```
class AlphaValidation1: TextValidating {
    static let sharedInstance = AlphaValidation1()
    private init(){}

    let regExFindMatchString = "^[a-zA-Z]{0,10}"
    let validationMessage = "Can only contain Alpha characters"

    var regExMatchingString: String { get {
        return regExFindMatchString + "$"
        }
    }

    func validateString(str: String) -> Bool {
        if let _ = str.range(of:regExMatchingString, options:
            .regularExpression) {
            return true
        } else {
            return false
        }
    }
    func getMatchingString(str: String) -> String? {
        if let newMatch = str.range(of:regExFindMatchString, options:
            .regularExpression) {
            return str.substring(with:newMatch)
        } else {
            return nil
        }
    }
}
```

Protocol Extensions

```
class AlphaValidation1: TextValidating {
    static let sharedInstance = AlphaValidation1()
    private init(){}

    let regExFindMatchString = "^[a-zA-Z]{0,10}"
    let validationMessage = "Can only contain Alpha characters"

    var regExMatchingString: String { get {
        return regExFindMatchString + "$"
        }
    }

    func validateString(str: String) -> Bool {
        if let _ = str.range(of:regExMatchingString, options:
            .regularExpression) {
            return true
        } else {
            return false
        }
    }
    func getMatchingString(str: String) -> String? {
        if let newMatch = str.range(of:regExFindMatchString, options:
            .regularExpression) {
            return str.substring(with:newMatch)
        } else {
            return nil
        }
    }
}
```

Protocol Extensions

```
protocol TextValidating {
    var regExFindMatchString: String {get}
    var validationMessage: String {get}
}
```

Protocol Extensions

```
extension TextValidating {

    var regExMatchingString: String { get {
        return regExFindMatchString + "$"
        }
    }

    func validateString(str: String) -> Bool {
        if let _ = str.range(of:regExMatchingString, options:
            .regularExpression) {
            return true
        } else {
            return false
        }
    }
    func getMatchingString(str: String) -> String? {
        if let newMatch = str.range(of:regExFindMatchString, options:
            .regularExpression) {
            return str.substring(with: newMatch)
        } else {
            return nil
        }
    }
}
```

Protocol Extensions

```
struct AlphaValidation: TextValidating {
    static let sharedInstance = AlphaValidation()
    private init(){}

    let regExFindMatchString = "^[a-zA-Z]{0,10}"
    let validationMessage = "Can only contain Alpha characters"
}

struct AlphaNumericValidation: TextValidating {
    static let sharedInstance = AlphaNumericValidation()
    private init(){}

    let regExFindMatchString = "^[a-zA-Z0-9]{0,15}"
    let validationMessage = "Can only contain Alpha Numeric characters"
}


struct DisplayNameValidation: TextValidating {
    static let sharedInstance = DisplayNameValidation()
    private init(){}

    let regExFindMatchString = "^[\\s?[a-zA-Z0-9\\-_\\s]]{0,15}"
    let validationMessage = "Display Name can contain only contain
        Alphanumeric Characters"
}
```

Protocol Extensions

```
var myString1 = "abcxyz"
var myString2 = "abc123"

let validation = AlphaValidation.sharedInstance

let valid1 = validation.validateString(str: myString1)
let newString1 = validation.getMatchingString(str: myString1)

let valid2 = validation.validateString(str: myString2)
let newString2 = validation.getMatchingString(str: myString2)
```

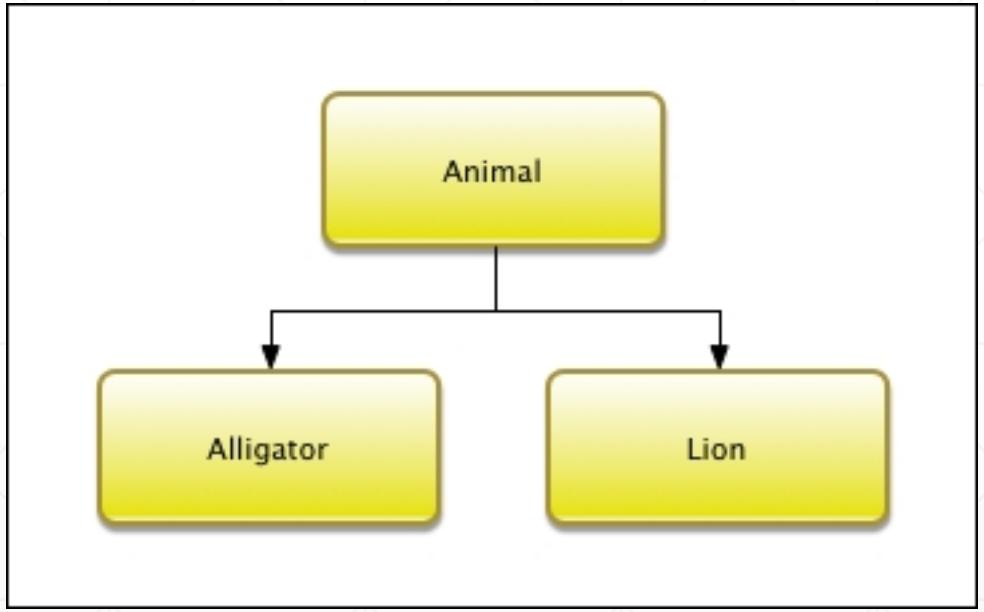
Protocol Oriented Design

- The difference between OOP and POP design
- What is protocol-oriented design?
- What is protocol composition?
- What is protocol inheritance?

Protocol Oriented Design - Requirements

- We will have three categories of animal: sea, land, and air.
- Animals may be members of multiple categories. For example an alligator can be a member of both the land and sea categories.
- Animals may be able to attack and/or move when they are on a tile that matches the categories they are in.
- Animals will start off with a certain number of hit points and if those hit points reach 0 or less then they will die.
- For our example here we will define two animals (Lion and Alligator) but we know that the number of animal types will grow as we develop the game.

Protocol Oriented Design - OOD



Protocol Oriented Design - OOD

```
class Animal
{
    private var landAnimal = false
    private var landAttack = false
    private var landMovement = false

    private var seaAnimal = false
    private var seaAttack = false
    private var seaMovement = false

    private var airAnimal = false
    private var airAttack = false
    private var airMovement = false

    private var hitPoints = 0
}
```

Protocol Oriented Design - OOD

```
init()
{
    landAnimal = false
    landAttack = false
    landMovement = false
    airAnimal = false
    airAttack = false
    airMovement = false
    seaAnimal = false
    seaAttack = false
    seaMovement = false
    hitPoints = 0
}
```

Protocol Oriented Design - OOD

```
func isLandAnimal() -> Bool { return landAnimal }
func canLandAttack() -> Bool { return landAttack }
func canLandMove() -> Bool { return landMovement }
func isSeaAnimal() -> Bool { return seaAnimal }
func canSeaAttack() -> Bool { return seaAttack }
func canSeaMove() -> Bool { return seaMovement }
func isAirAnimal() -> Bool { return airAnimal }
func canAirAttack() -> Bool { return airAttack }
func canAirMove() -> Bool { return airMovement }

func doLandAttack() {}
func doLandMovement() {}
func doSeaAttack() {}
func doSeaMovement() {}
func doAirAttack() {}
func doAirMovement() {}

func takeHit(amount: Int) { hitPoints -= amount }
func hitPointsRemaining() -> Int { return hitPoints }
func isAlive() -> Bool { return hitPoints > 0 ? true : false }
}
```

```
class Lion: Animal {

  override init() {
    super.init()
    landAnimal = true
    landAttack = true
    landMovement = true
    hitPoints = 20
  }

  override func doLandAttack() { print("Lion Attack") }
  override func doLandMovement() { print("Lion Move") }
}

class Alligator: Animal {

  override init() {
    super.init()
    landAnimal = true
    landAttack = true
    landMovement = true

    seaAnimal = true
    seaAttack = true
    seaMovement = true
    hitPoints = 35
  }

  override func doLandAttack() { print("Alligator Land Attack") }
  override func doLandMovement() { print("Alligator Land Move") }
  override func doSeaAttack() { print("Alligator Sea Attack") }
  override func doSeaMovement() { print("Alligator Sea Move") }

}
```

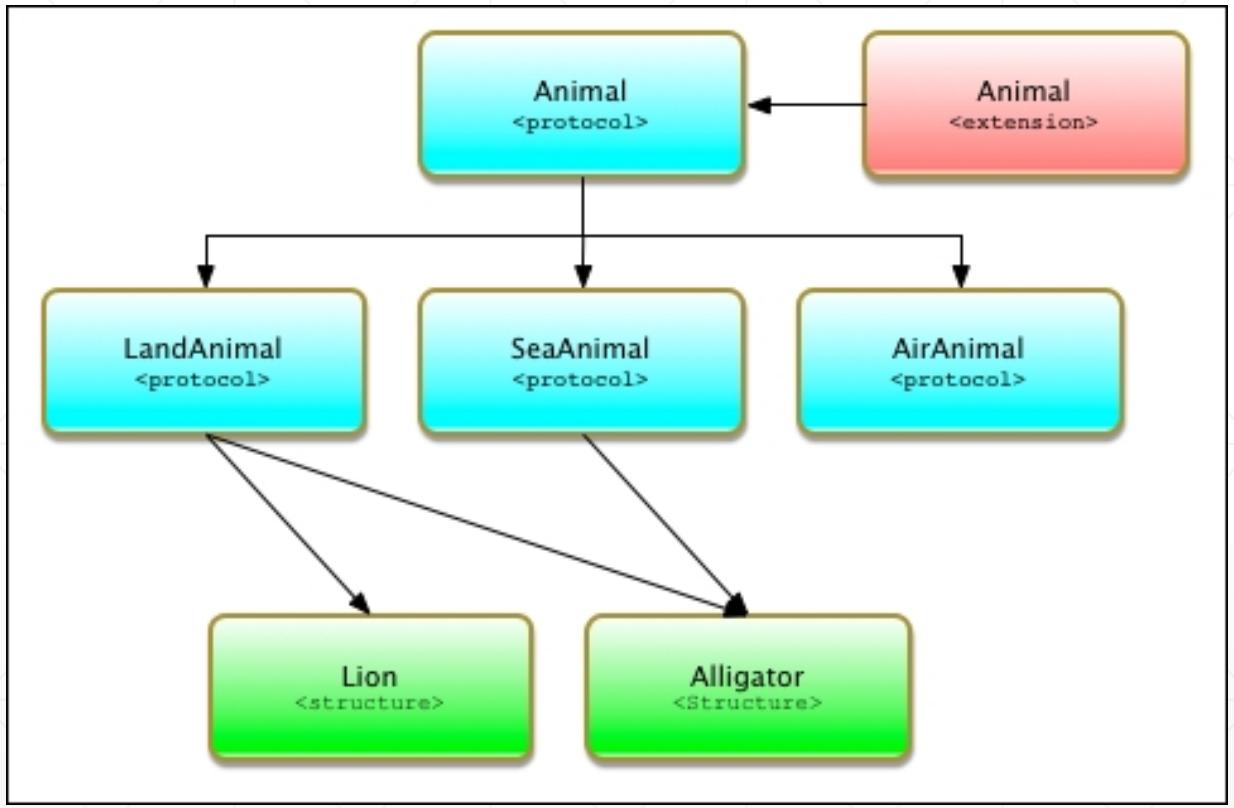
```
var animals = [Animal]()

var an1 = Alligator()
var an2 = Alligator()
var an3 = Lion()

animals.append(an1)
animals.append(an2)
animals.append(an3)

for (index, animal) in animals.enumerated() {
  if animal.isAirAnimal() {
  print("Animal at \(index) is Air")
}
if animal.isLandAnimal() {
  print("Animal at \(index) is Land")
}
if animal.isSeaAnimal() {
  print("Animal at \(index) is Sea")
}
}
```

Protocol Oriented Design - POP



Protocol Oriented Design - POP

```
protocol Animal {
    var hitPoints: Int {get set}
}
```

Protocol Oriented Design - POP

```
extension Animal {
    mutating func takeHit(amount: Int) { hitPoints -= amount }
    func hitPointsRemaining() -> Int { return hitPoints }
    func isAlive() -> Bool { return hitPoints > 0 ? true : false }
}
```

Protocol Oriented Design - POP

```
protocol LandAnimal: Animal {
    var landAttack: Bool {get}
    var landMovement: Bool {get}

    func doLandAttack()
    func doLandMovement()
}

protocol SeaAnimal: Animal {
    var seaAttack: Bool {get}
    var seaMovement: Bool {get}

    func doSeaAttack()
    func doSeaMovement()
}

protocol AirAnimal: Animal {
    var airAttack: Bool {get}
    var airMovement: Bool {get}

    func doAirAttack()
    func doAirMovement()
}
```

```
struct Lion: LandAnimal {
    var hitPoints = 20
    let landAttack = true
    let landMovement = true

    func doLandAttack() { print("Lion Attack") }
    func doLandMovement() { print("Lion Move") }
}

struct Alligator: LandAnimal, SeaAnimal {
    var hitPoints = 35
    let landAttack = true
    let landMovement = true
    let seaAttack = true
    let seaMovement = true

    func doLandAttack() { print("Alligator Land Attack") }
    func doLandMovement() { print("Alligator Land Move") }
    func doSeaAttack() { print("Alligator Sea Attack") }
    func doSeaMovement() { print("Alligator Sea Move") }
}
```

```
var animals = [Animal]()

var an1 = Alligator()
var an2 = Alligator()
var an3 = Lion()

animals.append(an1)
animals.append(an2)
animals.append(an3)


for (index, animal) in animals.enumerated() {
    if let animal = animal as? AirAnimal {
       print("Animal at \(index) is Air")
    }
    if let animal = animal as? LandAnimal {
       print("Animal at \(index) is Land")
    }
    if let animal = animal as? SeaAnimal {
       print("Animal at \(index) is Sea")
   }
}
```

Protocol Oriented Design - POP

```
for (index, animal) in animals.enumerated() where animal is SeaAnimal {
    print("Only Sea Animal: \(index)")
}
```