МЕНЕДЖЕРЫ ЗАВИСИМОСТЕЙ

Существуют для того чтобы пользоваться чужим кодом, сторонними библиотека и фреймворками.

Нужны чтобы:

* Стандартизировать работу с библиотеками;
* Облегчить обновление сторонних библиотек;
* Синхронизировать установку внутренних зависимостей библиотек – все устанавливается корректно, во вторых нужной версии.

Cocoapods

Централизованный менеджер зависимостей, написанный Ruby, у cococapods есть собственный понятный сайт для поиска зависимостей. На компьютер устанавливается один раз.

Чтобы загрузить в проект нужно:

* Ввести в терминале sudo gem install cocoapods – это вводится в случае если cocoapods ранее не был установлен на пк. Это команда устанавливает его на пк;
* Вводим cd и переносим папку с проектом в терминал;
* Заходим в папку с проектом через ls;
* Вводим pod init, после этого появляется podfile – его лучше открывать через xcode;
* Внутрь podfile вводим(копируем с сайта) название и версию библиотеки;
* После этих действий вводим в терминале pod install. Идет установка библиотеки в проект, после этого появляется белый проект в папке и теперь мы работаем с ним;
* После установки у нас появляется папка Pods ее лучше помечать .gitignore ибо она много весит а память в Git ограничена.

Чтобы удалить cocoapods из проекта:

* pod deintegrate;
* pod clean;
* После этих команд можно удалять остатки проекта в ручную.

Плюсы cocoapods:

* Централизованная система – удобно обновлять все зависимости;
* Почти все библиотеки поддерживают;
* Есть возможность dynamic/static установки библиотек.

Минусы:

* Каждый раз при компиляции будут грузиться все библиотеки;
* Создается отдельный файл xcodeworkspace;
* Долгая первая установка потому что cocoapods подгружает репозиторий Specs.

Carthage – картейдж

* Децентрализованный менеджер зависимостей (все библиотеки живут сами по себе, нет большой папки с проектами);
* Open-Source написан на swift;
* Не каждая зависимость поддерживает Carthage.

Плюсы:

* Проект собирается быстрее чем если использовать cocoapods;
* Проект сам по себе остается неизменным, добавляются лишь фреймворки ;
* Есть возможность dynamic/static установки библиотек.

Минусы:

* Много лишних шагов при установки;
* Далеко не все библиотеки поддерживают.

Фреймворки бывают статические и динамические:

* Динамические – библиотеки, в отличие от статических, вместо того, чтобы копироваться в один монолитный исполняемый файл, загружаются в память, когда они действительно необходимы. Это может произойти либо во время загрузки, либо во время выполнения;
* Статические – библиотеки представляют собой коллекции объектных файлов. В свою очередь, объектный файл - это просто имя файла, который выходит из компилятора и содержит машинный код.

Использование статических фреймворков вместо динамических фреймворков уменьшает размер приложения и значительно сокращает время запуска приложения.

Разница между фреймворком и библиотекой:

Библиотека – это место где куски кода решающие какую-то проблему лежат на удаленном сервере (пример GitHub) или диске. Имеет единственную ответственность.

Фреймворк – это пакет, который может содержать такие ресурсы, как динамические библиотеки, строки, заголовки, изображения, раскадровки и т.д. с небольшими изменениями в его структуре он может даже содержать другие фреймворки