CS4962 - iOS
Extensions, Protocols, and Generics
Package Management
Application level package managers

- CocoaPods
- Carthage
- SPM (Swift Package Manager)
CocoaPods

- Centralized, searchable repository
- Automatically integrates with your project
- Simple to get started
Carthage

- Decentralized system
- Manual Integration required
- Creates Frameworks (requires iOS8+)
Swift Package Manager (SPM)

- Decentralized system
- Developed by Apple
- Still in development. Not yet available (Swift 3)
CocoaPods Example
Extensions

• Add new functionality to types
• Do not allow stored properties
• Enums, Classes, Structs, Enums, Types, Protocols can be extended
• Localize and namespace common operations
Extensions

- Extensions can implement properties

```swift
extension UIColor {
    var redValue: CGFloat {
        var r: CGFloat = 0
        self.getRed(&r, green: nil, blue: nil, alpha: nil)

        return r
    }
}
```
Extensions

- Extensions can implement methods

```swift
extension String {
    func debugLog() {
        print("[DEBUG]: " + self)
        RemoteLogger.debugLog(self)
    }
}
```
Extensions

- Extensions can be made for types

```swift
extension Int {
    func power(power: Int) -> Int {
        let value = pow(Double(self), Double(power))
        return Int(value)
    }
}
```
Extensions

- Extensions can be made for protocols

```swift
extension CollectionType {
    var evenElementCount: Bool {
        return count % 2 == 0
    }
}
```
Extensions

• Extensions can be made for protocols with constraints

```swift
extension SequenceType where Generator.Element: Comparable {

    var ascending: Bool {
        // Determine if sequence is in ascending order
        return false
    }
}
```
Extensions

• Extensions can conform to protocols

```swift
extension GamesViewController: UICollectionViewDataSource {
    func collectionView(collectionView: UICollectionView,
                         numberOfItemsInSection section: Int) -> Int {
        return 10
    }
}
```
Protocols

• Express a contract of methods and properties that implementers will implement
• Similar to Interfaces in Java
• Swift doesn't have abstract classes
• Protocols can mimic behavior of OOP and multiple inheritance
Object-Oriented Programming

class Plane {
    var passedSafetyCheck: Bool
    var passengerCount: Int
    var fuelLevel: Double
    var maxFuelLevel: Double

    var fuelPercentage: Double {
        return fuelLevel / maxFuelLevel
    }
}
Object-Oriented Programming

class Vehicle {
    var passedSafetyCheck: Bool
    var fuelLevel: Double
    var maxFuelLevel: Double

    var fuelPercentage: Double { return fuelLevel / maxFuelLevel }
}
Object-Oriented Programming

class RefuelingTanker: Vehicle {
    ...
}

class Plane: Vehicle {
    var passengerCount: Int
}

class Shuttle: Vehicle {
    var passengerCount: Int
}
Object-Oriented Programming

- Base classes can grow to allow abstraction

```swift
class Vehicle {
    var passedSafetyCheck: Bool

    // Gas
    var fuelLevel: Double
    var maxFuelLevel: Double

    // Electric
    var chargeLevel: Double
    var maxChargeLevel: Double
}
```
Object-Oriented Programming

• Or deep levels of abstraction

class Vehicle { var passedSafetyCheck: Bool }

class GasVehicle: Vehicle {
    var fuelLevel: Double
    var maxFuelLevel: Double
}

class ElectricVehicle: Vehicle {
    var chargeLevel: Double
    var maxChargeLevel: Double
}
Protocols

protocol Fuelable {
    var fuelLevel: Double { get set }
    var maxFuelLevel: Double { get set }
}

protocol Electric {
    var chargeLevel: Double { get set }
    var maxChargeLevel: Double { get set }
}
Protocols

protocol Vehicle {
  var passedSafetyCheck: Bool { get set }
}

protocol Shuttleable: Vehicle {
  var passengerCount: Int { get set }
  var maxPassengerCount: Int { get set }
}
Protocols

```swift
struct Plane: Shuttleable, Fuelable {
    var passedSafetyCheck: Bool
    var fuelLevel: Double
    var maxFuelLevel: Double

    ...
}

struct Shuttle: Shuttleable, Electric {
}
```
Protocols

- Allows multiple protocol conformance

```swift
struct HybridShuttle: Shuttleable, Electric, Fuelable {
    var passedSafetyCheck: Bool
    var fuelLevel: Double
    var maxFuelLevel: Double
    var chargeLevel: Double
    var maxChargeLevel: Double
}
```
Protocols

extension Fuelable {
    var fuelPercentage: Double {
        return fuelLevel / maxFuelLevel
    }
}

extension Electric {
    var chargePercentage: Double {
        return chargeLevel / maxChargeLevel
    }
}
Protocols

• Protocols can be treated as types

```swift
// Can contain Planes and Shuttles
var vehicles: [Vehicle]

// Can contain items that conform to Electric and Vehicle
var electricVehicles: [protocol<Electric, Vehicle>]
```
Protocol-Oriented Programming

- WWDC 2015: Protocol-Oriented Programming in Swift
- Crustacean Sample Code: A demonstration of protocol-oriented programming in Swift.
Generics

• Allow implementation of data structures and methods to handle any generic type

```swift
struct Array<Element> {
    ...
}

var numbers: [Int]
var numbers: Array<Int>
```
Generics

- Methods can be handle generics as well

```swift
func swappedTuple<T, U>(first: T, second: U) -> (U, T) {
    return (second, first)
}
```
Generics

- Constraints can be applied to generics as well

```swift
func generateHash<T: Hashable>(values: [T]) -> Int {
    // poor hash algorithm
    var hash: Int = 0
    for value in values {
        hash ^= value.hashValue
    }
    return hash
}
```
Example