Protocols

Closures

Road Map

- Enums demo
- Protocols
- *Protocols demo*

Protocols

List of methods and properties that

*should be implemented by adopter**

*and something more 😎

Protocols

```
//  example protocol with single property
protocol NamedCreature {
    //  can be { get } or { get set }
    var name: String { get }
     // can be a function like func sayOurName()
}

class Animal: NamedCreature {
    let nickName: String = ""
    var name: String {
        return nickName
    }
}

struct Person {
    let firstName: String
    let secondName: String
}

extension Person: NamedCreature {
    var name: String {
        return firstName + " " + secondName
    }
}
```

Protocol polymorphism

- Different types can implement same methods/properties
- Standard library relies on it a lot
- Classes, Structures, Enums can adopt Protocol
- Protocol inheritance

StdLib example

- *CustomStringConvertible* for printing
- Each adopter implements

```
var description: String
```

- This string is printed out
- Want custom string to be printed out - adopt
- Print of non-adopters

Protocol extension

- You can add methods/computed properties*

```
extension NamedCreature {
    func call() {
        print("Hey, \(fullName)!")
    }
}

let person = Person(firstName: "Tim", secondName: "Cook")
person.call() /* Prints `Hey, Tim Cook!` */

let cat = Pet(nickName: "Barsik")
cat.call() /* Prints `Hey, Barsik!` */
```

* **If you rewrite same methods in adopter, the one in the protocol extension version will be used**

Protocol extension

- You can adopt Protocols for existing types in extensions

```
extension Int: NamedCreature {
    var fullName: String {
        if abs(self) < 128 {
            return "Small Int"
        } else {
            return "Big Int"
        }
    }
}

100.call() /* Prints `Hey, Small Int!` */
```

Protocol checks

Check if instance adopts protocol

```
class Monkey {
}

let monkey = Monkey()
let monkeyName = (monkey as? NamedCreature)?.name
```

Useful Protocols

- CustomStringConvertible

```
var description: String { get }
```

- Equitable

```
func ==(lhs: Self, rhs: Self) -> Bool
```

- Comparable
- Hashable

```
var hashValue: Int { get }
```

http://nshipster.com/swift-comparison-protocols/

Protocol associatedtype

Adopter can substitute any specific type instead of associatedtype

```
protocol Animal {
    associatedtype FoodType

    mutating func eat(food: FoodType)
}

struct Grass {
}

struct Cow: Animal {
    func eat(food: Grass) {

    }
}

struct Tiger: Animal {
    func eat(food: Cow) {

    }
}
```

Swift

is a first Protocol oriented programming language