You Belong With Me:
An Extension Love Story

Neem Serra
@TeamNeem

ME!

```
class Neem: Person {
    let job = "Software Developer"
    let company = "Slalom"
    let location = "St. Louis"

    let favoriteLanguage = "Swift"
    let favoriteFood = "Cupcakes"

    let twitter = "@TeamNeem"
    let github = "TeamNeem"
    let email = "neem.serra@gmail.com"
}
```

Logan

Co-presenter

Logan

```
class Logan: Baby {
    let favoriteAnimal = "Dragon"

    let birthHeight = 0.53
    var currentHeight = 0.62

    let initialWeight =
        Weight(pounds: 6, ounces: 2)

    var ageInMonths: Int = 4
    var words = ["coo", "agoo"]
}
```

Extensions add new functionality to an existing class, structure, enumeration, or protocol type.

You can add new things to existing stuff

How can I make a baby just go to sleep???

The Baby Class

```
class Baby: Person {
    func drinkMilk() {}

    func cry() {}

    func pee(diaper: Diaper) {}

    func poop(diaper: Diaper) {}

    func lookCute() {}
}
```

Extension

```
extension Baby {
    func sleep() {}
}
```

Now I can call baby.sleep()

Old Way 1: Create a Custom Subclass

```
class CustomBaby: Baby {
    func sleep() {}
}

class Logan: CustomBaby {}

class HypotheticalSecondBaby: CustomBaby {}
```

Old Way 2: Have duplicated code

```
class Logan: Baby {
    func sleep() {
        //same code
    }
}

class HypotheticalSecondBaby: Baby {
    func sleep() {
        //same code
    }
}
```

Limitations of extensions

```
class HypotheticalSecondChild: Baby {
    override func sleep() {

    }
}
```

Overriding non-@objc declarations from extensions is not supported

Can’t override functionality from an extension in a subclass

The Mommy Protocol

```
protocol MommyProtocol {
    func soothe(baby: Baby)
    func changeDiaper(diaper: Diaper) -> Diaper
    func feedBaby() -> String
    func multitask()
}
```

The Mommy Protocol

```
protocol MommyProtocol {
    func soothe(baby: Baby)
    func changeDiaper(diaper: Diaper) -> Diaper
    func feedBaby() -> String
    func multitask()
}
```

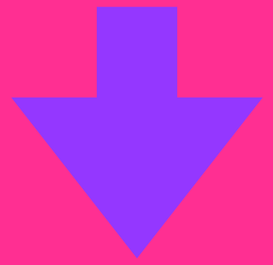
```
class Neem: Person, MommyProtocol {
    func soothe(baby: Baby) {}
    func changeDiaper(diaper: Diaper) -> Diaper {}
    func feedBaby() -> String {}
    func multitask() {}
}
```

Extend Mommy Protocol

```
extension MommyProtocol {
    func multitask() {
        //do all the things
    }

    func feedBaby() -> String {
        return "Breastfeeding"
    }
}
```

```
class Neem: Person, MommyProtocol {
    func soothe(baby: Baby) {}
    func changeDiaper(diaper: Diaper) -> Diaper {}
    func feedBaby() -> String {}
    func multitask() {}
}
```



```
class Neem: Person, MommyProtocol {
    func soothe(baby: Baby) {}
    func changeDiaper(diaper: Diaper) -> Diaper {}
}
```

Cleaner code

Neem class is a hot mess

Neem class is a hot mess

```
class Neem: Person, MommyProtocol, DeveloperProtocol,
    AdultingProtocol, SelfCareProtocol {
    func soothe(baby: Baby) { baby.sleep() }
    func changeDiaper(diaper: Diaper) -> Diaper {
        return Diaper() }
    func writeCode() {}
    func eatCupcakes() {}
    func paintNails() {}
    func experienceAnxiety() {}
    func payBills() {}
    func attendMeetings() {}
    func readComics() {}
    func hostParties() {}
    func testCode() {}
    func foldLaundry() {}
    func sleepTooLittle() {}
    func refactor() {}
}
```

Organize using extensions!

```swift
//MARK: - Mommy Protocol Methods
extension Neem: MommyProtocol {
    func soothe(baby: Baby) {
        baby.sleep()
    }

    func changeDiaper(diaper: Diaper)
        -> Diaper {
        return Diaper()
    }
}
```

```swift
//MARK: - Adulting Protocol Methods
extension Neem: AdultingProtocol {
    func foldLaundry() {}

    func payBills() {}

    func experienceAnxiety() {}

    func sleepTooLittle() {}
}
```

```swift
//MARK: - Developer Protocol Methods
extension Neem: DeveloperProtocol {
    func writeCode() {}

    func attendMeetings() {}

    func testCode() {}

    func refactor() {}
}
```

```swift
//MARK: - Self Care Protocol Methods
extension Neem: SelfCareProtocol {
    func readComics() {}

    func eatCupcakes() {}

    func hostParties() {}

    func paintNails() {}
}
```

Product Debug Source Control Window Help

100%

An Extensions Love Story 〉 iPhone XR

You Belong With Me: An Extensions Love Story: **Ready** | Toda

You Belong With Me: An Extensions Love Story 〉 You Belong With Me: An Extensions Love Story 〉 Neem.s

```
1  class Neem: Person {
2      let job = "Software Developer
3      let company = "Slalom"
4      let location = "St. Louis"
5
6      let favoriteLanguage = "Swift
7      let favoriteFood = "Cupcakes"
8
9      let twitter = "@TeamNeem"
10     let github = "TeamNeem"
11     let email = "neem.serra@gmail
12 }
13
14 //MARK: - Mommy Protocol Methods
15 extension Neem: MommyProtocol {
16     func soothe(baby: Baby) {
```

- Neem
  - job
  - company
  - location
  - favoriteLanguage
  - favoriteFood
  - twitter
  - github
  - email
- **Mommy Protocol Methods**
  - Neem
    - soothe(baby:)
    - changeDiaper(diaper:)
- **Adulting Protocol Methods**
  - Neem
    - foldLaundry()
    - payBills()
    - experienceAnxiety()
    - sleepTooLittle()
- **Self Care Protocol Methods**
  - Neem
    - readComics()
    - eatCupcakes()
    - hostParties()
    - paintNails()
- **Developer Protocol Methods**
  - Neem
    - writeCode()
    - attendMeetings()
    - testCode()
    - refactor()

Product Debug Source Control Window Help

100%

An Extensions Love Story 〉 iPhone XR

You Belong With Me: An Extensions Love Story: Read

Toda

You Belong With Me: An Extensions Love Story 〉 You Belong With Me: An Extensions Love Story 〉 eem.s

```
1  class Neem: Person {
2      let job = "Software Developer"
3      let company = "Slalom"
4      let location = "St. Louis"
5
6      let favoriteLanguage = "Swift"
7      let favoriteFood = "Cupcakes"
8
9      let twitter = "@TeamNeem"
10     let github = "TeamNeem"
11     let email = "neem.serra@gmail"
12 }
13
14 //MARK: - Mommy Protocol Methods
15 extension Neem: MommyProtocol {
16     func soothe(baby: Baby) {
```

- Neem
  - job
  - company
  - location
  - favoriteLanguage
  - favoriteFood
  - twitter
  - github
  - email
- **Mommy Protocol Methods**
  - Neem
    - soothe(baby:)
    - changeDiaper(diaper:)
- **Adulting Protocol Methods**
  - Neem
    - foldLaundry()
    - payBills()
    - experienceAnxiety()
    - sleepTooLittle()
- **Self Care Protocol Methods**
  - Neem
    - readComics()
    - eatCupcakes()
    - hostParties()
    - paintNails()
- **Developer Protocol Methods**
  - Neem
    - writeCode()
    - attendMeetings()
    - testCode()
    - refactor()

Product Debug Source Control Window Help

100%

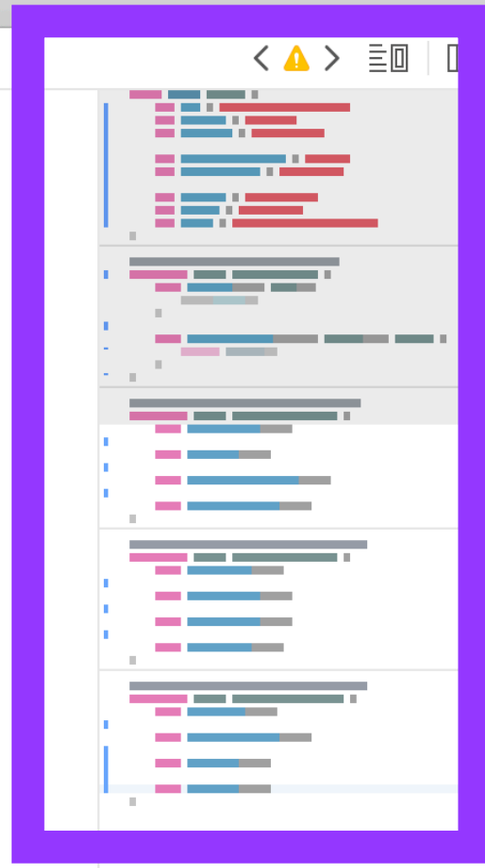
An Extensions Love Story 〉 iPhone XR

You Belong With Me: An Extensions Love Story: **Ready** | Toda

You Belong With Me: An Extensions Love Story 〉 You Belong With Me: An Extensions Love Story 〉 Neem.s

```
1  class Neem: Person {
2      let job = "Software Developer
3      let company = "Slalom"
4      let location = "St. Louis"
5
6      let favoriteLanguage = "Swift
7      let favoriteFood = "Cupcakes"
8
9      let twitter = "@TeamNeem"
10     let github = "TeamNeem"
11     let email = "neem.serra@gmail
12 }
13
14 //MARK: - Mommy Protocol Methods
15 extension Neem: MommyProtocol {
16     func soothe(baby: Baby) {
```

- Neem
  - job
  - company
  - location
  - favoriteLanguage
  - favoriteFood
  - twitter
  - github
  - email
- **Mommy Protocol Methods**
  - Neem
    - soothe(baby:)
    - changeDiaper(diaper:)
- **Adulting Protocol Methods**
  - Neem
    - foldLaundry()
    - payBills()
    - experienceAnxiety()
    - sleepTooLittle()
- **Self Care Protocol Methods**
  - Neem
    - readComics()
    - eatCupcakes()
    - hostParties()
    - paintNails()
- **Developer Protocol Methods**
  - Neem
    - writeCode()
    - attendMeetings()
    - testCode()
    - refactor()



Self-documenting code

Diaper time!

```
class Diaper {
    var contents: [String]

    init(contents: [String] = []) {
        self.contents = contents
    }
}
```

Only append if unique

```
extension Array where Element: Equatable {
    @discardableResult
    mutating func appendDistinct(_ newElement:
        Element) -> Bool {
        guard !contains(newElement) else { return
            false }
        append(newElement)
        return true
    }
}
```

Only append if unique

```
extension Array where Element: Equatable {
    @discardableResult
    mutating func appendDistinct(_ newElement:
        Element) -> Bool {
        guard !contains(newElement) else { return
            false }
        append(newElement)
        return true
    }
}
```

```
func poop(diaper: Diaper) {
    diaper.contents.appendDistinct("poop")
}
```

Constrained Extension

```
extension Array where Element: Equatable {
    @discardableResult
    mutating func appendDistinct(_ newElement:
        Element) -> Bool {
        guard !contains(newElement) else { return
            false }
        append(newElement)
        return true
    }
}
```

Only add extension when it matches a particular type

Custom Initializers

```
class Diaper {
    var contents: [String]

    init(contents: [String] = []) {
        self.contents = contents
    }
}

extension Diaper {
    convenience init(pee: Bool, poop: Bool) {
        var diaperContents = [String]()
        if pee {
            diaperContents.append("pee")
        }

        if poop {
            diaperContents.append("poop")
        }

        self.init(contents: diaperContents)
    }
}
```

Custom init for struct

```
struct Weight {
    let totalOunces: Double
}

let weight = Weight(totalOunces: 98)
```

Custom init for struct

```
struct Weight {
    let totalOunces: Double
}

let weight = Weight(totalOunces: 98)

extension Weight {
    init(pounds: Double, ounces: Double) {
        self.totalOunces = pounds * 16 + ounces
    }
}

let initialWeight = Weight(pounds: 6, ounces: 2)
```

Extension Limitations

- Can’t add new properties with an extension
- It would change the structure and memory of the class
- Ex. Can’t add a “pounds” property to the weight struct

Nested Types

Nested Types

```
extension Diaper {
    enum Kind {
        case clean, poop, pee
    }

    var kind: Kind {
        switch self.contents {
        case let contents where
            contents.contains("poop"):
            return .poop
        case let contents where
            contents.contains("pee"):
            return .pee
        default:
            return .clean
        }
    }
}
```

Clean Extra if Poopy

```
extension Neem: MommyProtocol {
    func changeDiaper(diaper: Diaper) -> Diaper {
        if diaper.kind == .clean {
            return diaper
        } else if diaper.kind == .poop {
            cleanExtra()
        }

        return Diaper()
}

func cleanExtra() {}
```

Computed Properties

Default height is in meters

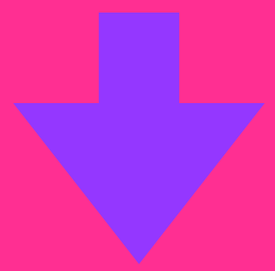
```
let birthHeight = 0.53
var currentHeight = 0.62
```

Extend double to convert

```
extension Double {
    var km: Double { return self * 1_000.0 }
    var m: Double { return self }
    var cm: Double { return self / 100.0 }
    var mm: Double { return self / 1_000.0 }
    var ft: Double { return self / 3.28084 }
    var inch: Double { return self / 39.37008 }
}
```

Call It What You Want

```
let birthHeight = 0.53
var currentHeight = 0.62
```



```
let birthHeight = 20.87.inch
var currentHeight = 24.5.inch
```

Mutating instance methods

Superman

```
extension Double {
    mutating func growAnInch() {
        self = self + 1.inch
    }
}
```

```
func grow() {
    currentHeight.growAnInch()
}

//currentHeight would change from
    24.5.inch to 25.5.inch
```

End Game

- Add functionality to classes, structs, etc.
- Add default implementation in protocols
- Organize code
- Self-document code
- Add custom initializers
- Add nested types
- Add computed properties
- Add mutating instance methods

Extensions: Forever and Always

- Questions?
- @TeamNeem
- github.com/TeamNeem
- Best repo for Swift extensions: https://github.com/SwifterSwift/SwifterSwift