BHINNEKA

BUSINESS SUPER-ECOSYSTEM

Dependency Injection with Swinject

Fadhil Hanri Kamarz
Mobile Development Specialist
Apps Department

BHINNEKA

BUSINESS SUPER-ECOSYSTEM

Love tech. Apple Developer Academy alumni. AnnoyanceBox and MOWL's creator. UI/UX design enthusiast. iOS Developer. Bhinneka Family, Mobile Apps Department.

fadhil.kamarz@bhinneka.com
linkedin.com/in/fadhilhanri
@fadhilhaka

Dependency Injection

Dependency injection means giving an object its instance variables. (James Shore, 2006)

What is Dependency injection?

How to implement Dependency injection?

```swift
class Improvement {
    let companyCourses: [Course] = [Course(name: .androidDevelopment, skill: .kotlin),
                                    Course(name: .iOSDevelopment, skill: .swift),
                                    Course(name: .webDevelopment, skill: .javascript)]

    func getCourses() -> [Course] {
        companyCourses
    }
}

class Engineer {
    var skills: Set<String> = []

    func learnCourse(from name: CourseName) {
        let selfGrowth = Improvement()
        let course = selfGrowth.getCourses().filter { $0.name == name }.first!
        skills.insert(course.skill.rawValue)
    }
}

let fadhil = Engineer()
    fadhil.learnCourse(from: .androidDevelopment)
```

Property Injection

```swift
protocol Development {
    func getCourses() -> [Course]
}

class Improvement: Development {
    let companyCourses: [Course] = [Course(name: .androidDevelopment, skill: .kotlin),
                                    Course(name: .iOSDevelopment, skill: .swift),
                                    Course(name: .webDevelopment, skill: .javascript)]

    func getCourses() -> [Course] {
        companyCourses
    }
}

class Engineer {
    var selfGrowth: Development?
    var skills: Set<String> = []

    func learnCourse(from name: CourseName) {
        let course = selfGrowth?.getCourses().filter { $0.name == name }.first!
        skills.insert((course?.skill.rawValue)!)
    }
}

class Injector {
    static func provideImprovement() -> Development {
        return Improvement()
    }
}

let fadhil = Engineer()
    fadhil.selfGrowth = Injector.provideImprovement()
    fadhil.learnCourse(from: .webDevelopment)
```

Method Injection

```swift
protocol Development {
    func getCourses() -> [Course]
}

class Improvement: Development {
    let companyCourses: [Course] = [Course(name: .androidDevelopment, skill: .kotlin),
                                    Course(name: .iOSDevelopment, skill: .swift),
                                    Course(name: .webDevelopment, skill: .javascript)]

    func getCourses() -> [Course] {
        companyCourses
    }
}

class Engineer {
    var skills: Set<String> = []

    func learnCourse(from name: CourseName, using improvement: Development) {
        let course = improvement.getCourses().filter { $0.name == name }.first!
        skills.insert((course.skill.rawValue))
    }
}

class Injector {
    static func provideImprovement() -> Development {
        return Improvement()
    }
}

let fadhil = Engineer()
    fadhil.learnCourse(from: .webDevelopment, using: Injector.provideImprovement())
```

Initializer Injection

```swift
protocol Development {
    func getCourses() -> [Course]
}

class Improvement: Development {
    let companyCourses: [Course] = [Course(name: .androidDevelopment, skill: .kotlin),
                                    Course(name: .iOSDevelopment, skill: .swift),
                                    Course(name: .webDevelopment, skill: .javascript)]

    func getCourses() -> [Course] {
        companyCourses
    }
}

class Engineer {
    let improvement: Development
    var skills: Set<String> = []

    init(improvement: Development) {
        self.improvement = improvement
    }

    func learnCourse(from name: CourseName) {
        let course = improvement.getCourses().filter { $0.name == name }.first!
        skills.insert(course.skill.rawValue)
    }
}

class Injector {
    static func provideImprovement() -> Development {
        return Improvement()
    }
}

let fadhil = Engineer(improvement: Injector.provideImprovement())
    fadhil.learnCourse(from: .iOSDevelopment)
```

Dependency Injection

@propertyWrapper

```swift
class Improvement: Development {
    let companyCourses: [Course] = [Course(name: .androidDevelopment, skill: .kotlin),
                                    Course(name: .iOSDevelopment, skill: .swift),
                                    Course(name: .webDevelopment, skill: .javascript)]

    func getCourses() -> [Course] { companyCourses }
}

class Engineer {
    @Injected var improvement: Development
    private var skills: Set<String> = []

    func learnCourse(from name: CourseName) {
        let course = improvement.getCourses().filter { $0.name == name }.first!
        skills.insert(course.skill.rawValue)
        print(skills)
    }
}

@propertyWrapper
struct Injected {
    public var wrappedValue: Development {
        get { Injector.provideImprovement() }
        set { newValue }
    }
}

class Injector {
    static func provideImprovement() -> Development {
        Improvement()
    }
}

let fadhil = Engineer()
    fadhil.learnCourse(from: .iOSDevelopment)
```

Dependency Injection

It’s simple, isn’t it?

Why do we really use Dependency Injection?

Adopt modular design

Dependency Injection

Loosens the tight coupling between modules classes or services
Improve code maintainability
Improve code reusability
Allow mock implementation on dependency which make unit testing less painful

www.devopedia.org

www.jamesshore.com

www.raywenderlich.com

itnext.io

betterprogramming.pub

TERIMA KASIH

#BELANJA BISNIS JADI PRAKTIS

bhinneka.com