

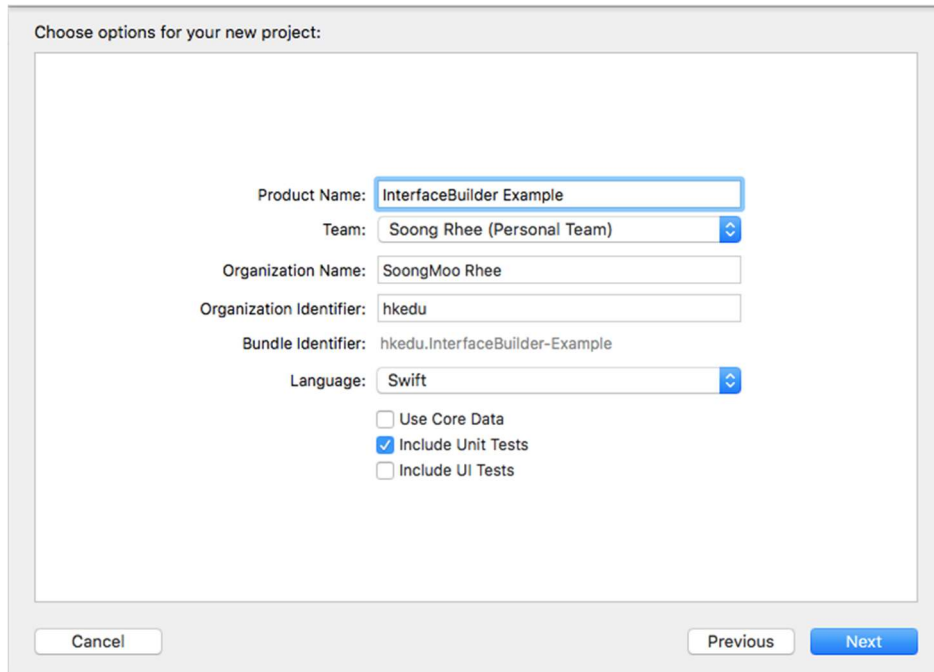
로또는 1~45 사이의 숫자 중 중복되는 숫자 없이 6 개를 뽑는 게임이다.  
그리고 앱은 5 개의 게임으로 구성되어 있다,

앱 UI는 단순하다. 먼저 로또 번호를 추출하는 버튼과 테이블을 만들고 여러 줄에 번호가 나열되도록 만든다. 마지막으로 랜덤 번호를 추출하도록 만든다.

## 21-1. 앱 UI 구성하기

1. Xcode 를 실행한 후 Single View Application 을 선택한다.

프로젝트명은 "LottoDrawExample" 라는 이름으로 새 프로젝트를 만든다



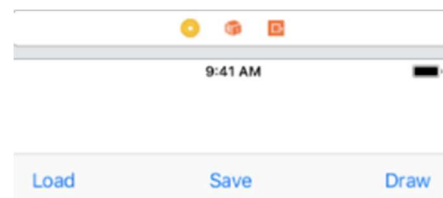
Main.storyboard 파일을 선택해서 오브젝트 라이브러리에 있는 [Toolbar]를 선택한 후 드래그 앤 드롭해서 추가한다.

[Toolbar]에는 [Bar Button Item]하나가 함께 추가된다.

추가한 [Toolbar]에 [Bar Button Item]을 2 개 더 드래그 앤 드롭해 넣는다.

그리고 각 버튼을 더블 클릭해 "Load", "Save", "Draw"라는 이름으로 변경한다.

[Flexible Space Bar Button Item]이라는 이름으로 각각 버튼 사이에 하나씩 총 2 개를 추가하면 된다.

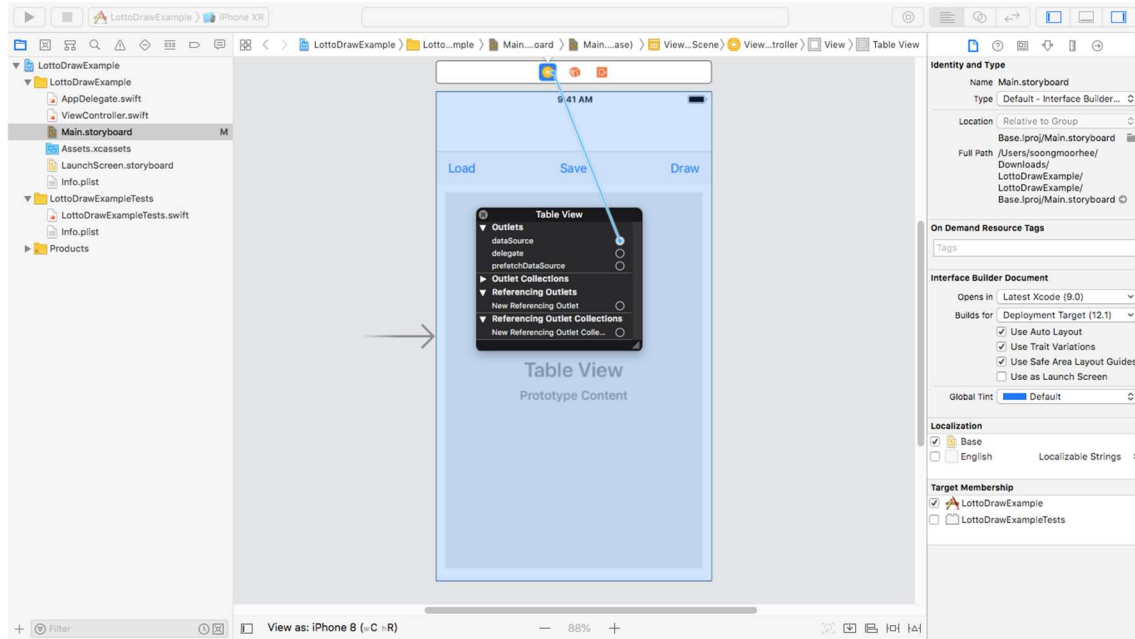


툴바가 준비되었으니 이번에는 테이블 뷰를 추가해 보자.

오브젝트 라이브러리에서 [Table View]를 드래그 앤 드롭해서 추가한다,

[Table View]가 화면에 딱 차도록 크기를 조절한다.

[Table View]를 마우스 오른쪽 버튼으로 클릭하면 아웃렛 목록이 나타난다,

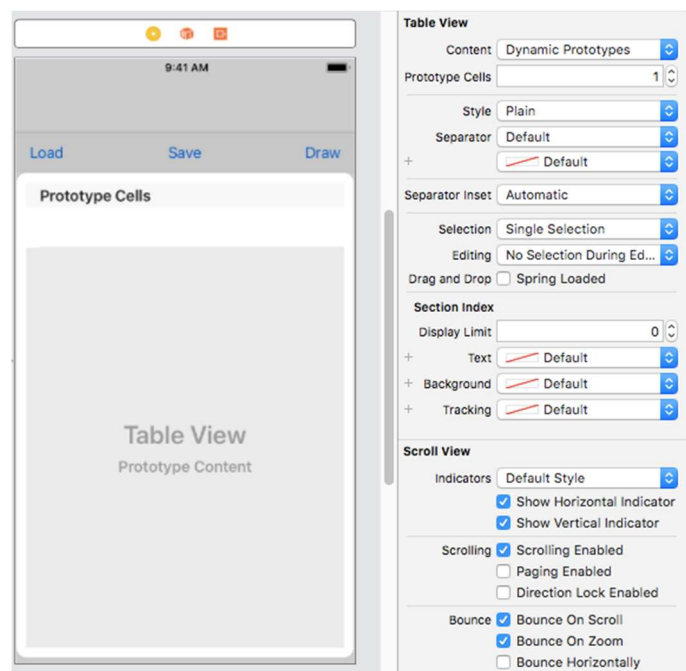


[dataSource]를 드래그 앤 드롭해서 [View Controller]에 연결해 준다.

[dataSource]를 연결했으니 테이블에 들어갈 셀에 관련된 설정을 해보자.

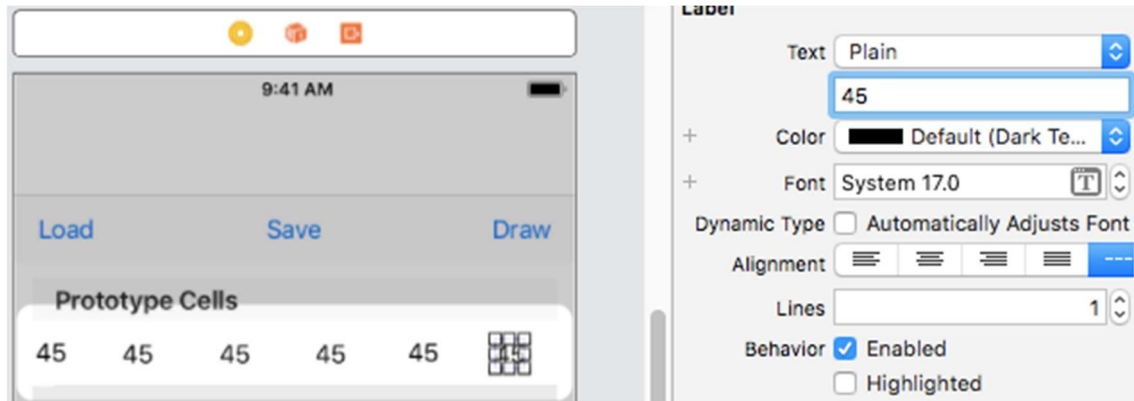
인터페이스 빌더 화면에서 [Table View]를 선택하고 Attributes inspector 를 선택한다,

맨 위에 [Content]에 "Dynamic Prototypes"라고 설정되어 있고 [Prototype Cells] 항목 값을 0에서 1로 변경한다.,



Prototype Cell 에 데이터를 표현할 수 있도록 UI 요소들을 추가해보자. 오브젝트 라이브러리에서 [Label]을 6 개 추가한다.

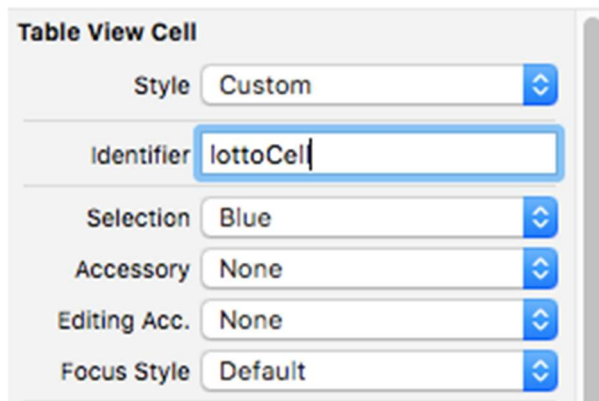
그리고 6 개 [Label]을 모두 선택한 후 Attributes inspector 의 [Label] -> [Text]의 두 번째 항목의 기본 텍스트 값을 '45'로 변경한다.



```
1  override func didReceiveMemoryWarning() {  
2      super.didReceiveMemoryWarning()  
3      // Dispose of any resources that can be recreated.  
4  }  
5  
6  // 해당 테이블의 섹션 개수 - 필수 메서드  
7  func numberOfSectionsInTableView(_ tableView: UITableView) -> Int {  
8      return 1  
9  }  
10  
11 // 해당 섹션의 셀 개수 - 필수 메서드  
12 func tableView(_ tableView: UITableView, numberOfRowsInSectionSection section:  
13 Int) -> Int {  
14     return 0  
15 }  
16  
17 // 셀 생성과 반환 - 필수 메서드  
18 func tableView(_ tableView: UITableView, cellForRowAtIndexPath indexPath:  
19 IndexPath) -> UITableViewCell {  
20     let cell = UITableViewCell()  
21  
22     return cell  
23 }  
24  
25  
26  
27
```

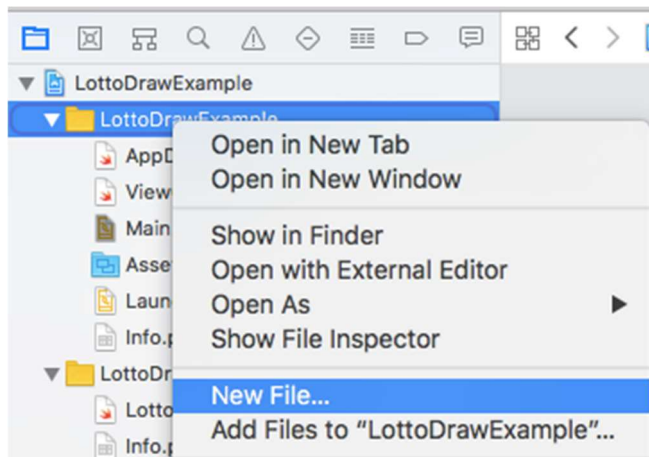
실행된 화면을 보면 아까 만들어 놓은 6 개의 [Label]이 보이지 않는 것일 수 있다. 이번 단계에서는 셀을 사용할 수 있게 만들어 보자.

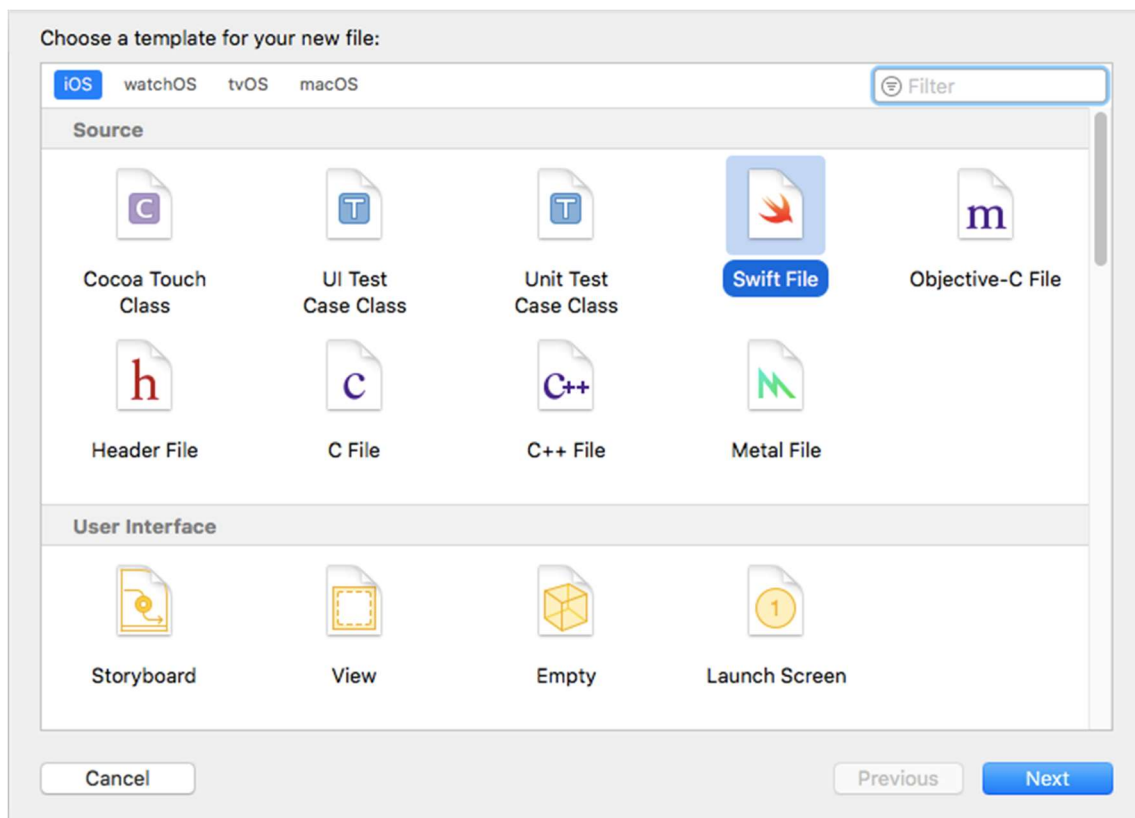
Main.storyBoard 에서 조금 전에 만든 Prototype Cell (Table View Cell)을 선택한다. 그리고 Attributes inspector 에서 [Table View Cell] -> [Identifier](셀 식별자 혹은 재사용 식별자)항목의 값을 'lottoCell'이라고 한다,



이제 셀을 사용할 때 필요한 클래스 파일을 만들어보자.

[프로젝트 이름] 폴더를 마우스 오른쪽 버튼으로 클릭해 [New File]을 선택한다.





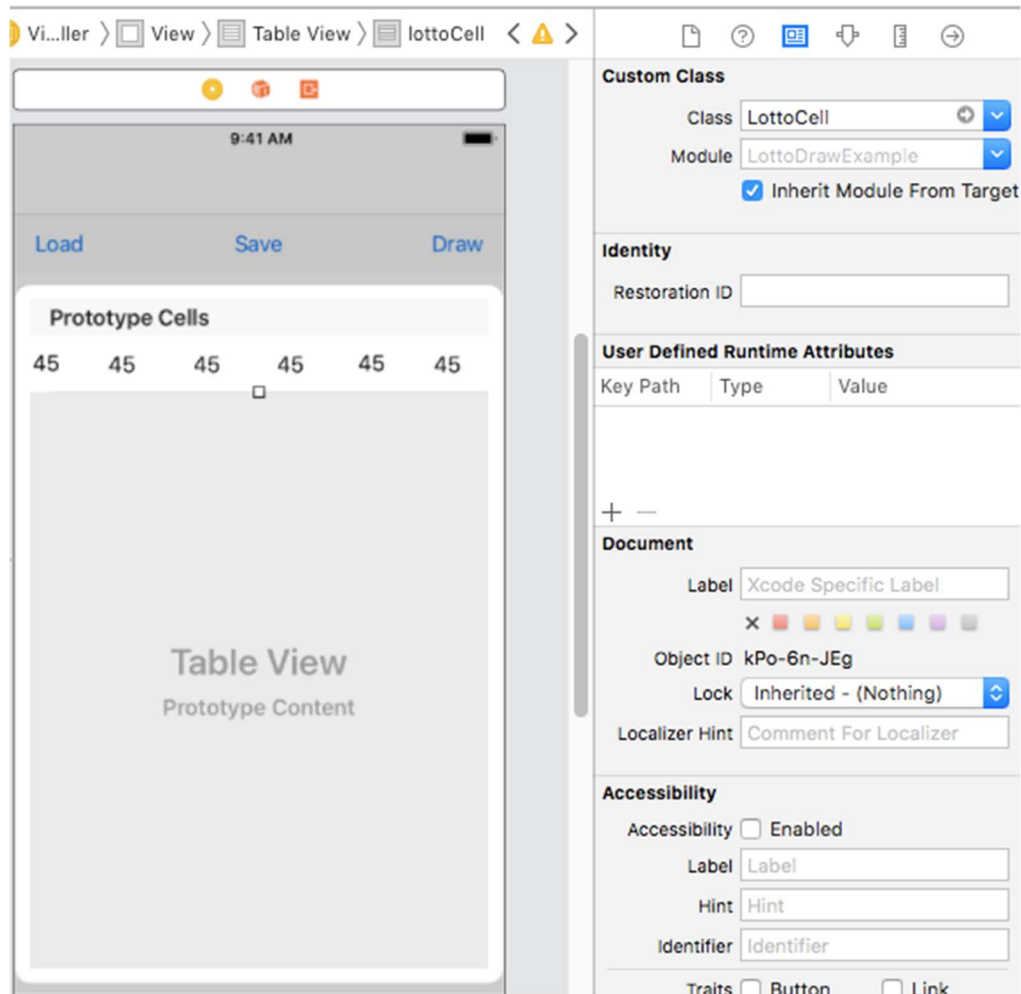
파일을 저장할 위치로 프로젝트 폴더(LottoDrawExample)를 선택하고 파일 이름을 'LottoCell'이라고 입력한 후 <Create>버튼을 클릭한다.

Project navigator 에서 LottoCell.swift 파일을 선택한후 아래 내용을 추가한다.

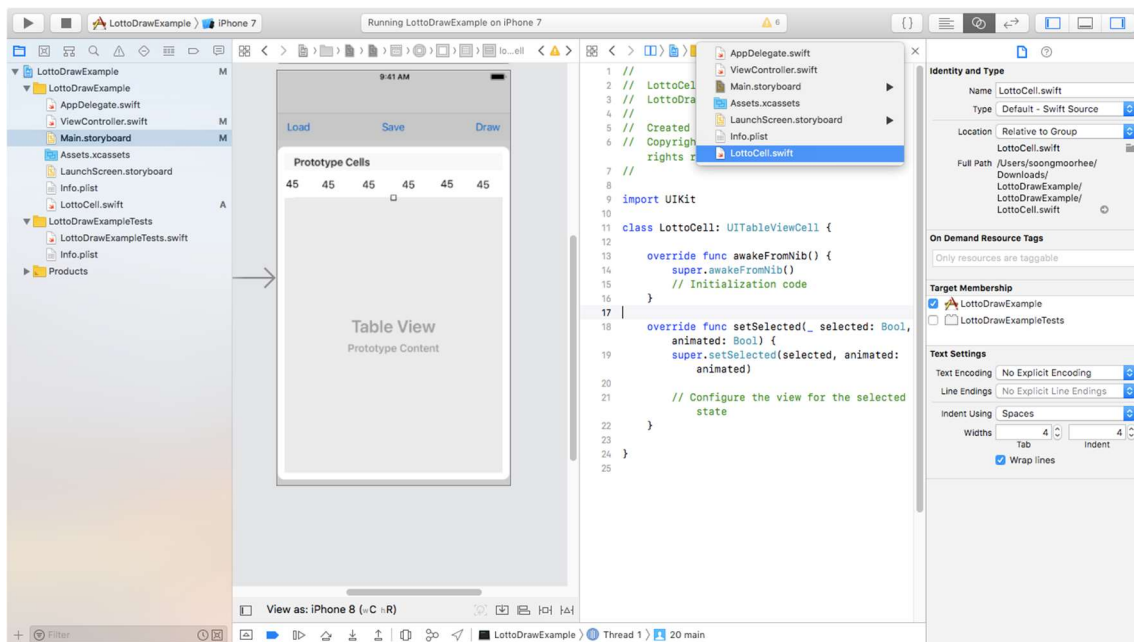
```
1  class LottoCell: UITableViewCell {
2
3      override func awakeFromNib() {
4          super.awakeFromNib()
5          // Initialization code
6      }
7
8      override func setSelected(_ selected: Bool, animated: Bool) {
9          super.setSelected(selected, animated: animated)
10
11          // Configure the view for the selected state
12      }
13  }
14
15
```

완성된 코드를 아웃렛에 연결 해보자. Main.storyboard 에서 아까만든 Prototype Cell(LottoCell)을 다시선택한다.

오른쪽 유틸리티 영역에서 Identity inspector 를 선택한다. 그리고 [Custom Class] -> [Class]항목 값을 미리 만들어둔 'LottoCell'로 선택한다,



커스텀 클래스를 연결했으니 에디터 영역의 레이아웃을 "LottoCell.swift"를 선택해 준다.



"LottoCell.swift" 파일에 Prototype Cell 에 있는 6 개 [Label] 각각의 아웃렛을 만들어 준다.

각각의 아웃렛 변수의 이름은 "number1~number6"으로 정한다.



아웃렛 변수를 만들었으니 데이터를 보여줄 준비는 끝났다. 로또 번호를 보여주기 전에 임의의 숫자를 출력해 보자.

<pre>// 해당 섹션의 셀 개수 - 필수 메서드 func tableView(_ tableView: UITableView, numberOfRowsInSectionSection section: Int) -&gt; Int {     return 1 }  // 셀 생성과 반환 - 필수 메서드 func tableView(_ tableView: UITableView, cellForRowAtIndexPath indexPath: IndexPath) -&gt; UITableViewCell {</pre>	
--	--

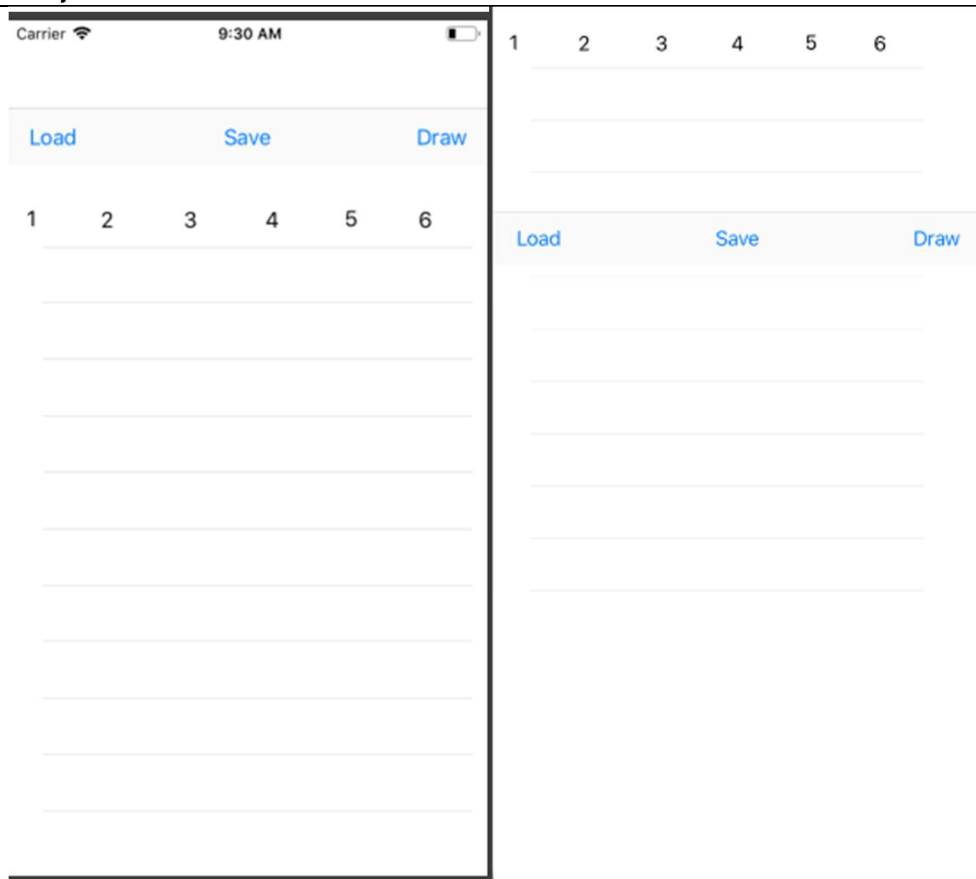


```
let cell = tableView.dequeueReusableCell(withIdentifier: "lottoCell", for: indexPath as IndexPath) as! LottoCell
```

```
cell.number1.text = "1"  
cell.number2.text = "2"  
cell.number3.text = "3"  
cell.number4.text = "4"  
cell.number5.text = "5"  
cell.number6.text = "6"
```

```
return cell
```

```
}
```



아이폰 7 은 정상적으로 출력이 되지만 나머지 아이폰에서는 잘 나오지 않을 수 있다.

## 21-2. 화면 크기에 반응하는 앱

기기에 따라 적절한 크기나 간격을 유지하면서 UI 가 구현되지 않는다. 이러한 문제는 앱 개발할 때 항상 발생한다.

iOS 와 Xcode 의 발전과 더불어 오토 레이아웃이 등장으로 손쉽게 해결할 수 있게 되었다.

1. Xcode 를 실행한 후 Single View Application 을 선택한다.

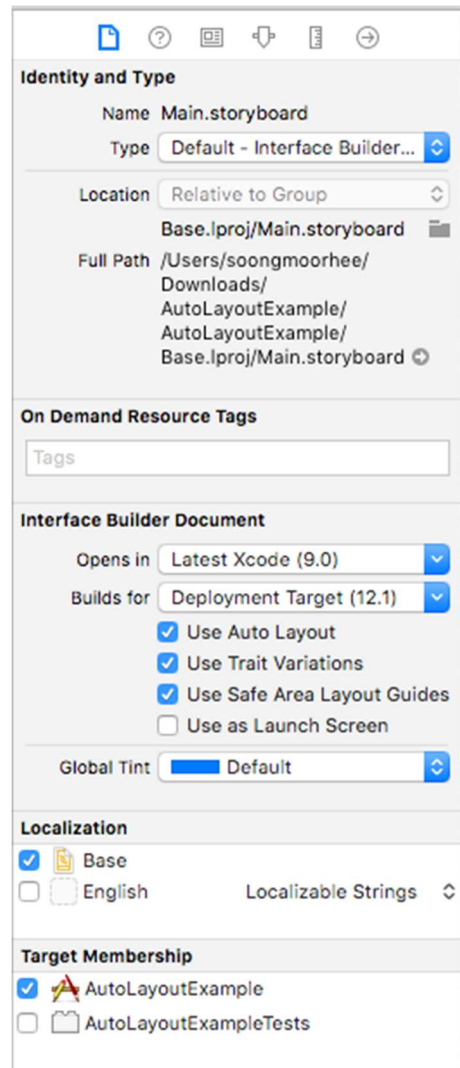
프로젝트명은 "AutoLayoutExample" 라는 이름으로 새 프로젝트를 만든다

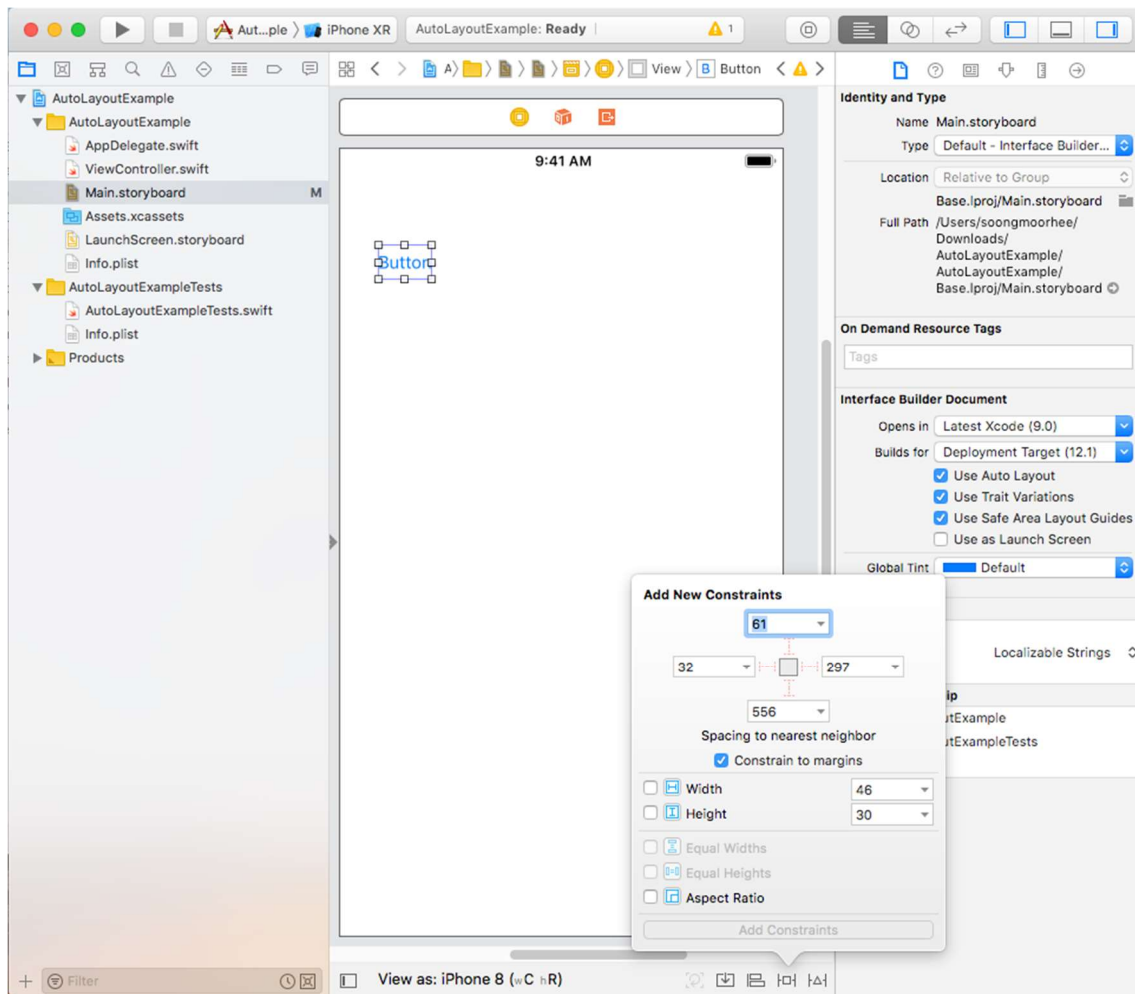
프로젝트가 새로 생성되면 Main.storyboard 파일을 선택하고 Document Outline 에서 [View Controller]를 선택한 후 File inspector 를 살펴본다.

[Interface Builder Document]항목을 보면 [Use Auto Layout], [Use Test Variations]이라는 2개의 항목이 존재하는 것을 확인할 수 있다. 첫번째 옵션은 오토레이아웃을 사용하기 위한 옵션인 걸 보면 알 수 있다. 두번째 옵션은 Trait Variations은 '특성 변화'라고 번역할 수 있으며, 하나의 스토리보드 안에 여러 다른 기기를 위한 오토 레이아웃을 설정하고 저장할 수 있도록 해주는 옵션이다. iOS 8 부터 사용 가능한 기능으로, 이전 버전의 [Use Size Class]라는 이름이 변경된 것이다. 오토 레이아웃을 실습할 때 꼭 꺼야 하는 기능이 아니므로 옵션을 설정한 상태로 두고 실습을 진행한다,

하지만 [Use Auto Layout]옵션은 꼭 체크를 해야 하므로 옵션 설정이 해제되어 있다면 체크한 다음 다음 단계로 넘어간다.

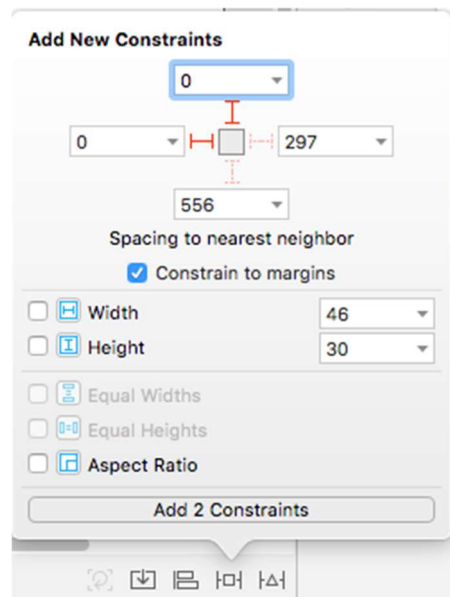
Main.storyboard 의 [view Controller]에 [Button]을 추가하고, 해당버튼을 선택한 상태로 인터페이스를 빌더 아래에 있는 <Pin> 버튼 클릭한다.





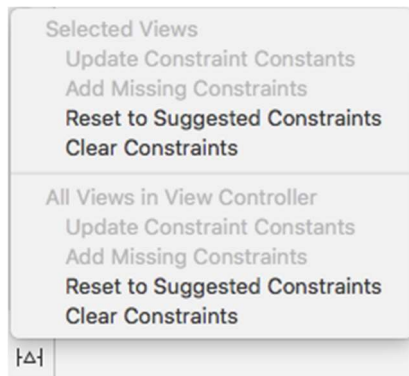
<Pin> 버튼을 클릭하면 Add New Constraints 라는 창이 나타난다. 이때 [Spacing to nearest neighbor] 항목의 값을 변경해서 이웃한 UI 요소와 간격을 지정할 수 있다. 이때 constraint(제약)를 만든다고 한다.


[Spacing to nearest neighbor] 항목의 위쪽과 왼쪽의 설정 값을 '0'으로 지정해보자. 값을 입력하면 붉은색 점선이 실선으로 활성화된다. 활성화된 것을 확인하고 맨 아래에 있는 <Add 2 Constraints> 버튼을 클릭한다.



Constraint 를 추가하면 화면상에 변할 위치가 표시된다.


여러개가 생성되면 UI 요소 각각의 최종 위치를 확인하기 힘들다. 따라서 UI 요소의 위치가 최종 위치로 이동해서 나타나게 변경해 본다.



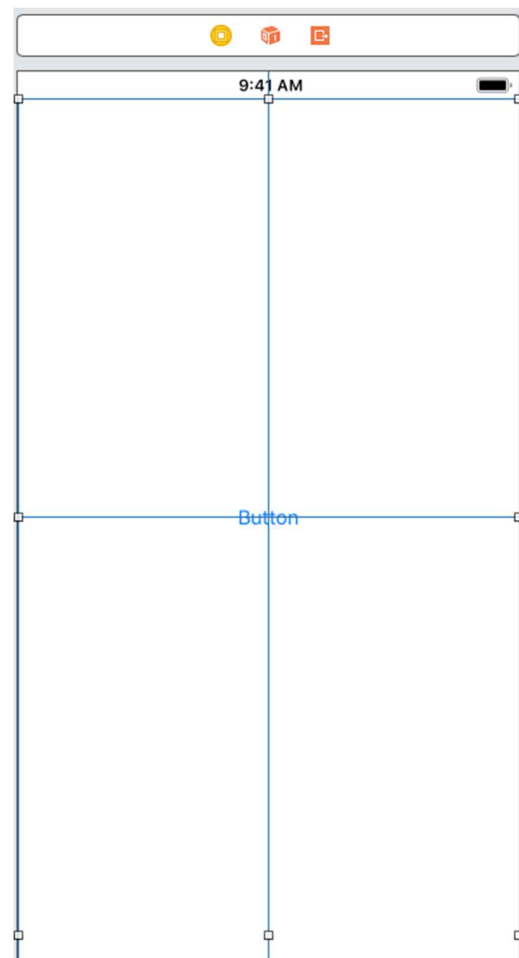
인터페이스 빌더 아래에 있는 constraint 관련 버튼 중 제일 오른쪽에 있는 <Resolve Auto Layout Issues>  버튼을 클릭하고 [Selected Views]항목 아래의 [Update Frames]을 선택한다, 그러면 [Button]오브젝트의 위치가 위로 변경된다.

현재 버전에서는 자동으로 적용되고 있다.

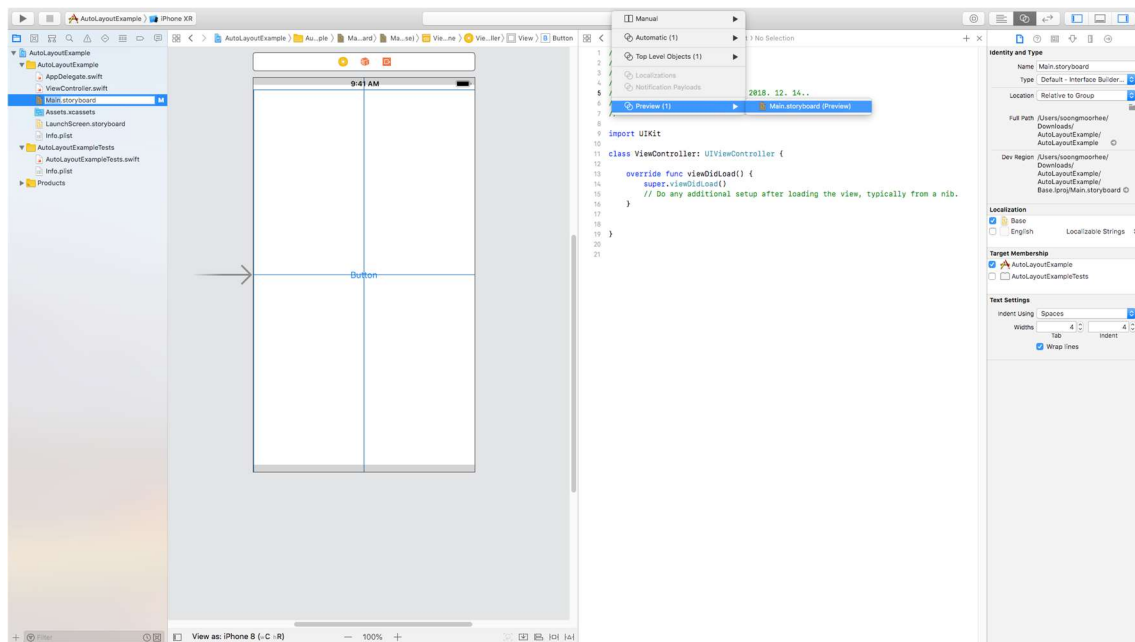
이번에는 뷰 컨트롤러의 화면 범위를 의미하는 컨테이너를 기준으로 버튼을 중앙에 정렬해보겠다.

[Button]을 선택하고 인터페이스 빌더 아래의 <Align>  버튼을 클릭해 Add New Aligment Constraints 창을 연다.

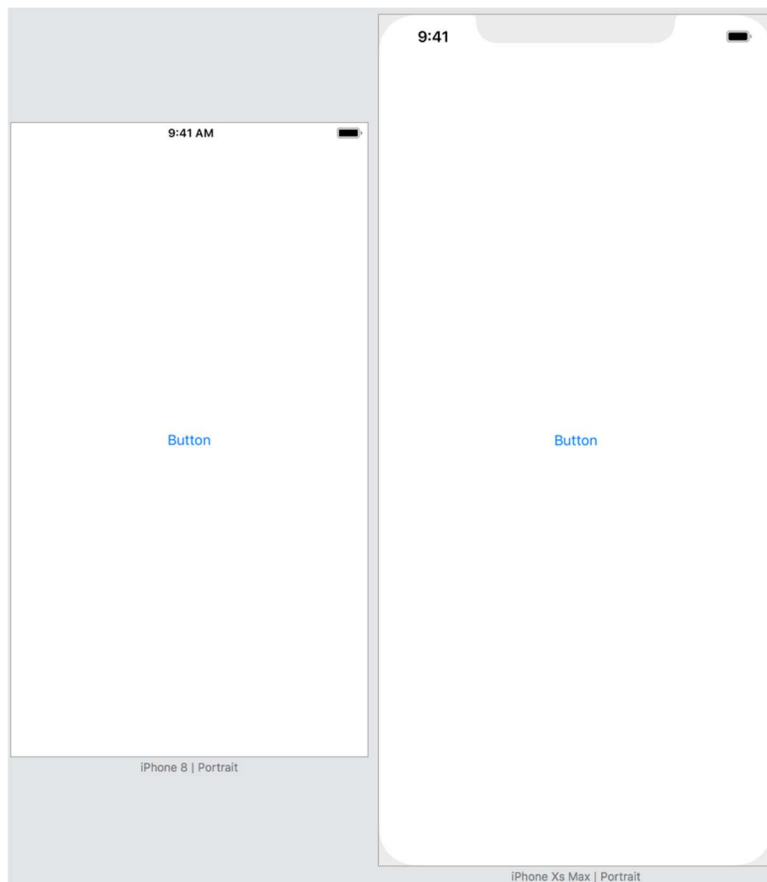
그리고 [Horizontally in Container], [Vertically in Constraints]항목에 체크하고 수치를 '0'으로 입력한다. 마지막으로 [Add 2 Constraint]버튼을 클릭하면 완료된다.



이렇게 constraint 를 적용하면 다양한 화면에서 어떻게 보여질지 한번 확인해보자

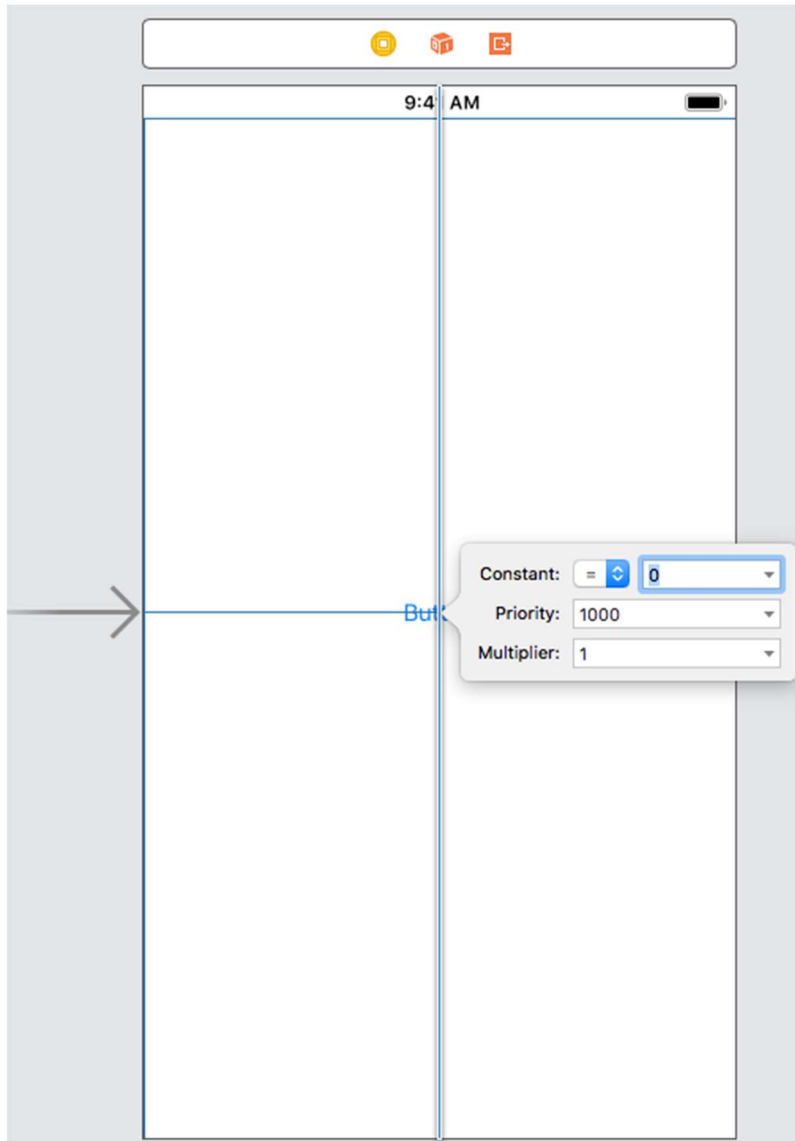


특정기기 화면을 기준으로 UI 를 확인 할 수 있다. 왼쪽 아래에 있는 <+> 버튼을 클릭하면 다른 기기를 추가할 수 있다.



이번에는 constraint 를 수정하는 방법을 살펴본다. 인터페이스 빌더 화면의 Document Outline 에서 뷰 구조를 살펴본다면 View 요소 하위에

[Constraints]라는 항목을 볼 수 있다. 수정할 때는 각 constraint 선을 더블클릭하면 수치를 조정하는 창이 나타나므로 간단하게 수정할 수 있다.



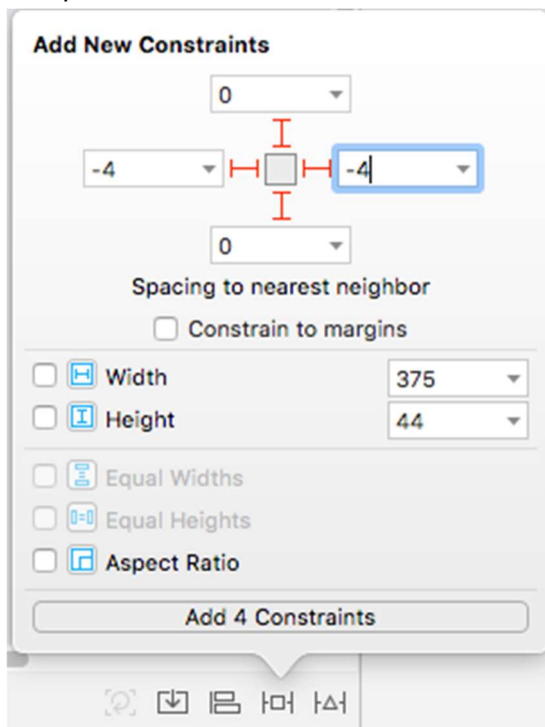
### 21-3. 로또 번호 생성기에 오토 레이아웃 적용하기

오토 레이아웃을 이용해 로또 번호 생성기 앱의 UI를 수정하는 과정은 각

UI 요소에 오토 레이아웃을 적용하는 과정이라고 생각하면 쉽다.

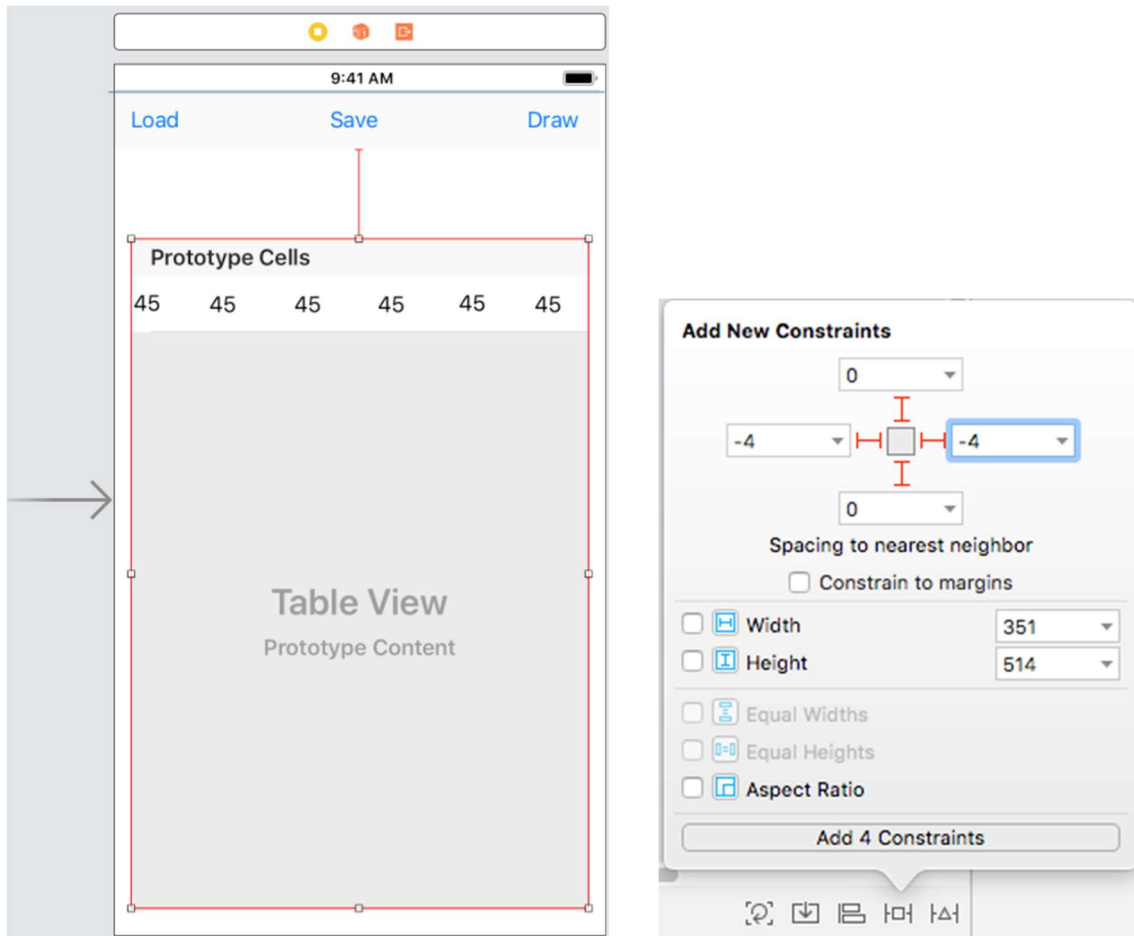
툴바부터 차례대로 constraint 를 추가하겠다. 기존에 작성 중이던 LottoDrawExample 프로젝트를 열고 Main.storyboard 파일을 선택해 인터페이스 빌더 화면으로 이동한다.

툴바를 선택하고 <pin>버튼을 클릭한 후 [Spacing to nearest neighbor]항목의 위와 아래에는 각각 '0'을 왼쪽과 오른쪽에는 각각 '-4'라는 설정 값을 입력해 constraint 를 추가한다. 최종 적용하면 <Add 4 Constraint>버튼을 클릭하고 <Resolve Auto Layout issues>버튼을 클릭한 후 [Selected View]->[Update Frames]을 선택한다.



이번에는 테이블 뷰에도 constraint 를 추가해 보자. [Table View]를 선택하고 <Pin>버튼을 클릭한 후 [Spacing to nearest neighbor]항목의 위와 아래에는 '0', 왼쪽과 오른쪽에는 '-4'라는 값을 설정해 constraint 를 모두추가한다. 그리고 <Add 4 Constraint>버튼을 클릭하고 <Resolve Auto Layout issues>버튼을 클릭한 후 [Select View] -> [Update Frames]을 선택한다.





새로 추가한 constraint 덕분에 UI가 잘 조정되었는지 확인해 보자.

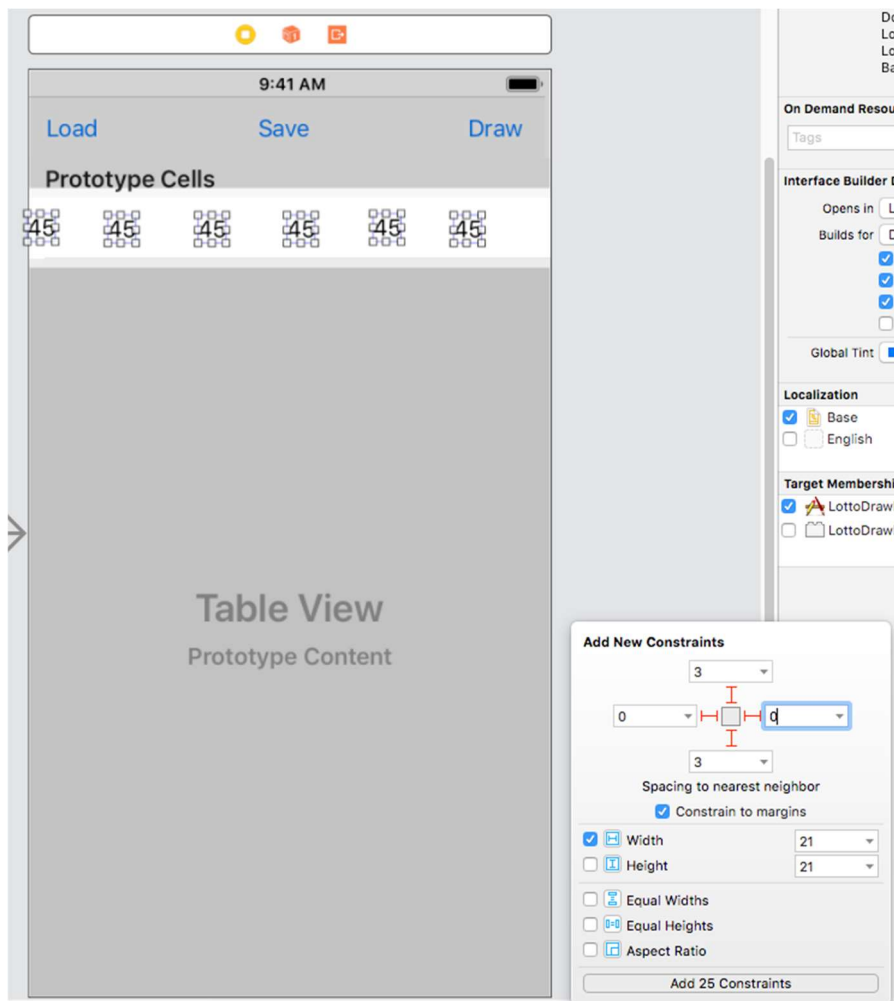
Simulator의 [Hardware] -> [Rotate Left/Right]를 선택해 화면을 회전해도 깔끔하게 배치된 UI를 확인할 수 있다.

#### 21-4. 버튼 정렬하기

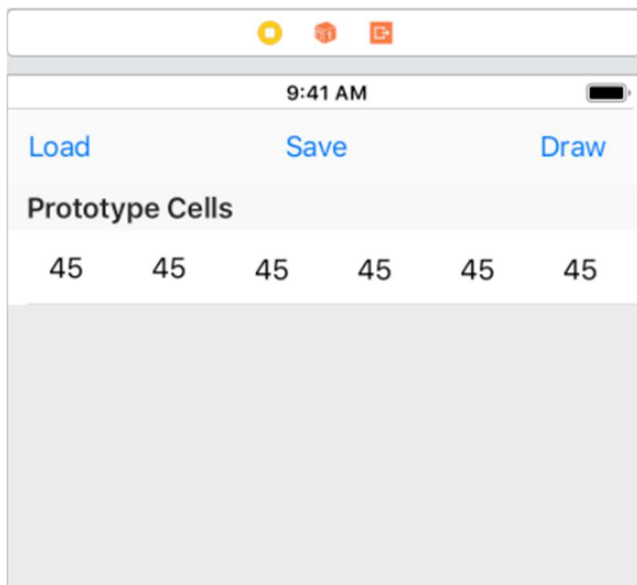
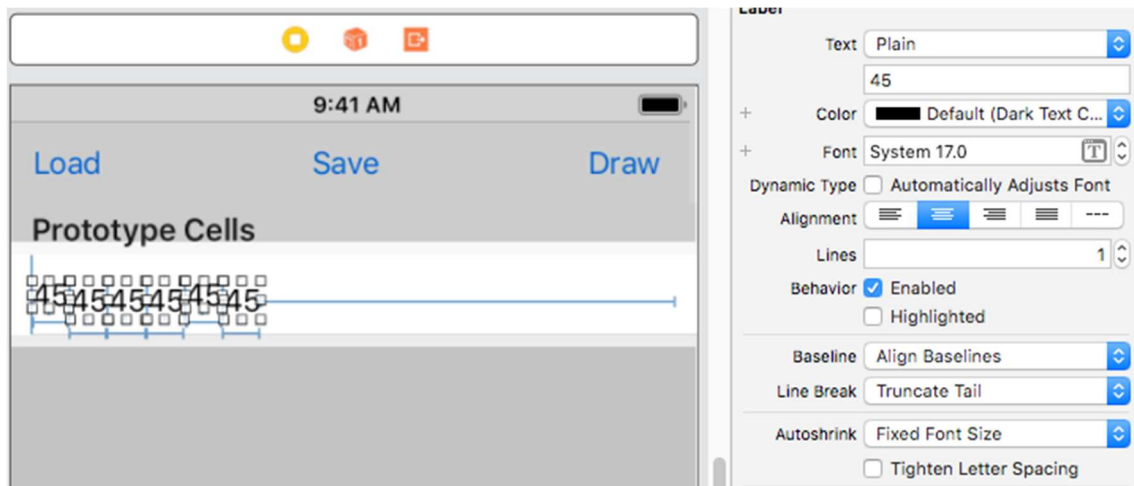
전체 UI 가 정돈된 것을 확인해 봤다, 그럼 Prototype Cell 에 있는 6 개의 [Label]도 정렬해 보자.

Prototype Cell 안에 있는 6 개의 [Label]을 한꺼번에 선택한다. <Pin> 버튼을 클릭한 후 [Spacing to nearest neighbor]항목의 위와 아래에는 '3', 오른쪽과 왼쪽은 '0'이라는 constraints 설정 값을 추가한다.

그리고 [Equal Widths] 항목을 체크를 켜고 [Update Frames]항목은 'All frames in Constrainer'를 선택한다.



보통 [Label]을 배치했을 때는 텍스트가 좌측 정렬되어 있을 것이다 이상태로는 간격을 맞추어 배치해도 어색할 수 있다 따라서 Attributes inspector의 Label -> Alignment 항목에서 중앙으로 정렬하도록 바꾼다.



이제 결과를 확인해 보자. 그런데 한줄만 나오면 반복적인 결과가 잘 나오는지 확인하기 힘들다. 따라서 ViewController.swift 파일에 있는 tableView(\_:numberOfRowsInSection:) 메서드 코드를 수정한다.

```

1 // 해당 섹션의 셀 개수 - 필수 메서드
2 func tableView(_ tableView: UITableView, numberOfRowsInSectionSection section: Int) ->
3 Int {
4     return 3
5 }

```

## 21-5. 로또번호 생성기

로또 번호를 추출하는 데는 복잡한 코드가 필요하지 않는다. 중복된 숫자가 나오지 않도록 하는 것이 핵심이다. 먼저 코드 구조를 살펴보고, 코드를 통해 번호를 추출하고 이를 앞에 보여주는 것 까지 만들어 보겠다.

### 1. 로또 번호를 생성하기 위한 코드 구조

로또는 1~45 사이의 숫자 중 중복 없이 6 개를 지정하는 게임이다.

1~45 까지의 임의의 수를 추출하려면 랜덤으로 숫자를 가져와야한다. 여기에 중복이 없는 숫자를 출력한다는 조건이 추가되는 것이다.

가장 먼저 생각나는 방법은 중복인지 아닌지 비교해보고 중복되는 숫자이면 새로운 숫자를 선택해 출력하는 것이다.

그런데 중복인지 아닌지를 비교해보는 방법은 출력해야 하는 숫자의 개수 많으면 비효율적이다.

실제로 로또 번호를 추첨하는 방송을 보면 미리 준비해둔 45 개의 공을 가지고 하나씩 뽑는 과정을 거친다는 것을 알 수 있다.

그러므로 미리 1~45 의 숫자가 들어 있는 배열에서 숫자를 하나씩 뽑는 것이다,

뽑힌 숫자는 배열에서 제거하면 중복될 일이 전혀 없다.

우선 원본이 될 배열부터 준비한다. 배열을 만들고 기본 범위의 값을 할당하면 된다.

```
var originalNumbers = Array(1...45)
```

다음으로 배열의 인덱스값을 랜덤으로 선택해서 해당 인덱스에 있는 숫자를 실제 테이블 뷰의 셀과 연결된 배열에 저장하고 원래 배열에서는 해당 숫자를 삭제해야 한다.

```
index =
```

```
    Int(arc4random_uniform(UInt32(originalNumbers.count)))
```

```
var columnArray = Array<Int>()
```

```
columnArray.append(originalNumbers[index])
```

```
originalNumbers.remove(at: index)
```

랜덤 숫자를 생성하는 여러가지 함수를 사용할 수 있지만 여기서는 `arc4random_uniform` 이라는 함수를 사용한다. 필요한 파라미터 `UInt32` 타입이어야 하고 생성되는 숫자를 대상으로 다시 타입 캐스팅 과정을 거치지만 한 줄 코드로 작성할 수 있다는 장점이 있다.

`columnArray` 는 추출된 값을 저장하는 배열이며 추출된 값은 `append` 메서드를 이용해서 추출한다.

그렇게 값을 추출하고 나면 원래 배열에서는 `removeAtIndex` 메서드를 이용해서 제거한다.

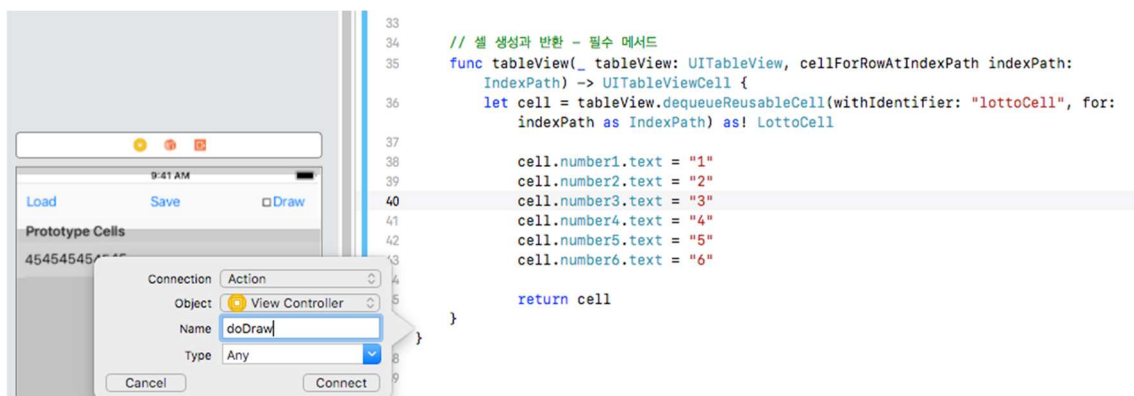
이렇게 6 회만 반복하면 중복 없는 6 개의 로또번호를 얻을 수 있다.

최종적으로 `columnArray` 에 있는 값을 테이블 뷰의 각 셀에 출력해주면 사람들은 앱 화면에서 추출된 값을 볼 수 있다.

## 2. 로또 번호 생성하기

Main.storyboard 파일, 오른쪽에는 ViewController.swift 파일이 우치하게 만든다.

그리고 툴바에 있는 <Draw>버튼을 마우스 오른쪽 버튼으로 클릭하면 나타나는 창에서 [Send Action] -> [selector]를 드래그 앤 드롭하여 ViewController의 마지막 중괄호 위에 연결한다.



생성된 `doDraw` 메서드를 정의한다. ViewController 클래스에 `lottoNumber` 라는 새로운 이중 배열 변수를 추가한다

1	<code>var lottoNumbers = Array&lt;Array&lt;Int&gt;&gt;()</code>
2	

```

3  @IBAction func doDraw(_ sender: UIBarButtonItem) {
4      lottoNumbers = Array<Array<Int>>()
5
6      var originalNumbers = Array(1...45)
7      var index = 0
8
9      for _ in 0...4 {
10         originalNumbers = Array(1...45)
11         var columnArray = Array<Int>()
12
13         for _ in 0...5 {
14             index = Int(arc4random_uniform(UInt32(originalNumbers.count)))
15             columnArray.append(originalNumbers[index])
16             originalNumbers.remove(at: index)
17         }
18
19         columnArray.sort(by: { $0 < $1 })
20         lottoNumbers.append(columnArray)
21     }
22
23 }
24
25

```

9~13 행

20

21

만들어 놓은 로또 번호를 테이블 뷰에 깔끔하게 나타내보자. 2 개의 메서드 tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int)와 func tableView(\_ tableView: UITableView, cellForRowAtIndexPath indexPath: IndexPath)메서드를 수정해 보자.

```

1  // 해당 섹션의 셀 개수 - 필수 메서드
2      func tableView(_ tableView: UITableView, numberOfRowsInSection section:
3  Int) -> Int {
4      return lottoNumbers.count
5  }
6
7  // 셀 생성과 반환 - 필수 메서드
8      func tableView(_ tableView: UITableView, cellForRowAtIndexPath indexPath:
9  IndexPath) -> UITableViewCell {
10         let cell = tableView.dequeueReusableCell(
11             withIdentifier: "lottoCell",
12             for: indexPath as IndexPath) as! LottoCell

```

```

13
14         let row:Int = indexPath.row
15
16         cell.number1.text = "\(lottoNumbers[row][0])"
17         cell.number2.text = "\(lottoNumbers[row][1])"
18         cell.number3.text = "\(lottoNumbers[row][2])"
19         cell.number4.text = "\(lottoNumbers[row][3])"
20         cell.number5.text = "\(lottoNumbers[row][4])"
21         cell.number6.text = "\(lottoNumbers[row][5])"
22
23         return cell
24     }
25
26

```

4

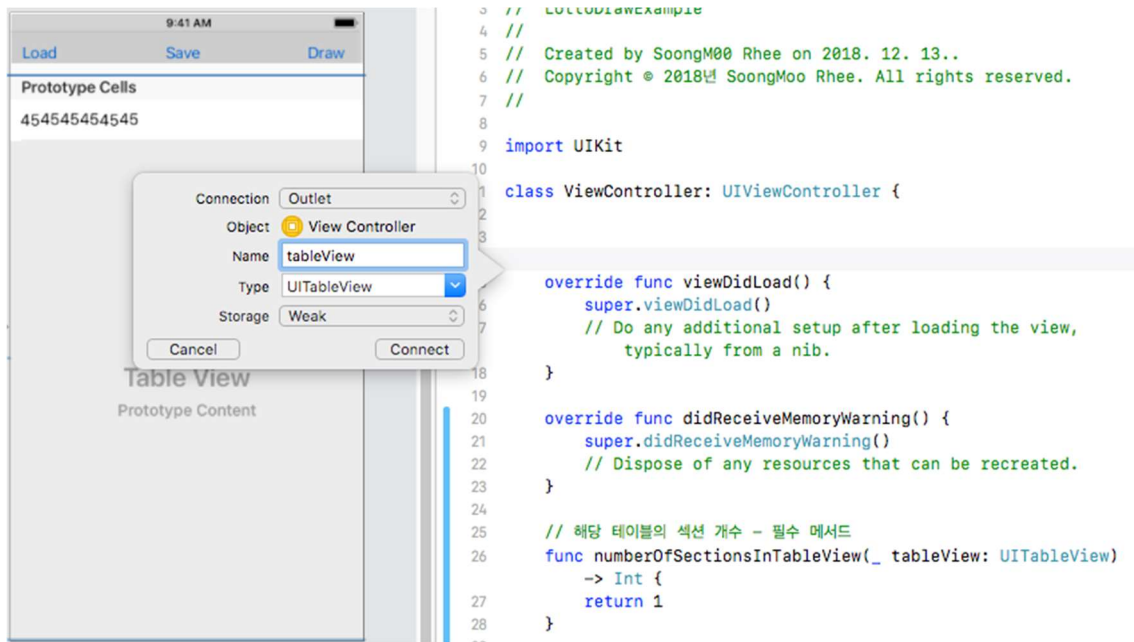
16

18~23

마지막으로 <Draw>버튼을 터치할 때마다 해당 값을 테이블 뷰에 적용하기 위해 reloadData 메서드를 호출해야 한다, 먼저 해당 메서드를 호출하기 위해 테이블 뷰의 아웃렛을 만들어준다 Assistant Editor 를 선택하고 왼쪽에는 Main.storyboard 파일, 오른쪽에는 ViewController.swift 파일이 위치하게 만든다.

그리고 뷰 구조를 나타내는 Document Outline 에서 [Table View]항목을 마우스 오른쪽버튼을 클릭해 선택한 후 ViewController.swift 파일의 Class ViewController: UIViewController { 아래로 드래그 앤 드롭해서 아웃렛을 연결한다.

아웃렛 연결 창에서는 [Name] 항목에 'tableView'를 입력하고 [Type]항목은 'UITableView'가 선택하고 <Connect>버튼을 클릭한다.



```

1 class ViewController: UIViewController {
2
3
4     @IBOutlet weak var tableView: UITableView!
5     override func viewDidLoad() {
6         super.viewDidLoad()
7         // Do any additional setup after loading the view, typically from a nib.
8     }
9     ...
10 }
11

```

마지막으로 didDraw 메서드의 가장 아랫부분에서 reloadData 메서드를 호출한다.

```

1 @IBAction func doDraw(_ sender: UIBarButtonItem) {
2
3     tableView.reloadData();
4 }

```

결과 확인



Load		Save		Draw	
1	7	10	11	21	26
19	24	35	37	43	44
3	5	9	16	37	40
6	9	28	35	36	41
25	29	34	35	39	41

## 21-6. 생성된 로또 번호 저장하기

<Draw>버튼을 터치했을 때 번호가 잘 나타나는지 확인했다.

그렇다면 이제 생성한 번호를 저장하고 불러오는 과정이다.

iOS 앱을 만들면서 데이터를 저장하는 방법은 여러가지 있다. 파일을 생성해서 그 안에 저장하는 방법, UserDefaults 클래스를 이용해 설정 값을 저장하는 방법, 별도의 데이터베이스에 저장하는 방법 등이 있다.

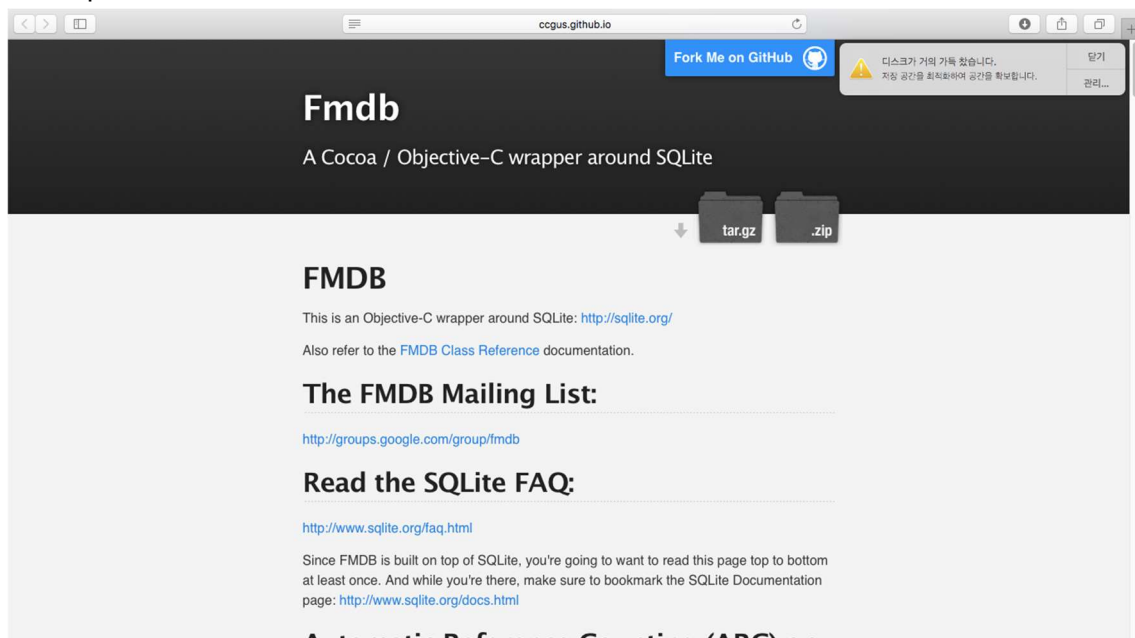
### 1. SQLite 데이터베이스 다루기

SQLite 데이터베이스에 로또를 저장하기에 앞서 간단한 데이터 베이스 다루기 예제를 진행해 보자,

모바일 앱에서는 SQLite 라는 데이터베이스를 많이 사용한다. 안드로이드는 기본으로 사용할 수 있도록 되어 있고 iOS 도 기본적으로 사용할 수 있지만 별도의 클래스 파일을 만들어 사용하는 것이 더 편리하다.

이러한 별도의 클래스 파일은 직접 만들 수도 있지만 기존에 공개된 유명한 라이브러리를 사용할 수도 있다. 새로 만드는 것보다는 SQLite 기반으로 이미 만들어져 있는 FMDB 라는 클래스 파일을 다운로드하자.

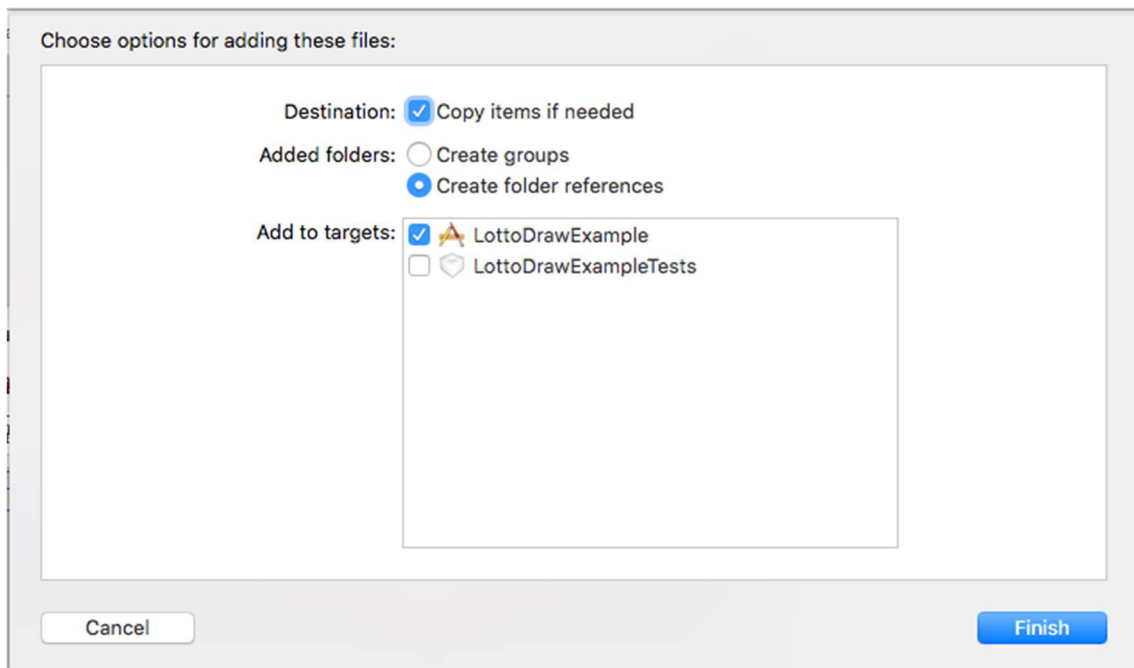
<http://ccgus.github.io/fmdb/> 라는 웹사이트에 접속해 화면 위 오른쪽에 있는 .zip 링크를 클릭한 후 FMDB 압축파일을 다운로드한다.



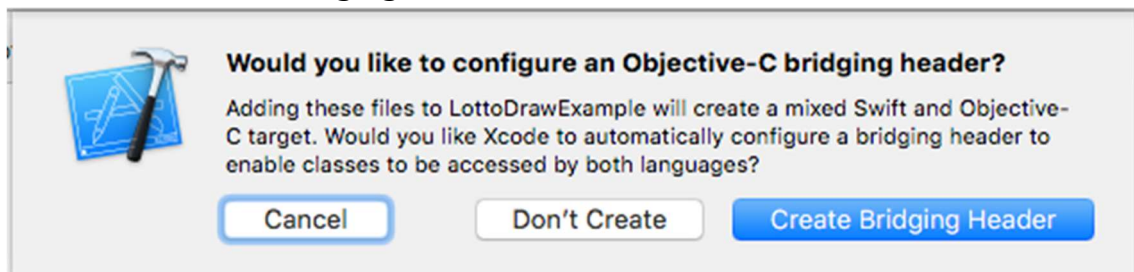
다운로드한 압축 파일의 압축을 해제하면 다양한 파일들을 볼 수 있다.

그 중 src/fmdb 폴더에 있는 \*.h 파일과 \*.m 파일 전체를 [프로젝트 이름] 폴더로 드래그 앤 드롭해 복사를 해준다.

해당 폴더를 추가하는 것과 관련된 옵션창이 나타난다. [Destination]항목의 [Copy item if needed]에 체크를 켜고 나머지는 기본 설정 그대로 두고 <Finish>를 클릭한다.



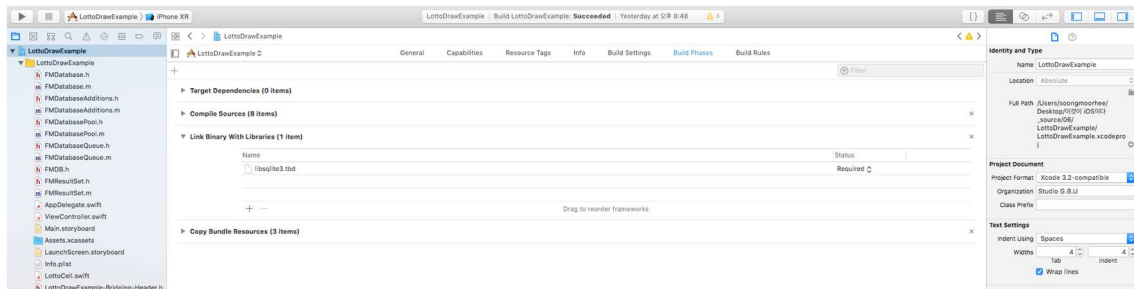
추가한 FMDB 관련 폴더는 Objective-C 로 만들어져 있다. 우리가 만드는 프로젝트는 Swift 로 만들기 때문에 2 개의 언어를 섞어서 사용하려면 중간다리가 필요하다. 이때 사용하는 것이 Objective-C Bridging Header 이다. 그러므로 <Create Bridging Header> 를 클릭한다.



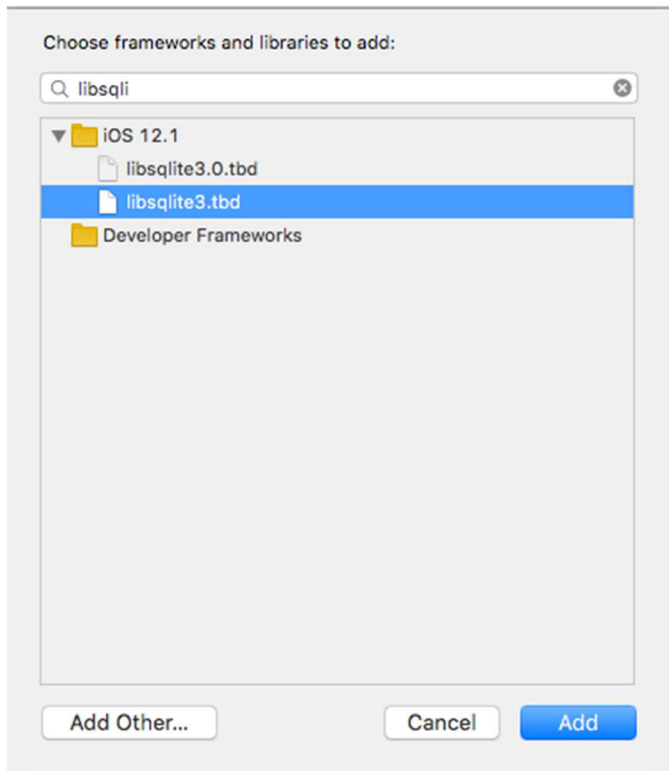
LottoDrawExample-Bridging-Header.h 파일이 추가된 것을 확인할 수 있다.

#import "FMDB.h"
------------------

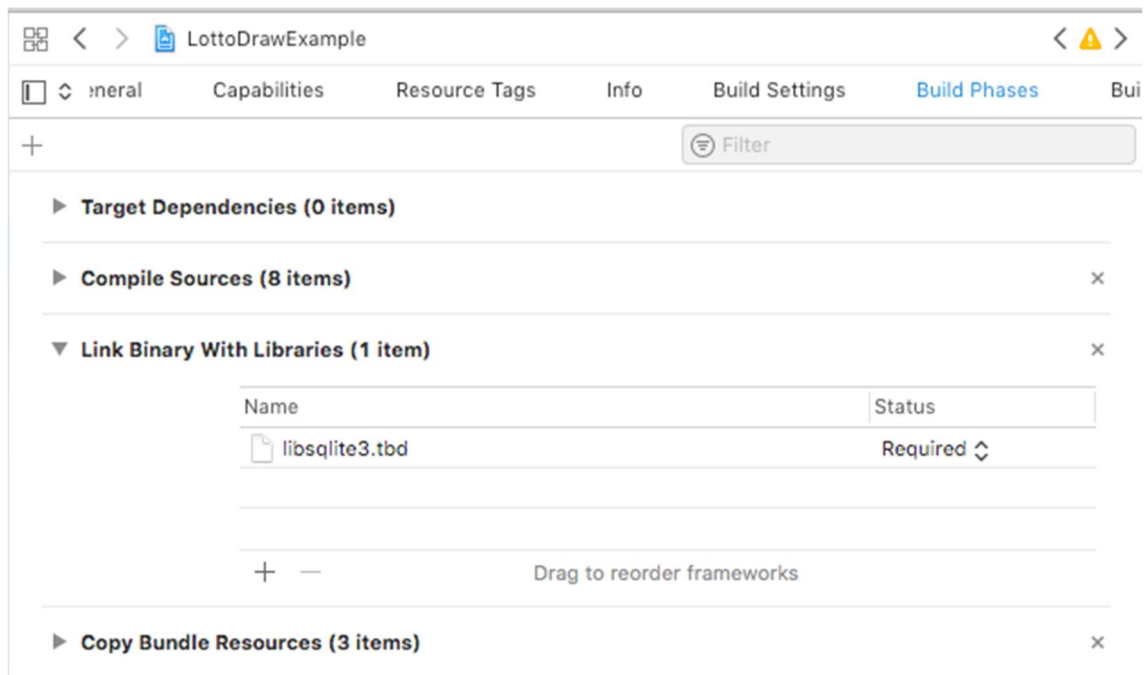
FMDB 클래스 파일들을 사요할 수 있게 설정을 마쳤지만 실제 SQLite 를 사용하기 위한 필수 라이브러리를 추가해야 한다, Project navigator 에서 [프로젝트 이름]을 선택하고 나타난 프로젝트 정보에서 [Build Phases]를 선택한다.



Choose framework and libraries to add 창 위에 있는 검색 창에 'libsqli'라고 입력하면 관련 라이브러리들이 나타난다. 이때 libsqlite3.tbd 를 선택하고 <Add> 버튼을 클릭한다.



해당 라이브러리가 추가된 것을 확인할 수 있다. 다양한 프로젝트에서 라이브러리나 프레임워크를 추가할 때는 이 기능을 이용하면 된다.



Main.storyboard 로 이동해서 <Load> 와 <Save>버튼에 액션을 연결한다, 위치는 ViewController.swift 파일의 viewController 클래스를 닫는 중괄호 위이고 <Load>버튼은 loadData 메서드, <Save>버튼은 saveData 메서드로 연결한다.

```
@IBAction func loadData(_ sender: Any) {
}
@IBAction func saveData(_ sender: Any) {
}
```

ViewController.swift 파일에 databasePath 변수를 추가하고 viewDidLoad()메서드를 수정한다.

```
var databasePath = String()

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.

    let fileMgr = FileManager.default
    let dirPaths = NSSearchPathForDirectoriesInDomains(
        .documentDirectory, .userDomainMask, true)
    let docsDir = dirPaths[0]
    databasePath = docsDir + "/lotto.db"
```

```

if !fileMgr.fileExists(atPath: databasePath as String) {
    let db = FMDatabase(path:databasePath as String)

    if db == nil {
        NSLog("DB 생성 오류")
    }

    if ((db.open()) != nil) {
        let sql_statement = "Create table if not exists lotto(id integer
primary key autoincrement, number1 integer, number2 integer, number3 integer,
number4 integer, number5 integer, number6 integer)"

        if !(db.executeStatements(sql_statement)) {
            NSLog("테이블 생성 오류")
        }

        db.close()
    } else {
        NSLog("디비 연결 오류")
    }
}

```

21-7. saveData 와 loadData 메서드를 구현한다.

```

@IBAction func doSave(_ sender: UIBarButtonItem) {
    let db = FMDatabase(path:databasePath as String)

    if ((db.open()) != nil) {
        do{
            try db.executeUpdate("delete from lotto", values: [])

            if ((db.hadError()) != nil) {
                NSLog("디비 초기화 오류")
            }

            for numbers in lottoNumbers {
                let insertQuery = "insert into lotto(number1,
number2, number3, number4, number5, number6) values (?, ?, ?, ?, ?, ?)"

```

```

        try db.executeUpdate(insertQuery, values:
["₩(numbers[0])", "₩(numbers[1])", "₩(numbers[2])", "₩(numbers[3])",
"₩(numbers[4])", "₩(numbers[5])"])

        if ((db.hadError()) != nil) {
            NSLog("저장 오류 ₩(insertQuery)")
        } else {
            NSLog("저장 성공 ₩(insertQuery)")
        }
    }
} catch let error as NSError {
    print("failed: ₩(error.localizedDescription)")
}
} else {
    NSLog("디비 연결 오류")
}
}

@IBAction func doLoad(_ sender: UIBarButtonItem) {
    lottoNumbers = Array<Array<Int>>>()
    let db = FMDatabase(path:databasePath as String)

    if ((db.open()) != nil) {
        let selectQuery = "select number1, number2, number3,
number4, number5, number6 from lotto"
        do{
            let result:FMResultSet? = try
db.executeQuery(selectQuery, values:[])

            if result != nil{
                while result!.next() {

```

```
        var columnArray = Array<Int>()

        columnArray.append(Int(result!.string(forColumn: "number1")!))

        columnArray.append(Int(result!.string(forColumn: "number2")!))

        columnArray.append(Int(result!.string(forColumn: "number3")!))

        columnArray.append(Int(result!.string(forColumn: "number4")!))

        columnArray.append(Int(result!.string(forColumn: "number5")!))

        columnArray.append(Int(result!.string(forColumn: "number6")!))

        lottoNumbers.append(columnArray)
        NSLog("자료 불러오기 성공")
    }
    } else {
        NSLog("자료 불러오기 실패")
    }
} catch let error as NSError {
    print("failed: ₩(error.localizedDescription)")
}

tableView.reloadData();
    }
}
```