Introducción a Dependency Injection

Miguel Ángel Jiménez Huerta
iOS Developer

Contenido

- Metas de DI
- Terminos basicos
- Como proveer de dependencias a nuestros objetos
- Dependency Inversion
- Cómo implementar DI en nuestros proyectos.

Goals

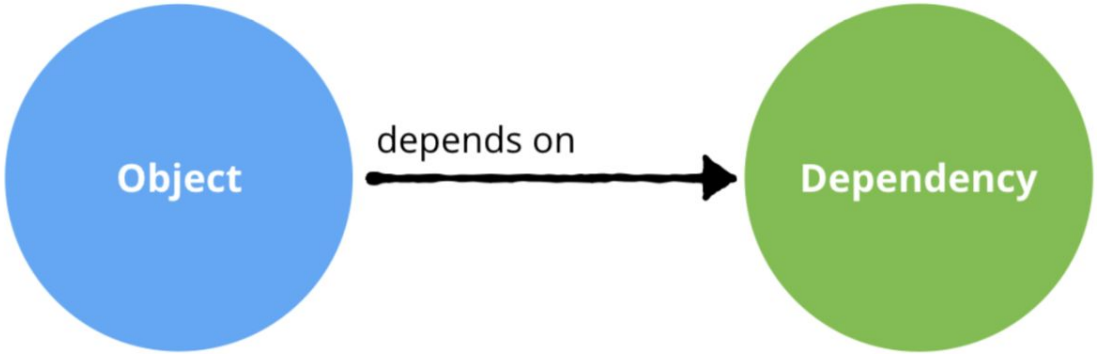
- **Maintainability**: Habilidad de cambiar partes del nuestra code-base sin introducir bugs. Habilidad de re-implementar secciones de código sin afectar el resto del código.
- **Testability**: Habilidad de poder crear Unit Tests y UI Tests que no dependan de cosas que no podemos controlar como Networking.
- **Substitutability:** Habilidad de sustituir la implementación de alguna dependencia al momento de compilar o en runtime. Poder reemplazar la implementación de alguna dependencia por una falsa o de diagnóstico.
- **Deferability:** Habilidad de posponer la decisión de qué tipo de tecnología utilizar, como qué tipo de base de datos usar desde el principio.

Goals

- **Parallel work streams:** Habilidad de tener a muchos desarrolladores trabajando en el mismo feature sin pisarse los pies.
- **Control during development:** Habilidad de poder controlar en development las implementaciones de las dependencias. Si se cae el servidor no deberíamos esperar a que lo arreglen.
- **Minimizing object lifetimes:** Como tenemos dependencias más pequeñas, su tiempo de vida es más corto y en el caso de que sea largo, no causa un gran impacto al uso de la memoria del dispositivo.
- **Reusability:** Habilidad de poder reutilizar componentes entre múltiples partes de la app o apps en las que trabajes.

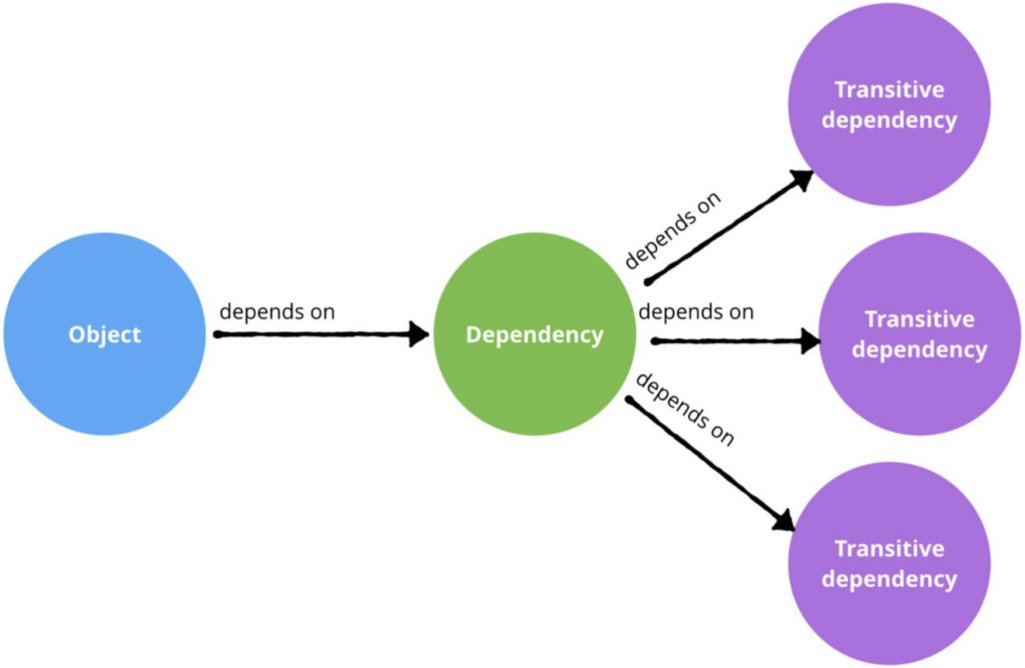
Definiendo terminos

Normalmente cuando hablamos de dependencias nos referimos a Librerías o Frameworks, en este caso una **dependencia** es un objeto que otro objeto necesita para hacer algo.



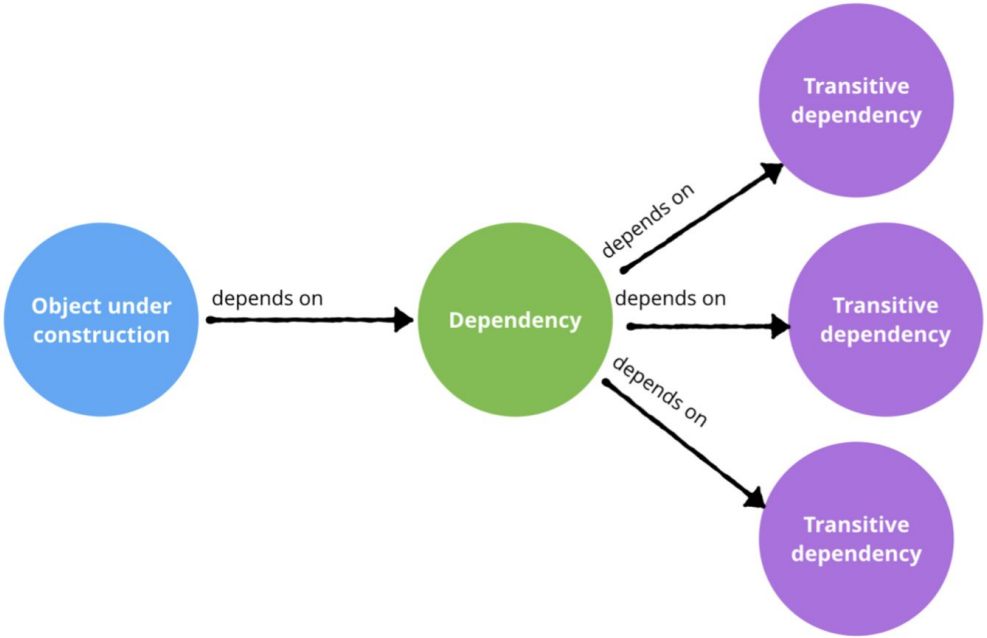
Definiendo terminos

Una dependencia puede de igual manera depender de otras dependencias. A estas dependencias se les llama **dependencias transitivas**.



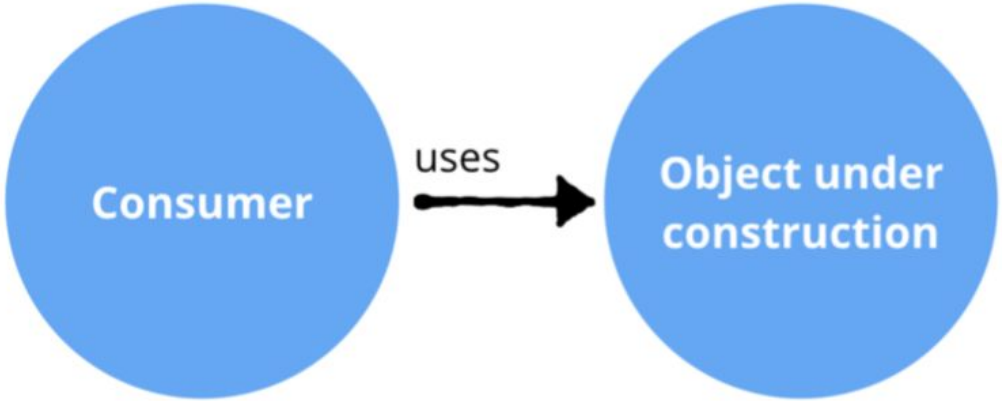
Definiendo terminos

El **objeto en construcción** es el objeto que necesita la dependencia

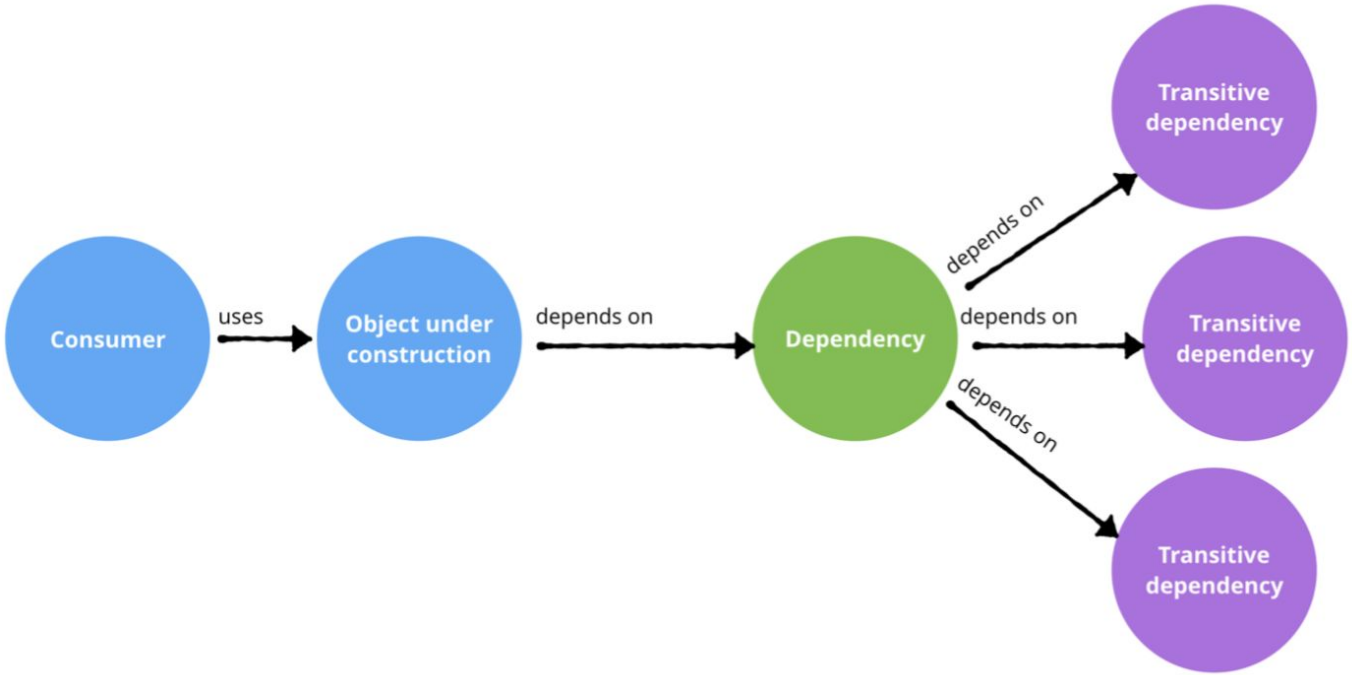


Definiendo terminos

La razón por la cual tenemos un objeto en construcción es porque alguien lo necesita y ese alguien es el **consumidor**.



Definiendo terminos



Accediendo a dependencias

Desde adentro de nuestro objeto:

Global property: Normalmente sería un objeto, variable o función que puedes acceder porque es visible globalmente. (NotificationCenter, UIApplication)**.**

Instantiation: Nuestro objeto puede instanciar directamente la dependencia que necesita. “let myObject = MyObject.init()”

Accediendo a dependencias

Desde afuera:

Constructor: Pasar una instancia de la **dependencia** en el constructor del **objeto en construcción.**

Mutable-stored property: Después de la instancia de nuestro **objeto en construcción**, alguien puede asignar nuestra **dependencia** a una variable del objeto.

Funcion: A través de una función visible de nuestro **objeto en construcción**

Accediendo dependencias

Ir al playground ...

Dependency Injection

La inyección de dependencias es un patrón de diseño que tiene el fin de manejar dependencias. Tenemos otros patrones como el **Service Locator, etc.**

La inyección de dependencias se trata de proveer las dependencias desde afuera en vez de desde adentro de nuestro **objeto en construcción**. Se dice que las dependencias son **Inyectadas** ya que vienen desde afuera.

Al obtener las dependencias desde fuera, nosotros tenemos el control de donde vienen las dependencias y cómo construirlas y customizarlas antes de inyectarlas

Inversion Control

Como nuestra clase ya no se encarga de crear las dependencias se pone en práctica este principio.

Inversión de control (Inversion of Control en inglés, IoC) es un principio de diseño de software en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales.

En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario.

La Historia de Parse

Parse was founded in 2011 by Tikhon Bernstam, Ilya Sukhar, James Yu, and Kevin Lacker, previously at Google and Y Combinator. The firm produces back-end tools for mobile developers that help mobile developers store data in the cloud, manage identity log-ins, handle push notifications and run custom code in the cloud.

On November 9, 2011, it raised $5.5 million in venture capital funding. In 2012, its tools were being used by 20,000 mobile developers and that number was growing at 40% monthly. On Sept 11, 2012, it added the ability to create custom code on the back end.

La Historia de Parse

Fast Company named Parse one of the top 50 most innovative companies of 2013.

Facebook acquired the firm for $85 million in 2013.

In 2014, Parse was reported to power 500,000 mobile apps.

On 28 January 2016, Facebook announced that it will close down Parse, with services effectively shutting down on 28 January 2017. The service operated until 30 January 2017, at which point all users needed to have migrated their applications to other platforms.Parse did open the application source code in order to allow users to perform the migration and release Parse Server. A range of vendors like Back4app, Stamplay, and AWS are able to host Parse applications, providing migration alternatives.

Principio de inversión de dependencias (Dependency Inversion)

Por sí solo el patrón de Dependency Injection es bueno, pero al combinarse con este principio podemos lograr resultados mejores.

Es la D de los principios SOLID de POO

Este principio para muchos es uno de los más importantes, la definición la podemos encontrar como:

A. Las clases de alto nivel no deberían depender de las clases de bajo nivel. Ambas deberían depender de las abstracciones.
B. Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.

Ir al Playground...

MongoDB Strengthens Mobile Offerings with Acquisition of Realm ...
https://www.mongodb.com/.../mongodb-strengthens-mobile-offerings-with-acquisitio...
Apr 24, 2019 - The **acquisition of** Realm will deepen **MongoDB's** relationship with developer ... Furthermore, **actual** results may differ materially from those ...

MongoDB will acquire mobile database startup Realm for $39 million ...
https://www.businessinsider.com/mongodb-realm-acquisition-price-2019-4
Apr 24, 2019 - **MongoDB** announced Wednesday that it plans to **acquire** mobile database startup Realm for $39 million. ... With its sights set on doubling down on the world of mobile, **MongoDB** announced Wednesday that it will **acquire** Realm for $39 million. ... Its database has been used to power apps from ...

MongoDB to acquire open-source mobile database Realm for $39 ...
https://techcrunch.com/.../mongodb-to-acquire-open-source-mobile-database-realm-st...
Apr 24, 2019 - **MongoDB** announced today that it is **acquiring** Realm, an open-source database geared for mobile applications, for $39 million. The startup had raised just over $40 million before being **acquired** today. ... With Realm, **Mongo** gets a strong mobile solution, adding to **MongoDB** Mobile, and it ...

Database Company MongoDB Announces $39M Acquisition Of Realm
https://www.benzinga.com/.../database-company-mongodb-announces-39m-acquisitio...
Apr 24, 2019 - **MongoDB** (NASDAQ: MDB) has entered an agreement to buy Realm, the company behind the Realm mobile database and Synchronization ...

Como implementar Dependency Injection?

Para implementar DI en nuestros proyectos les mostrare 4 maneras distintas

- On Demand:
- Factories:
- Dependency Container:
- Container hierarchy

On Demand

Nos sirve para aprender a utilizar DI y en proyectos muy simples.

En este caso cuando el **consumidor** necesita un nuevo **objeto en construcción** este crea o encuentra las dependencias necesarias para realizar la instancia, al hacer esto nuestro **consumidor** debe contener todas las dependencias necesarias para esto.

Nuestro **consumidor** necesita conocer los tipos concretos con los que trabaja.

Para sustituir la **dependencia** por otra, tenemos que buscar todos los lugares donde la utilizamos para reemplazarla.

On Demand

Pros:

- Facil de entender y de aplicar.
- Podemos testear nuestros **objetos en construcción.**
- Rapida de aplicar.

Cons:

- La creación de dependencias está descentralizada por lo que podemos repetir mucho código.
- Los **consumidores** conocen de las dependencias y puede que existan demasiadas haciendo que crezcan mucho.
- No es fácil de escalar conforme el proyecto crece.

Ir al playground...

Factories

Este método se trata de centralizar la instancia de las dependencias. Nos ayuda con las dependencias efímeras. No nos sirve para dependencias que deban vivir por mucho tiempo.

Una factory class solo debe contener **factory methods** que nos ayudan a crear **dependencias** o **objectos en construcción**.

Estas nos ayudan a que los **consumidores** no tengan que conocer cómo se crea el grafo de dependencias y nos ayuda a quitar toda esa lógica y poderla reutilizar en varios **consumidores.**

Al utilizar los factory methods para resolver una dependencia, logramos que solo una función en todo nuestro código sepa como resolverlo a la implementación. Y si deseamos cambiarlo solo necesitamos modificarlo en ese lugar.

Si hay algun parametro que no tengamos al momento de utilizar este método, lo podemos pasar como parámetro de la función.

Factories

Pros:

- Nuestras dependencias efímeras están centralizadas y podemos hacer cambios muy fácilmente.
- Podemos sustituir muchas dependencias para hacer testing cambiando un par de líneas de código.
- Los consumidores son más resilientes a cambios ya que no conocen cómo construir el grafo de dependencias. Y les quitamos una responsabilidad de encima.
- Si alguien el tu equipo quiere conocer cómo se crea alguna dependencia sólo tiene que ir a la factory class y toda la lógica estará ahí.

Cons:

- Si usas una sola factory class esta puede crecer mucho ya que contendrá todas las dependencias.
- No tienen estado y solo trabaja con dependencias efímeras. No contempla otro tipo de dependencias.

Ir al playground...

Dependency Container

Es muy parecido a las **factory classes** solo que este puede contener dependencias que no sean efímeras dentro de las variables de la clase. Estas **dependencias** se pueden reutilizar para varios **objetos en construcción.**

Las **long lived dependencies** se pueden crear al momento de inicializar el **Container** o pueden ser lazy para que se creen hasta que se necesiten.

Solo debemos crear una sola instancia de nuestro **Container** ya que este ya contiene las dependencias que deseamos utilizar, por lo que se recomienda colocarlo en el **AppDelegate** o otro objeto que viva el tiempo que queremos que nuestras dependencias vivan.

Dependency Container

Pros:

- Nuestro container sabe como crear el grafo de dependencias de todo nuestro proyecto y remueve la necesidad de que nuestros consumidores sepan cómo se construye el grafo.
- Manejan singletons, por lo que no necesitamos tener variables globales para accederlos, ahora están centralizados en un solo lugar.
- Puedes cambiar el grafo de dependencias dentro del Container y no afecta en nada el resto del código.

Cons:

- Es fácil terminar con un container masivo. Se puede separar en múltiples Containers para evitar este problema.

Frameworks

Hay varios frameworks que nos pueden ayudar a aplicar DI más fácilmente en nuestros proyectos.

- Swinject: https://github.com/Swinject/Swinject
- Weaver: https://github.com/scribd/Weaver
- Typhoon: https://github.com/appsquickly/typhoon

Hay que ser pragmáticos

Debemos saber reconocer en qué momentos nos conviene utilizar dependency injection, qué partes de nuestro código de verdad necesitan diseñarse para poder ser reemplazadas, todo lo que programamos cuesta tiempo y hay que saber dónde invertirlo para obtener el mayor beneficio. No todo lo que hacemos puede ser flexible y obviamente no todo debe ser rígido, debemos encontrar un equilibrio.

Bibliografia

https://clean-swift.com/dependency-inversion-a-little-swifty-architecture/

https://github.com/Swinject/Swinject

https://devexperto.com/principio-de-inversion-de-dependencias/

https://en.wikipedia.org/wiki/Parse_(platform)

https://en.wikipedia.org/wiki/Dependency_injection

https://store.raywenderlich.com/products/advanced-ios-app-architecture