**Auto Layout**
- Constraint - 각 뷰의 거리, 길이, 위치 등을 표현하기 위한 제약
- View Boundary(뷰의 경계)
    - Leading(좌), Trailing(우), Top(상), Bottom(하)

**Tip**
- 화면 배치의 기준이 되는 뷰를 잡고 시작하는 것이 좋아요
- 제약사항은 자신의 의도에 따라 천차만별

**Auto Layout Manu**
- Stack : 해당 객체들을 하나의 스택 뷰로 변환
- Align : 객체 정렬에 관한 제약사항 추가
- Pin : 객체의 크기 및 객체 간 거리에 관한 제약사항 추가
- Resolve Issues : IB 구성 중 발견된 오토레이아웃 관련 문제 해결

**Align Manu**
- 객체가 속한 화면(Container)에서 정렬 제약
    - Horizontally in Container
    - Vertically in Container
- 다른 객체와의 정렬 제약
    - Leading Edges
    - Trailing Edges
    - Top Edges
    - Bottom Edges
    - Horizontal Centers
    - Vertical Centers
    - Baselines

**Pin Manu**
- 인접 객체와의 고정거리 제약
- 가로, 세로 크기
    - Width
    - Height
- 다른 객체와의 크기 비율 제약
    - Equal Widths
    - Equal Heights
- 가로,세로 비율 제약
    - Aspect Ratio
- 다른 객체와의 정렬
    - Align
- **모든 단위는 포인트! (pt)**

**Resolve Auto Layout Issues Manu**
- 설정한 제약사항에 따라 프레임(자동)수정
- 현재 화면의 프레임에 따라 제약사항(자동)수정
- 부족한 제약사항 (자동)추가
- IB가 제안하는 제약사항으로 제약사항 초기화(자동)
- 제약사항 모두 삭제

Update Frames, Clear Constraints 외에는 사용 자제

**Size Class**
- 특정화면 크기와 방향을 나타내는 것이 아니라 폭과 높이를 Compact, Regular라는 사이즈로 구분

참조 https://goo.gl/cHzzoJ

**NSObject의 생명주기**
- IOS의 모든 객체는 NSObject에 정의된대로 생명주기를 갖습니다.
- 기본적으로는 메모리에 생성될 때 alloc, 메모리에서 해제될때 dealloc 메소드를 시스템에서 호출합니다.
- alloc과 dealloc 메소드를 재정의한다면 언제 메모리에 생성되고 해제되었는지 알 수있으며 , 적절한 처리도 해줄수있습니다.

**UIViewController의 생명주기 메소드**
- UIViewController의 생명주기를 알고 있다면 적절한 타이밍에 원하는 것들을 세팅해 줄 수 있습니다.
  예) 화면이 보이지 전에 사진을 셋팅, 화면이 나타난 후에 특정 애니메이션 재생 등
- 시스템에서 각각 메소드가 갖는 생명주기 타이밍에 맞게 메소드를 호출하게 됩니다.
- 해당 메소드들은 프로그래머가 직접 호출하지 말아야 합니다.
- 오버라이드 하는 메소드이므로 꼭 해당 메소드 내에서 [super 메소드이름]을 통해 기존 메소드를 꼭 호출해 주어야 합니다.

**UIViewController의 대표적인 생명주기 메소드(호출 순서대로)**

`- (void)loadView`
- UIViewController의 view가 생성될 때 호출

`- (void)viewDidLoad`
- UIViewController의 인스턴스화 된 직후(메모리에 객체가 올라간 직후) 호출, 처음 한번 세팅해 주어야 하는 값들을 넣기에 적절

`- (void)viewWillAppear:(BOOL)animated`
- UIViewController의 view가 화면에 보여지기 직전에 호출, 화면이 보여지기 전에 준비할 때 사용, animated파라미터는 뷰가 애니메이션을 동반하여 보여지게 되는지 시스템에서 전달해주는 불리언 값

`- (void)viewWillLayoutSubviews`
- view의 하위뷰들의 레이아웃이 결정되기 직전 호출

`- (void)viewDidLayoutSubviews`
- view의 하위뷰들의 레이아웃이 결정된 후 호출, 주로 view의 하위뷰들이 사이즈 조정이 필요할 때 호출

`- (void)viewDidAppear:(BOOL)animated`
- view가 화면에 보여진 직후에 호출, 화면이 표시된 이후 애니메이션 들을 보여주고 싶을 때 유용

`- (void)viewWillDisappear:(BOOL)animated`
- view가 화면에서 사라지기 직전에 호출

`- (void)viewDidDisappear:(BOOL)animated`
- view가 화면에서 사라지기 직후에 호출

**어플리케이션 상태변화(Application Life Cycle)**
- **ios 어플리케이션의 상태변화 모식도**
    - Not Running : 실행되지 않았거나, 시스템에 의해 종료된 상태
    - Inactive : 실행 중이지만 이벤트를 받고있지 않은 상태, 예를들어 앱 실행 중 미리알림 또는 일정 얼럿이 화면에 덮여서 앱이 실질적으로 이벤트를 받지 못하는 상태 들을 뜻합니다.
    - Active : 어플리케이션이 실질적으로 활동하고 있는 상태
    - Background : 백그라운드 상태에서 실질적인 동작을 하고 있는 상태, 예를 들어 백그라운드에서 음악을 실행하거나, 걸어온 길을 트래킹하는 등의 동작을 뜻합니다.
    - Suspended : 백그라운드 상태에서 활동을 멈춘 상태, 빠른 재실행을 위하여 메모리에 적재된 상태이지만 실질적으로 동작하고 있지는 않습니다. 메모리가 부족할 때 비로소 시스템이 강제종료하게 됩니다.
- **ios 어플리케이션 상태변화에 따른 시스템의 App Delegate 객체 메소드 호출**
    - 우리는 대부분의 상태변화를 app delegate 객체에 호출되는 메소드를 오버라이드하여 알아챌 수 있습니다. 상태변화에 따라 호출되는 메소드는 아래와 같습니다.
- application:willFinishLaunchingWithOptions: 어플리케이션이 최초 실행될때 호출되는 메소드입니다.
- application:didFinishLaunchingWithOptions: 어플리케이션이 실행된 직후 사용자의 화면에 보여지기 직전에 호출됩니다. 어플리케이션 실행에 필요한 것들을 최초로 초기화할때 사용됩니다.
- applicationDidBecomeActive: 어플리케이션이 Active상태로 전환된 직후 호출됩니다.
- applicationWillResignActive: 어플리케이션이 inactive상태로 전환되기 직전 호출됩니다. 실질적으로 화면에서 사라지는 시점인 경우가 많기 때문에 현재의 어플리케이션 상황등을 저장할때 주로 사용됩니다.
- applicationDidEnterBackground: 어플리케이션이 백그라운드 상태로 전환된 직후 호출됩니다.
- applicationWillEnterForeground: 어플리케이션이 Active 상태가 되기 직전에, 화면에 보여지기 직전의 시점에 호출됩니다.
- applicationWillTerminate: 어플리케이션이 종료되기 직전에 호출되는 메소드입니다.