Dependency Injection

A pretentious name for a simple concept

Aaron DeGrow

What?

”… **dependency injection** is a software design pattern that implements inversion of control for resolving dependencies." ~ Wikipedia

- Typical, non-injection dependency usage: Object creates or finds (i.e., using singleton) the objects it is dependent on.
  - E.g., A class for an API service (CustomWebService) or NSUserDefaults.

```
class ClientWithoutInjection {
    let dependency: SomeClass

    init() {
        dependency = SomeClass(param1: 1, param2: 2)
    }
}
```

- With dependency injection, dependencies are passed in from external code, either via the initializer or properties/setter methods. Responsibility is shifted to code external to the class (inversion of control).

No Dependency Injection

```
// Some class that our Client class will use (e.g., an API)
class SomeClass {
    init(param1: Int, param2: Int) { }
    convenience init() {
        self.init(param1: 1, param2: 2)
    }
    func doSomething() -> Bool {
        // Actual method that does stuff (e.g., hits the network to login)
        return false
    }
}


// No Dependency Injection
class ClientWithoutInjection {
    let dependency: SomeClass

    init() {
        dependency = SomeClass(param1: 1, param2: 2)
    }

    func someMethod() -> Bool {
        return dependency.doSomething()
    }
}
```

Initializer D.I.

```
// Some class that our Client class will use (e.g., an API)
class SomeClass {
    init(param1: Int, param2: Int) { }
    convenience init() {
        self.init(param1: 1, param2: 2)
    }
    func doSomething() -> Bool {
        // Actual method that does stuff (e.g., hits the network to login)
        return false
    }
}

// With Initializer D.I.
class ClientWithInitDI {
    let dependency: SomeClass

    init(dependency: SomeClass) {
        self.dependency = dependency
    }
    convenience init() {
        let dep = SomeClass()
        self.init(dependency: dep)
    }

    func someMethod() -> Bool {
        return dependency.doSomething()
    }
}
```

Initializer D.I. with a Protocol

```swift
protocol DependencyProtocol {
    func doSomething() -> Bool
}

// Some class that our Client class will use (e.g., an API)
class SomeClass: DependencyProtocol {
    init(param1: Int, param2: Int) { }
    convenience init() {
        self.init(param1: 1, param2: 2)
    }
    func doSomething() -> Bool {
        // Actual method that does stuff (e.g., hits the network to login)
        return false
    }
}

// With Protocol and Initializer D.I.
class ClientWithProtocolInitDI {
    let dependency: DependencyProtocol

    init(dependency: DependencyProtocol) {
        self.dependency = dependency
    }
    convenience init() {
        let dep = SomeClass()
        self.init(dependency: dep)
    }

    func someMethod() -> Bool {
        return dependency.doSomething()
    }
}
```

Benefits Recap

- Flexibility / loose coupling
  - Dynamic behavior using different dependencies
  - Configuration can be externalized
  - Abstraction / future proofing
- Single responsibility / separation of concerns
- Testing
  - Mock or modified dependencies
  - No frameworks required
- Reduction in refactoring if/when dependencies change
- Parallel development (using protocol and mock objects)

Best Practices

- Use initializer method when possible (instead of property/setter)
  - Encourages immutability
  - Dependencies are ‘guaranteed’ to be available
  - Large number of dependencies passed in initializer: indicator of too much functionality in the class
- Use protocols instead of concrete classes
  - Using subclasses to override methods when unit testing is inconsistent / dangerous
    - If every single method isn’t overridden, tests will run using an object with real functionality
- Implement convenience initializer to simplify typical usage
  - Reduce work for external classes if dependencies don’t typically change or need to be configured.
    - E.g., NSNotificationCenter, NSUserDefaults, etc.

More Information

- https://medium.com/ios-os-x-development/dependency-injection-in-swift-a959c6eee0ab#.cvctqz35k
- https://www.natashatherobot.com/unit-testing-swift-dependency-injection/
- https://www.linkedin.com/pulse/dependency-injection-swift-20-johnathan-raymond
- https://www.objc.io/issues/15-testing/dependency-injection/