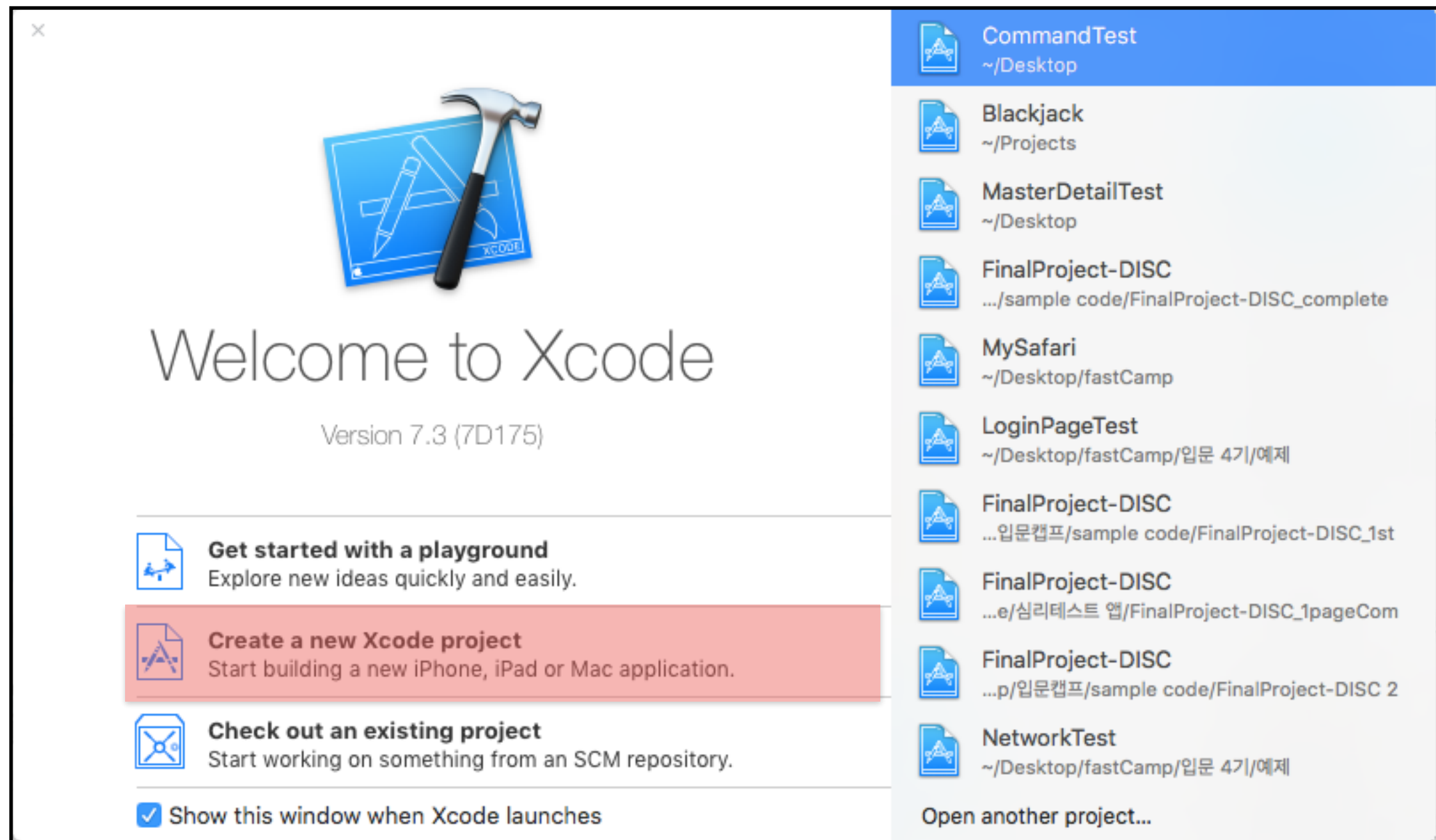
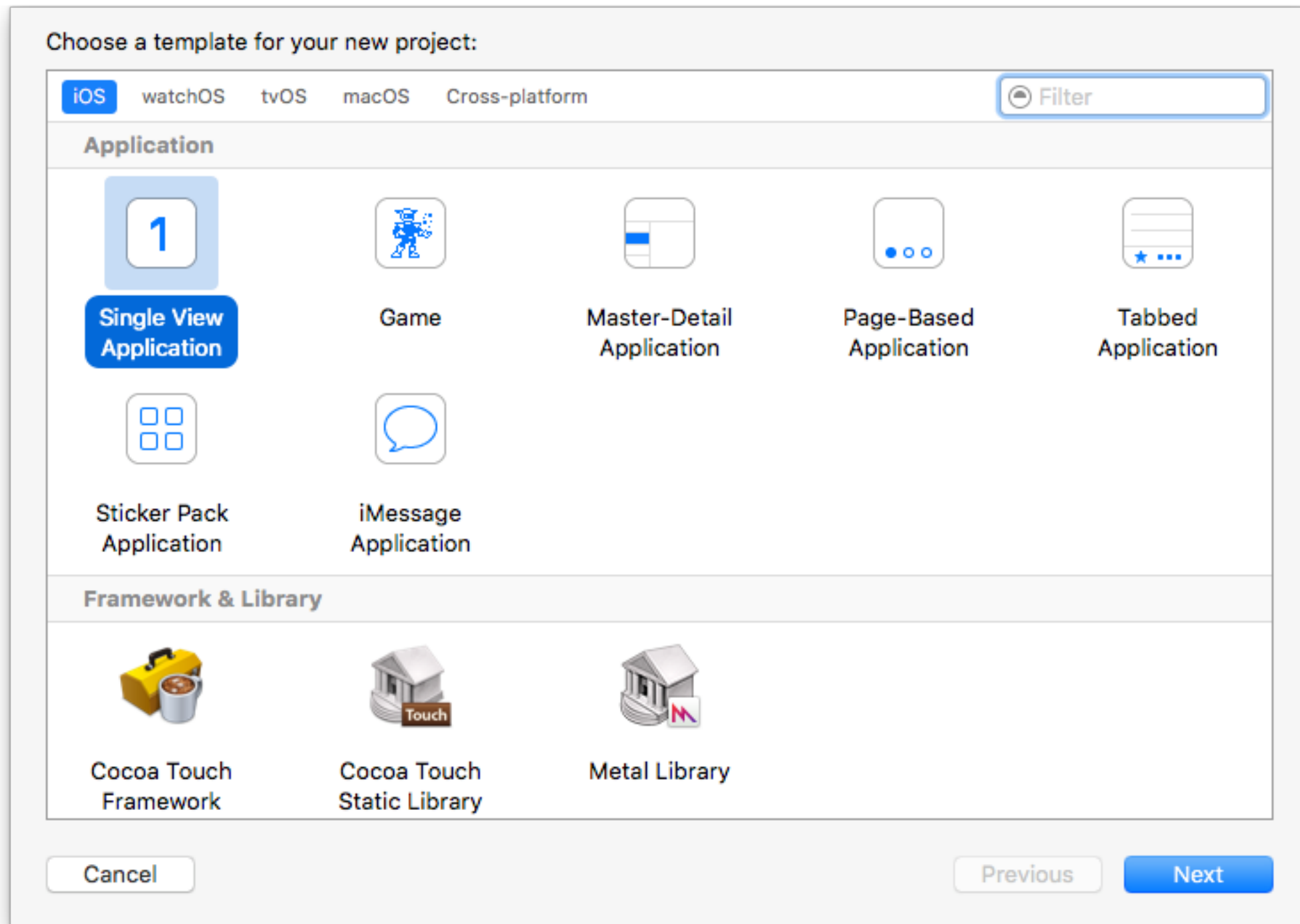

Xcode 사용법

Xcode - 시작



템플릿 선택



프로젝트 생성

Choose options for your new project:


Product Name:


Team:

Organization Name:

Organization Identifi...

Bundle Identifier:

Language: 

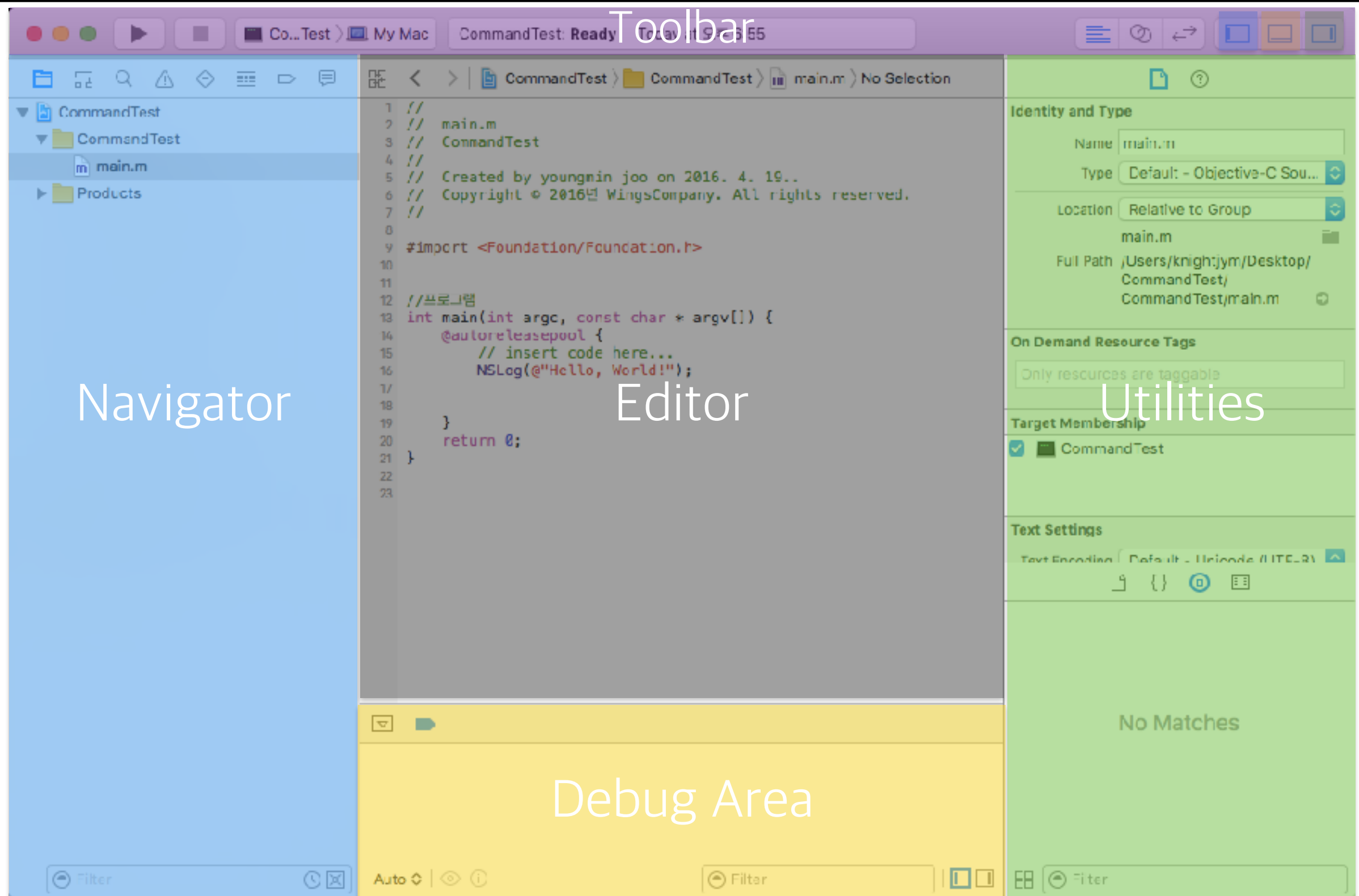
Devices: 

☐ Use Core Data

☐ Include Unit Tests

☐ Include UI Tests

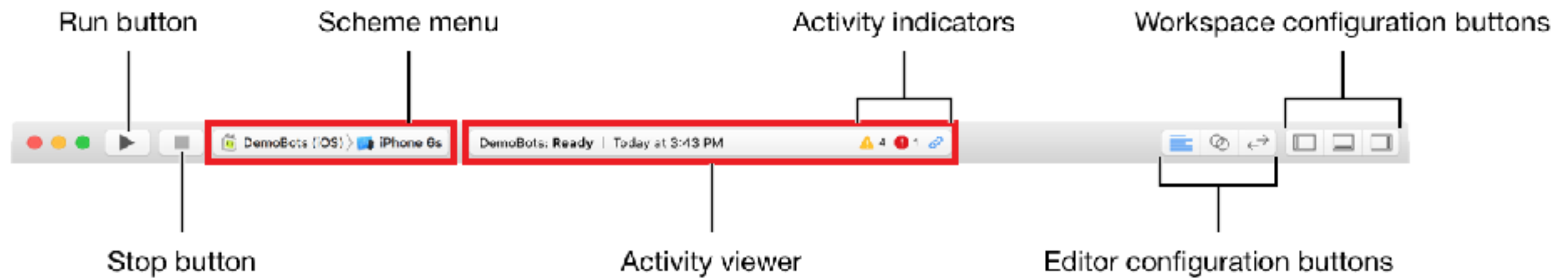
Xcode Main Window



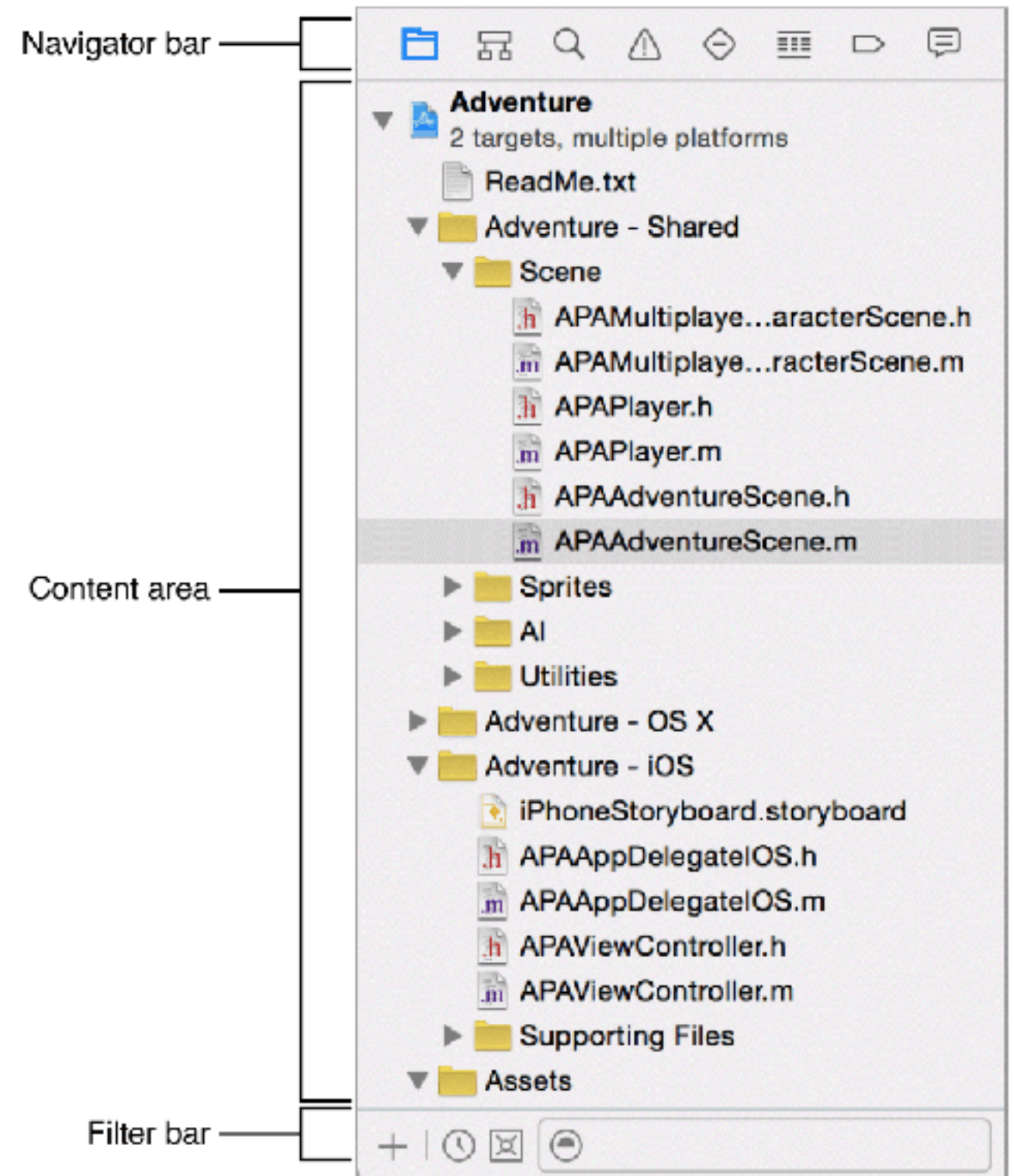
Xcode Main Window

- Navigator : 프로젝트 관리를 위한 도구 모음
- Editor : Project Navigator에서 선택한 파일의 내용을 수정하는 화면
- Debug Area: 프로그램 실행 중 Debugging를 위한 콘솔창
- Utilities : Project Navigator에서 선택된 파일의 상세 정보 및 UI속성 수정등의 작업을 위한 공간

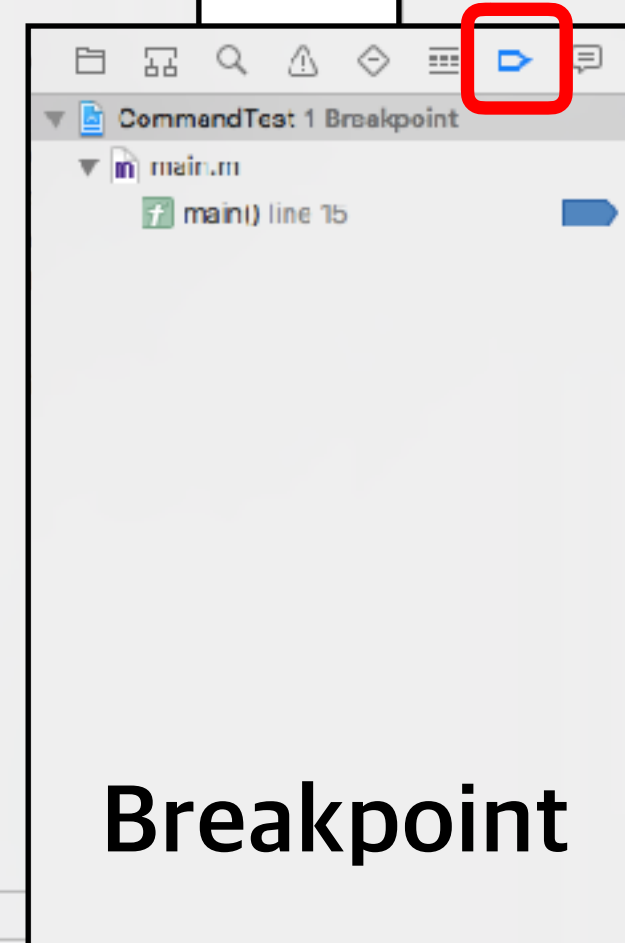
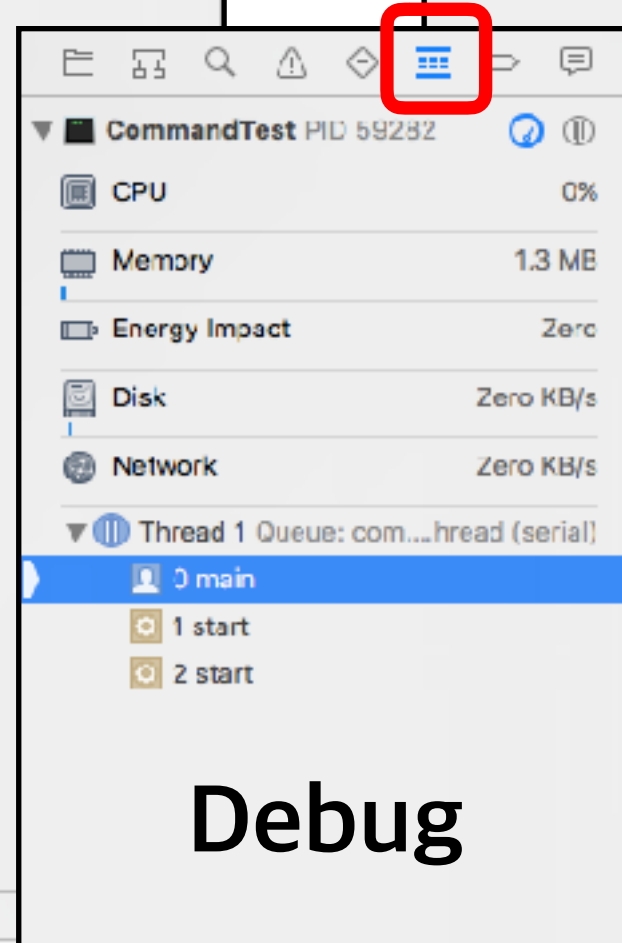
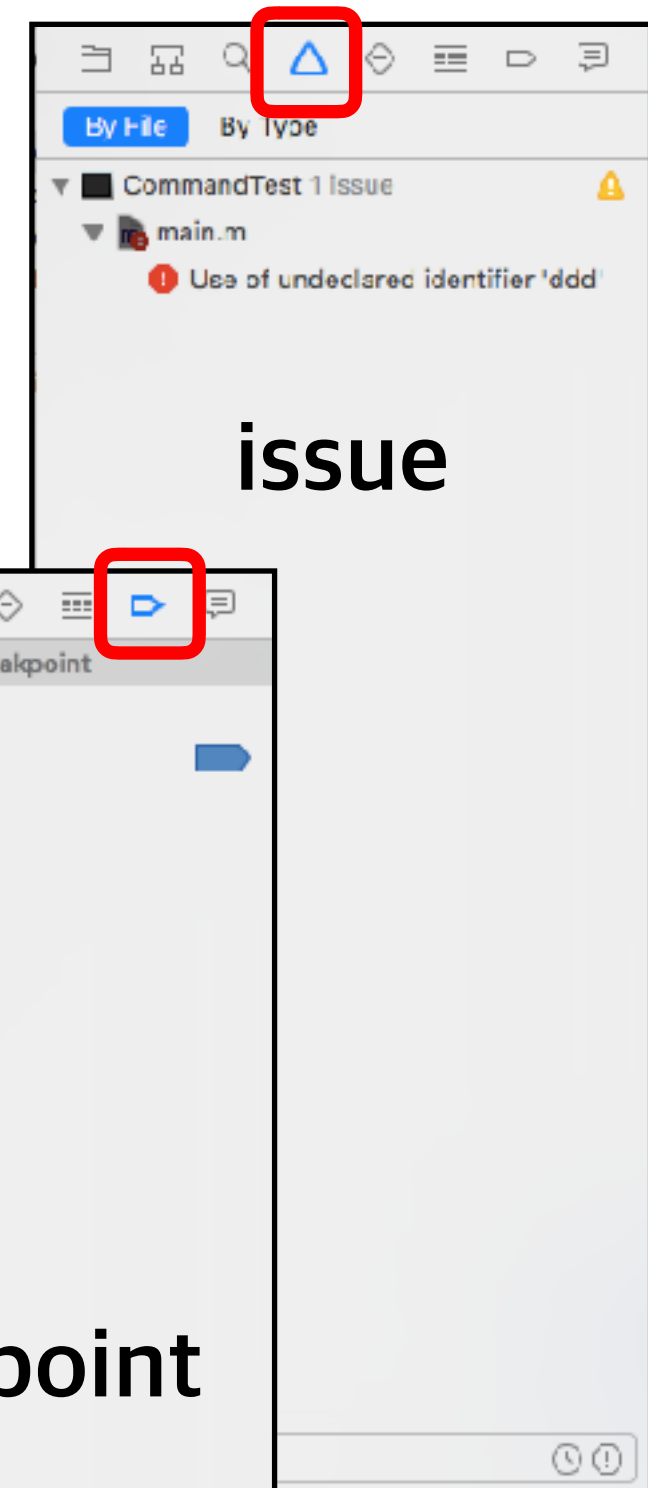
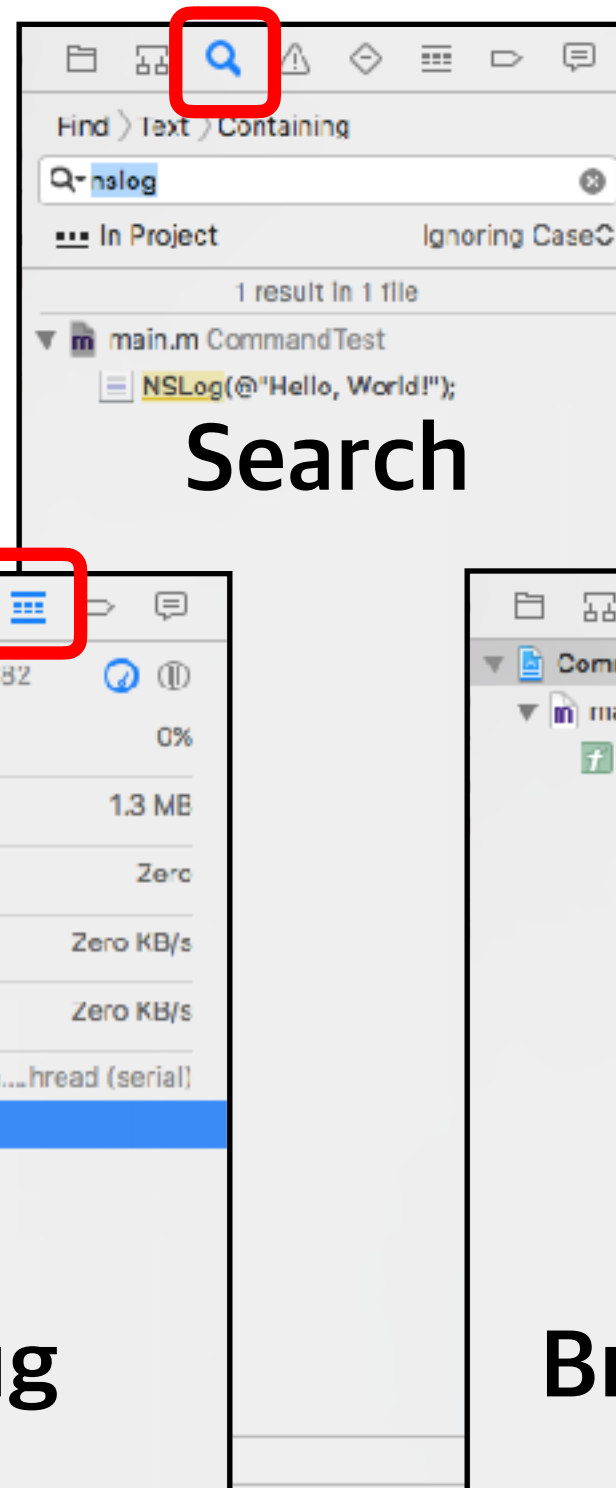
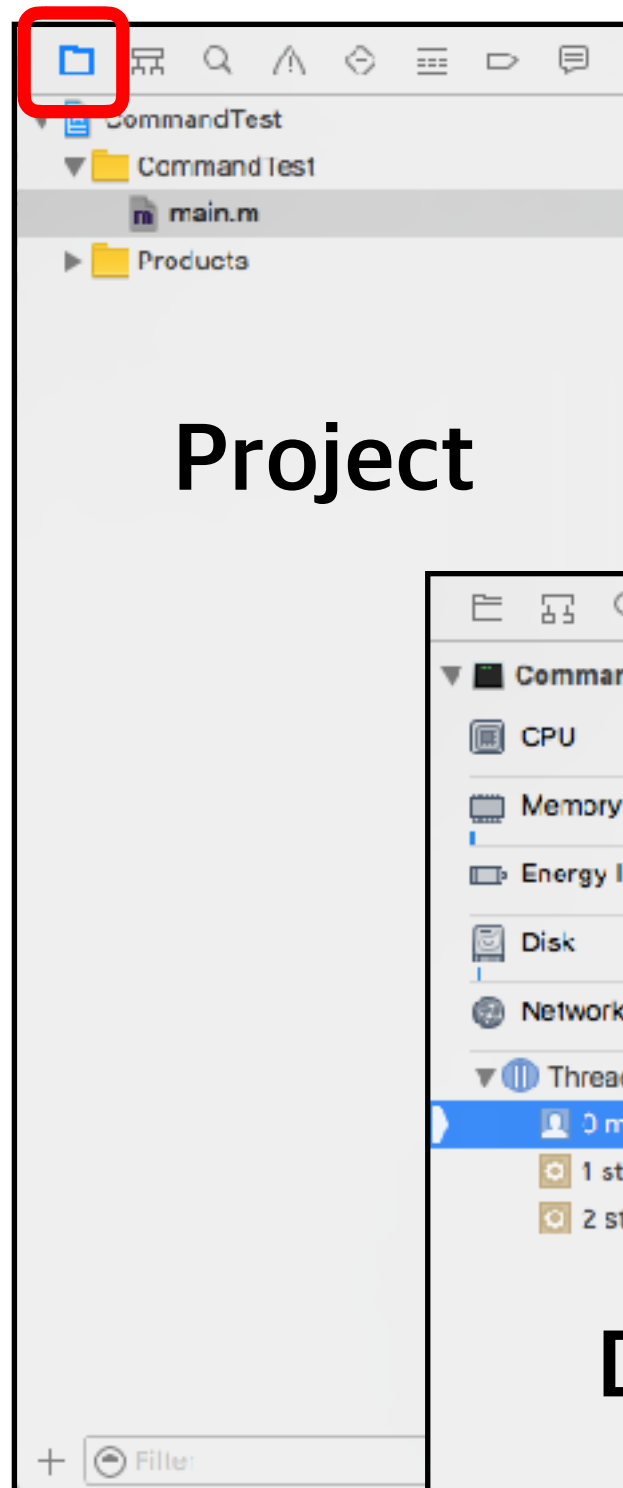
Toolbar



Navigator Area

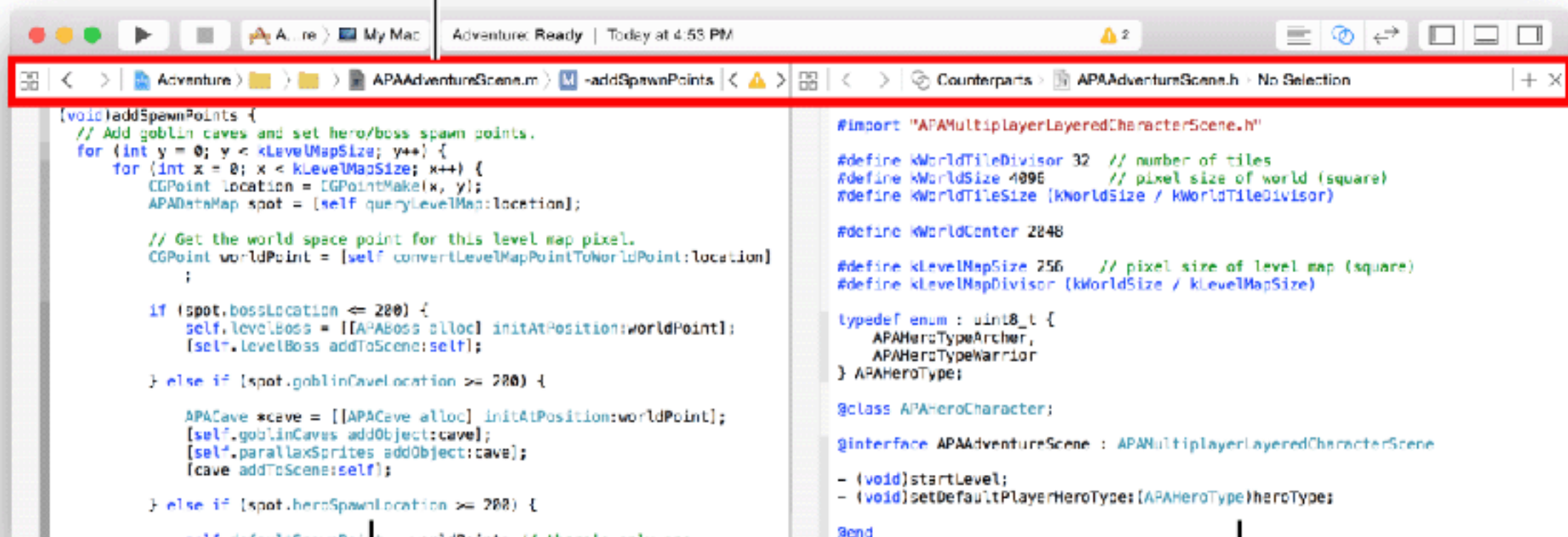


Navigator bar Menu



Editor




Jump bars



Standard editor pane

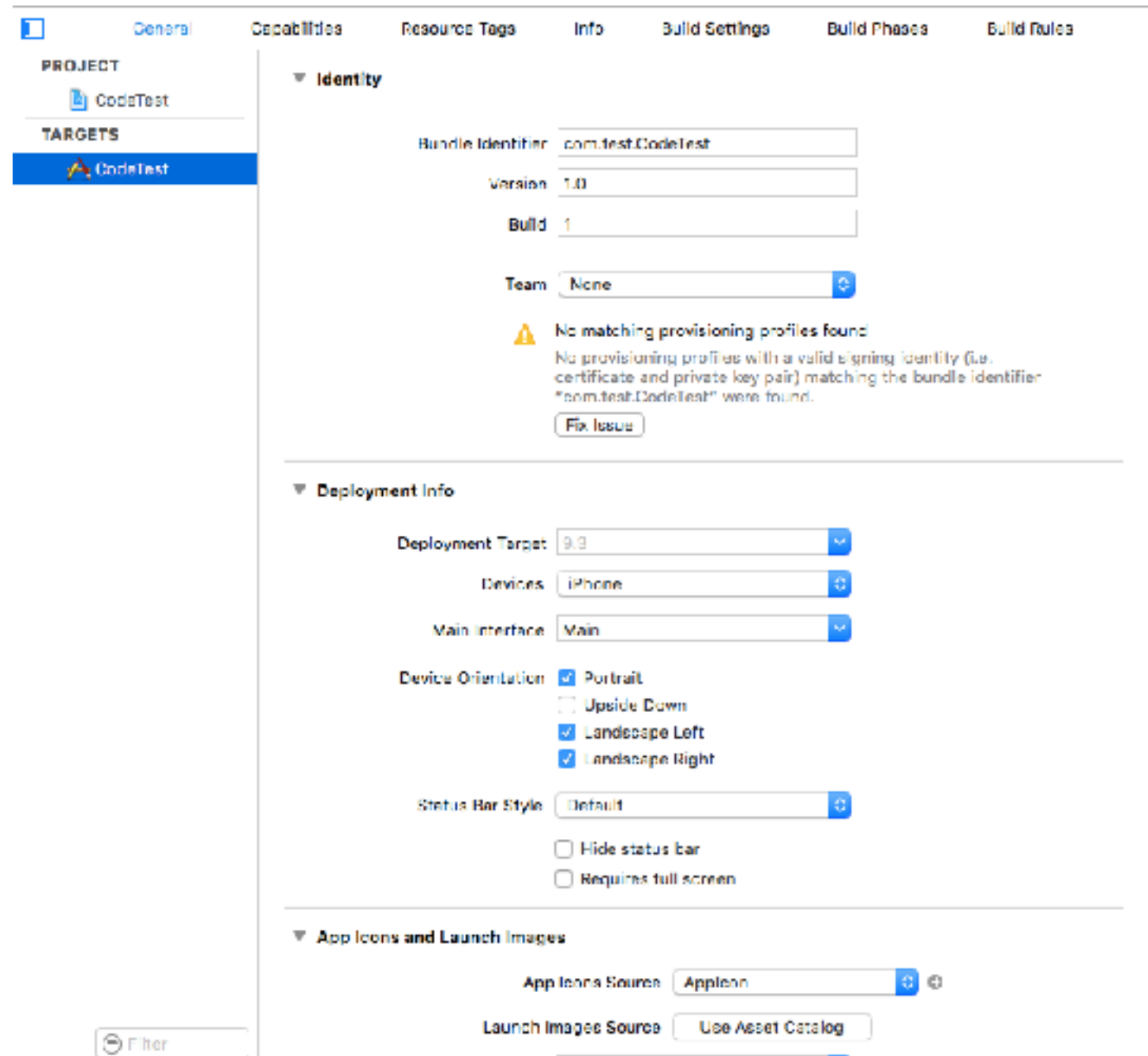
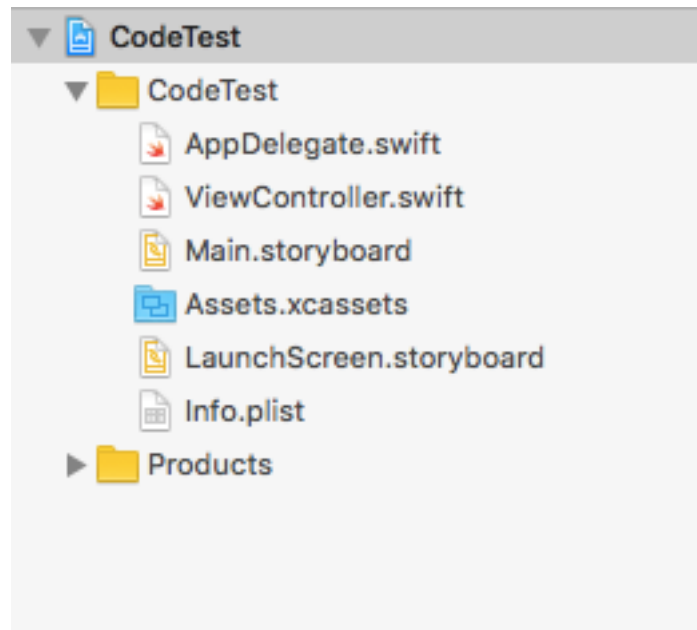
Assistant editor pane

Editor 상태편집

- Standard(): fills a single editor pane with the contents of the selected file.
- Assistant(): presents a separate editor pane with content logically related to that in the standard editor pane. Use the split controls in the
- Version (): shows the differences between the selected file in one pane and another version of that same file in a second pane.

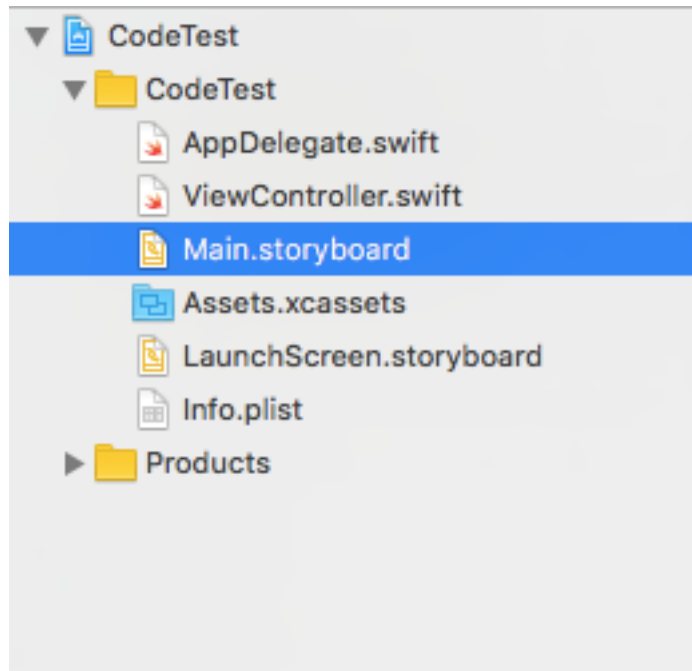
Project Editor

- 프로젝트 설정 변경



Source Editor

- 선택된 파일의 코드를 수정할 수 있다.



```
1 //
2 // ViewController.swift
3 // CodeTest
4 //
5 // Created by youngmin joo on 2016. 5. 3..
6 // Copyright © 2016년 WingsCompany. All rights reserved.
7 //
8
9 import UIKit
10
11 class ViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15         // Do any additional setup after loading the view, typically from a nib.
16     }
17
18     override func didReceiveMemoryWarning() {
19         super.didReceiveMemoryWarning()
20         // Dispose of any resources that can be recreated.
21     }
22
23 }
24
25
26
```

Break point

- debug를 위한 방법
- 여기서 실행을 멈춰라!

```
28 //프로그램
29 int main(int argc, const char * argv[]) {
30     @autoreleasepool {
31         // insert code here...
32
33         printf("여기에서 브레이크 포인트가 실행된다.");
34
35         printf("이코드는 아직 실행되지 않습니다.");
36
37         printf("다음 스텝을 눌러야 실행됩니다.");
38
39     }
40     return 0;
41 }
42
43
44
45
46
```

Thread 1: breakpoint 1.1

CommandTest > Thread 1 > 0 main

▸ A argv = (const char **) 0x7fff5fbff808
A argc = (int) 1

여기에서 브레이크 포인트가 실행된다. (lldb)

Break point

The screenshot shows a C program in a debugger. The code is as follows:

```
28 //프로그램
29 int main(int argc, const char * argv[]) {
30     @autoreleasepool {
31         // insert code here...
32
33         printf("여기에서 브레이크 포인트가 실행된다.");
34
35         printf("이코드는 아직 실행되지 않습니다.");
36
37         printf("이코드는 실행되었습니다.");
38
39     }
40
41     return 0;
42 }
43
44
45
46
```

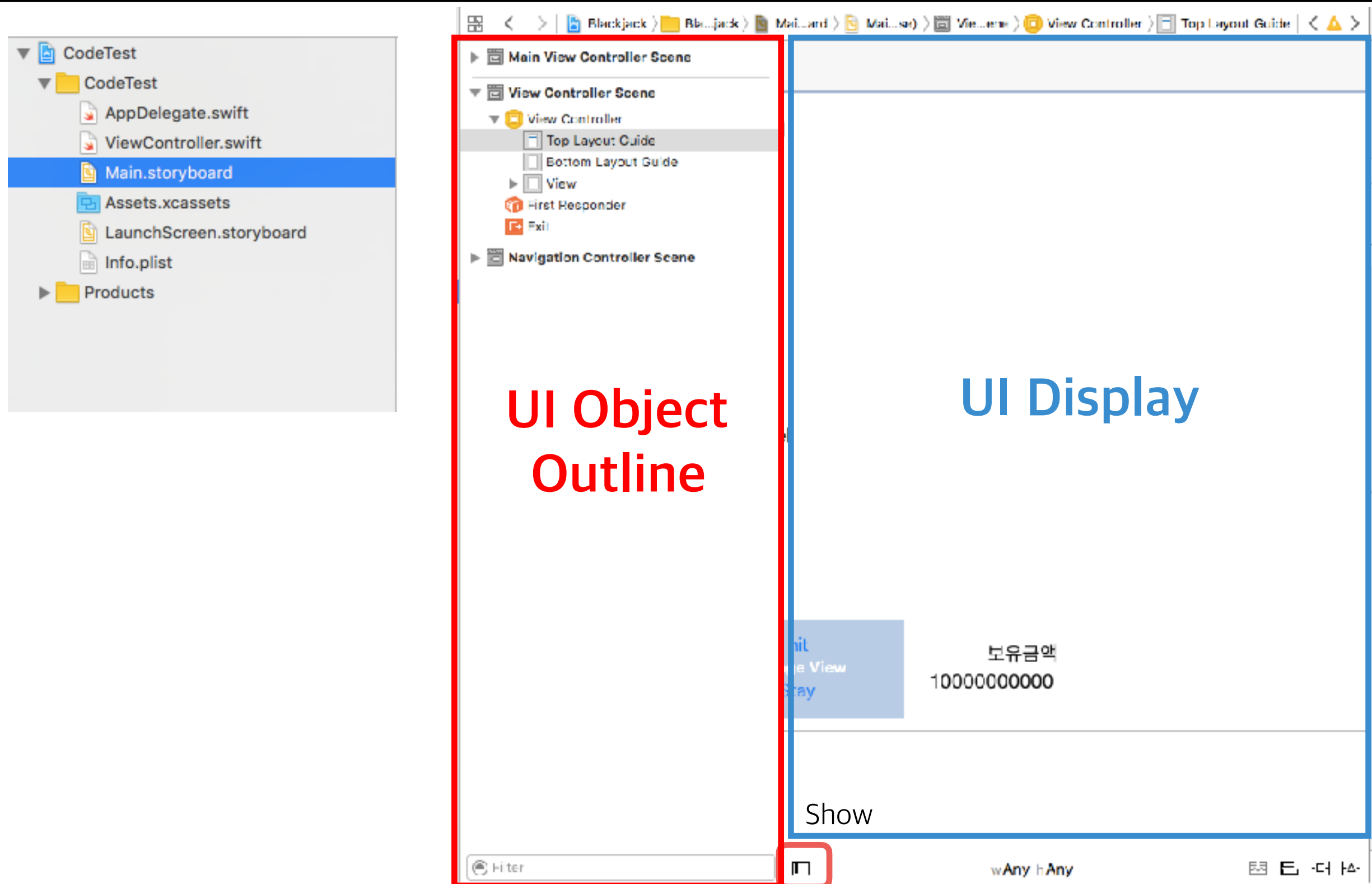
A breakpoint is set at line 35, indicated by a green bar and the text "Thread 1: breakpoint 1.1".

Red text labels with arrows point to the debugger's toolbar:

- break point enable** points to the blue play button icon.
- continue** points to the blue square icon with a right-pointing triangle.
- next Step** points to the blue square icon with a right-pointing triangle and a small upward arrow.

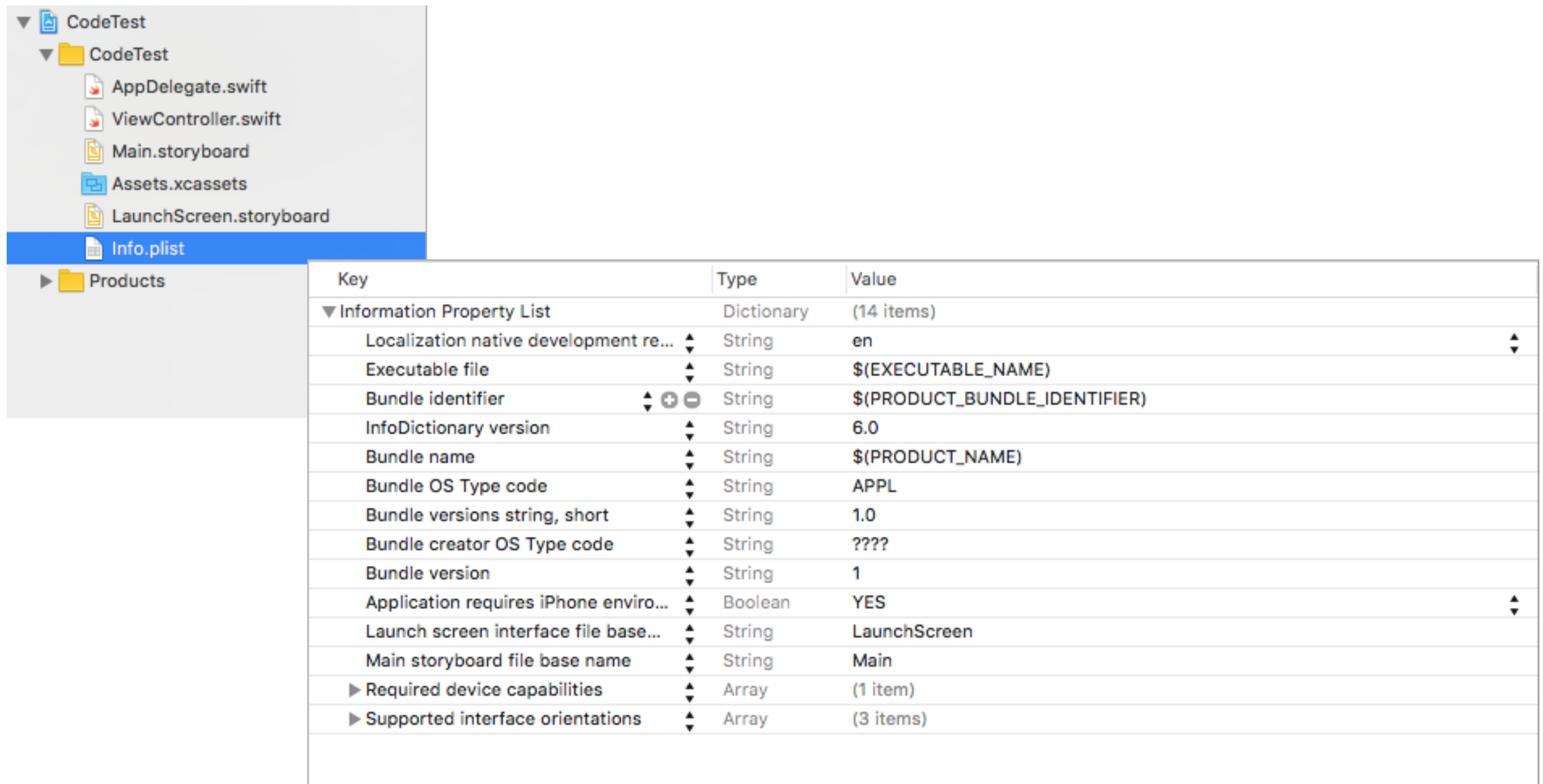
The debugger's status bar shows "CommandTest > Thread 1 > 0 main". The console output shows the message "여기에서 브레이크 포인트가 실행된다. (lldb)".

Interface Builder



Property list Editor

- property list(plist)파일 편집

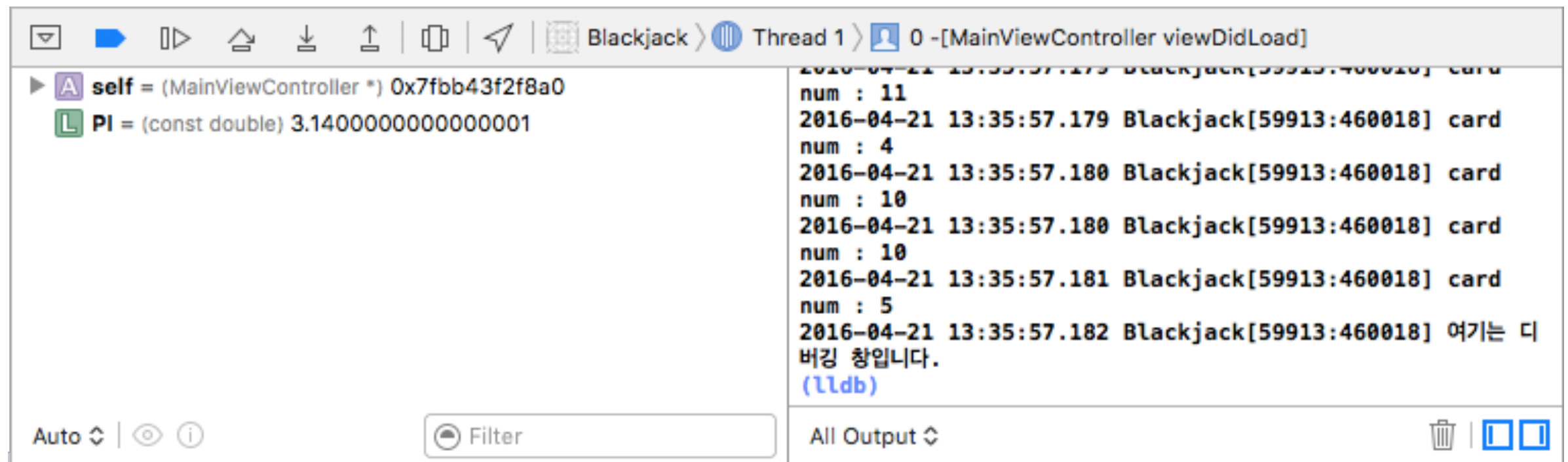


Key	Type	Value
▼ Information Property List	Dictionary	(14 items)
Localization native development re...	String	en
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)

Debug Area

Variables View

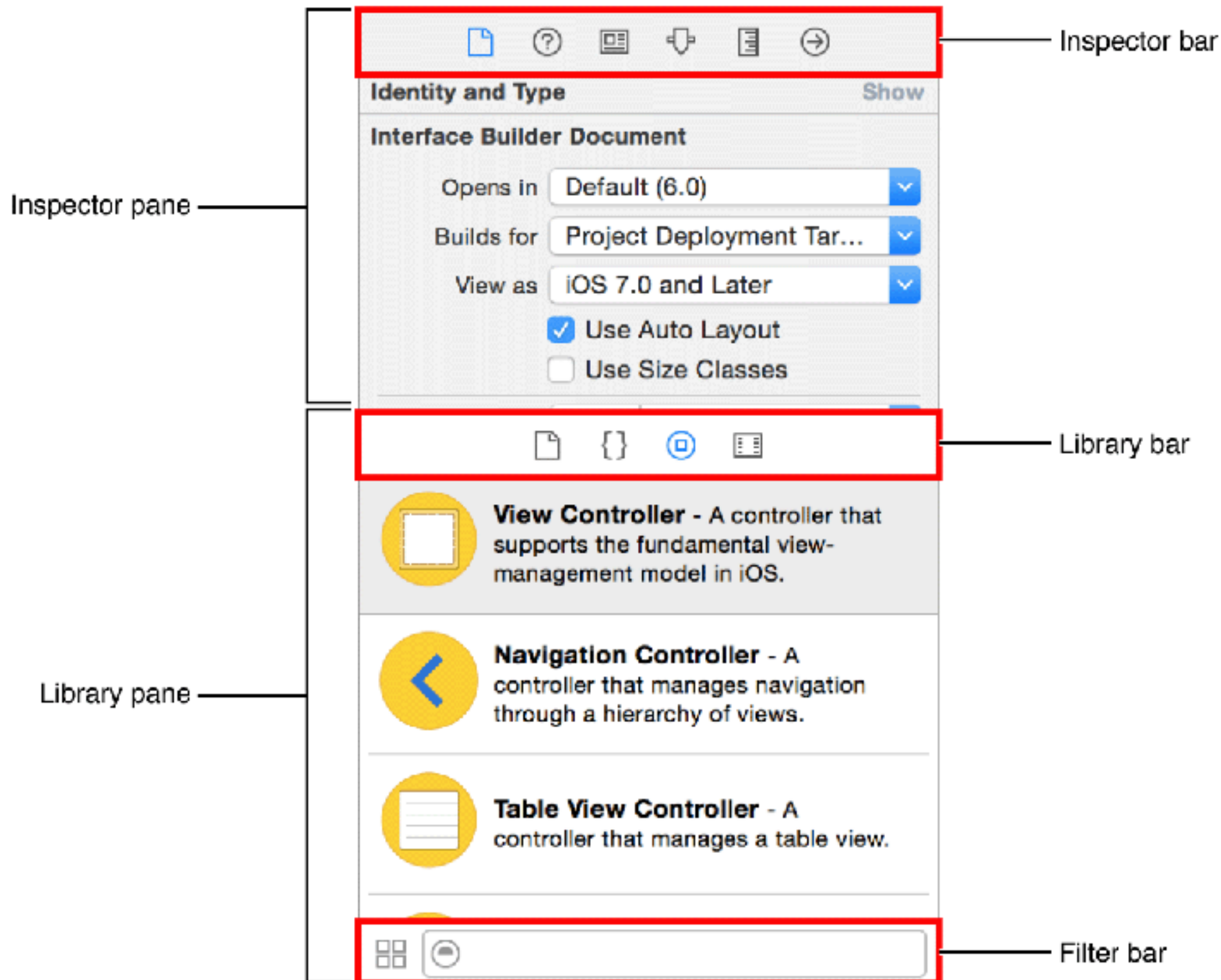
Breaking point로 실행 중 해당 변수의 값을 확인 가능



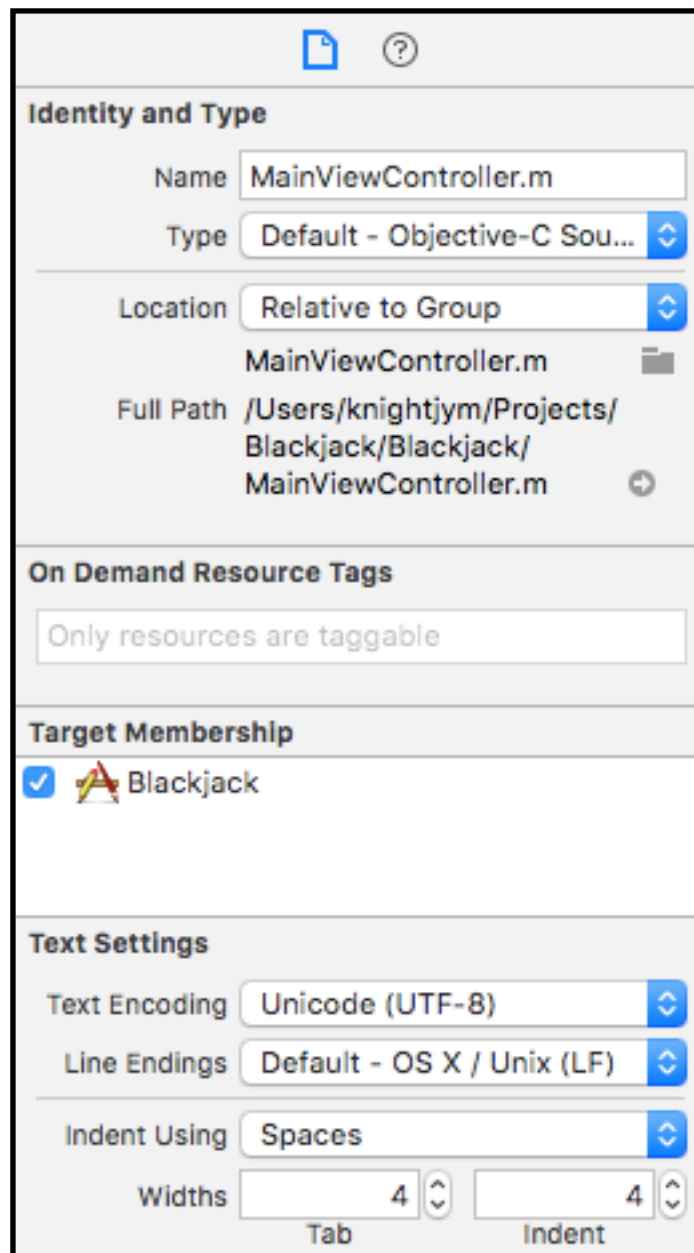
Console

프로그래머의 log출력과 직접메소드 실행이 가능

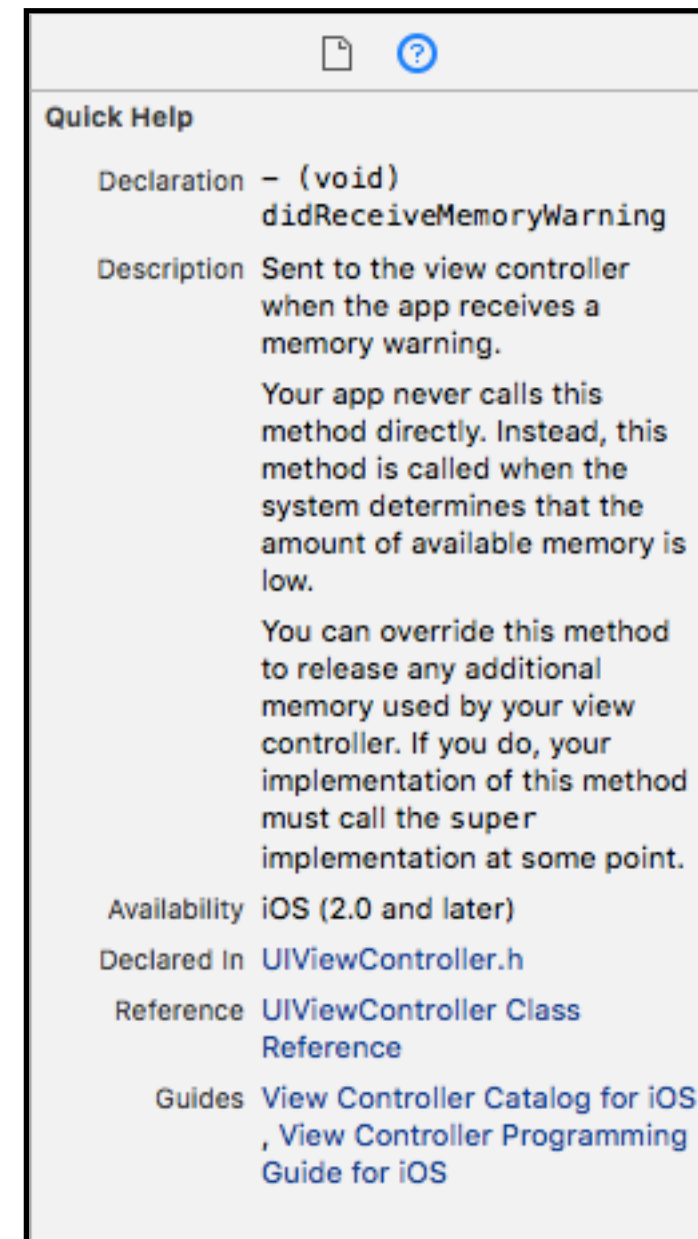
Utilities



Utilities - Inspector1

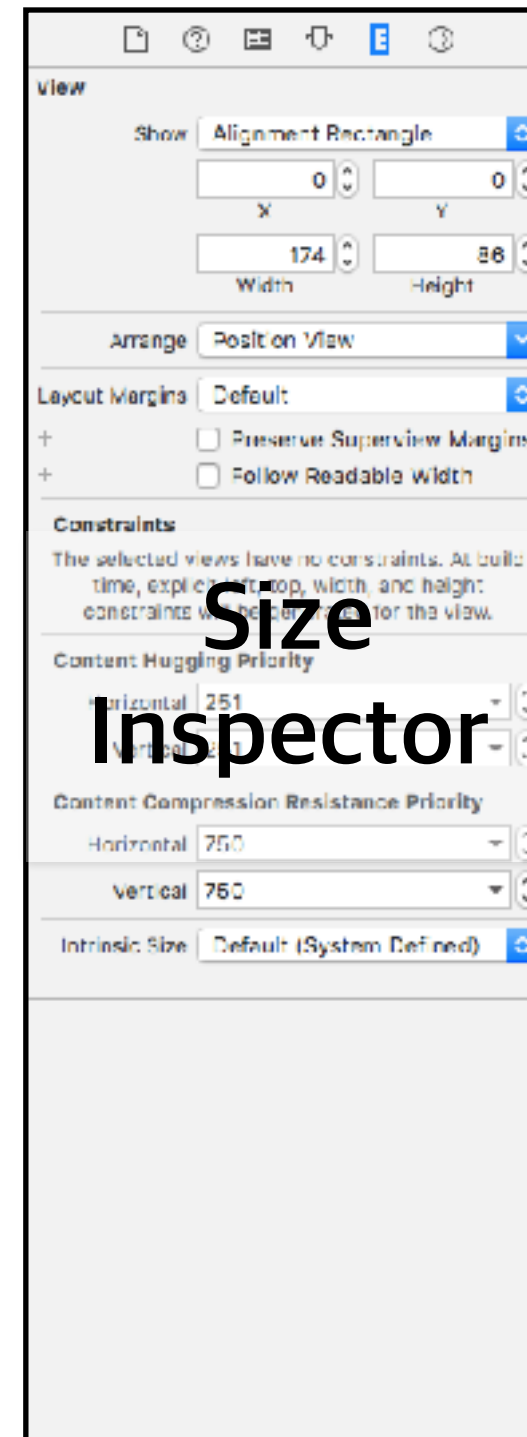
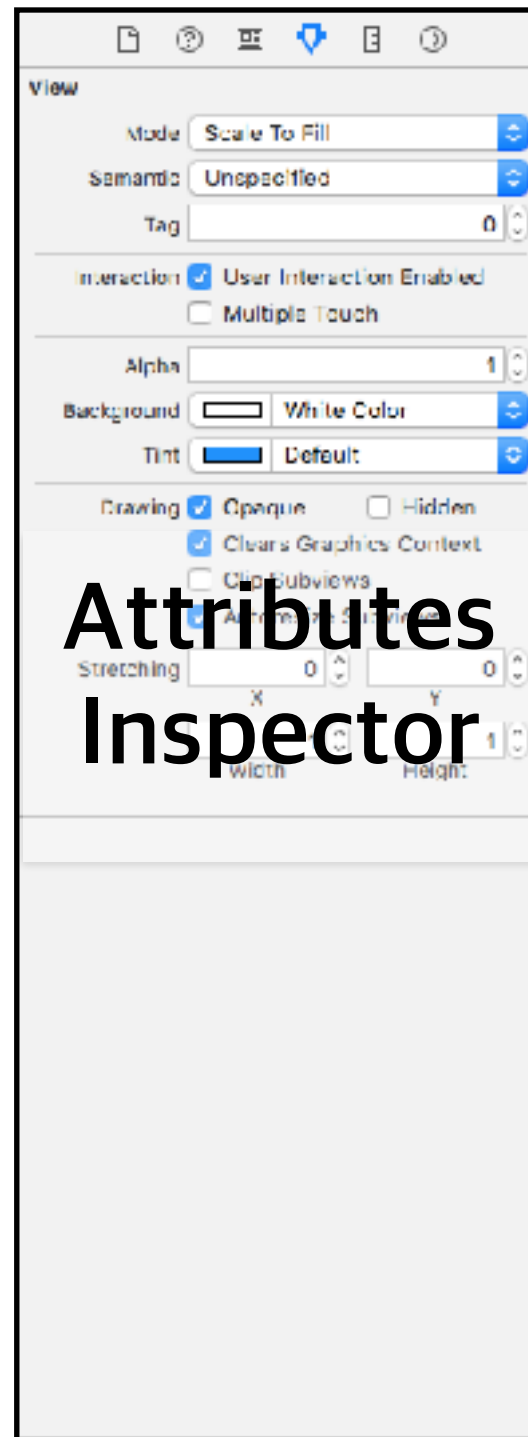
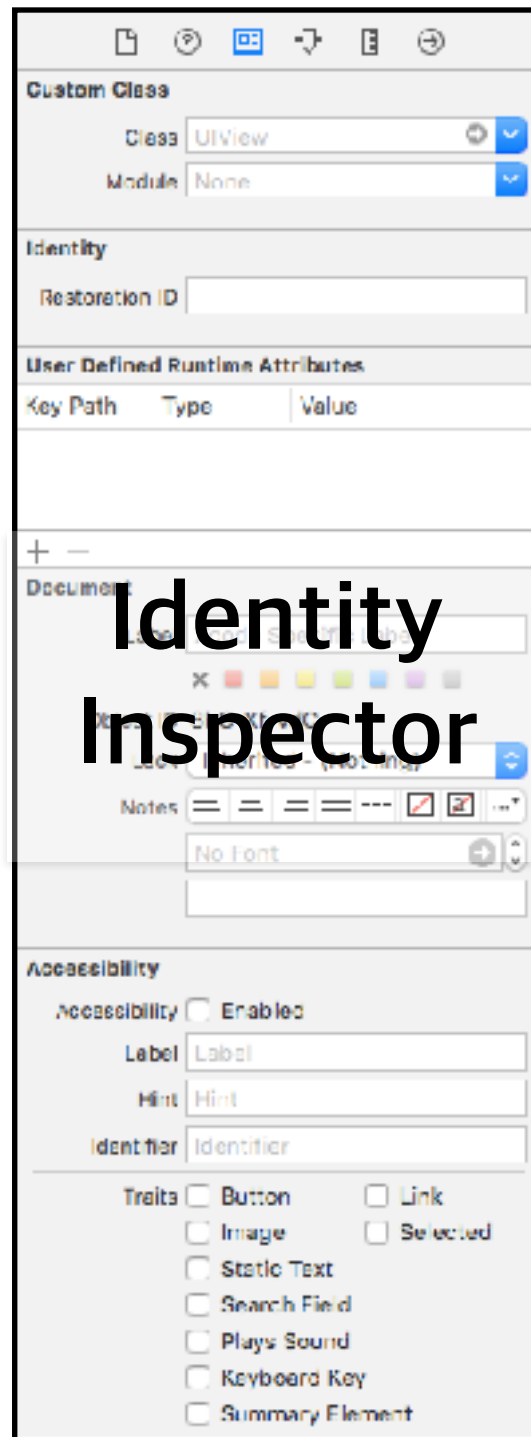


File Inspector



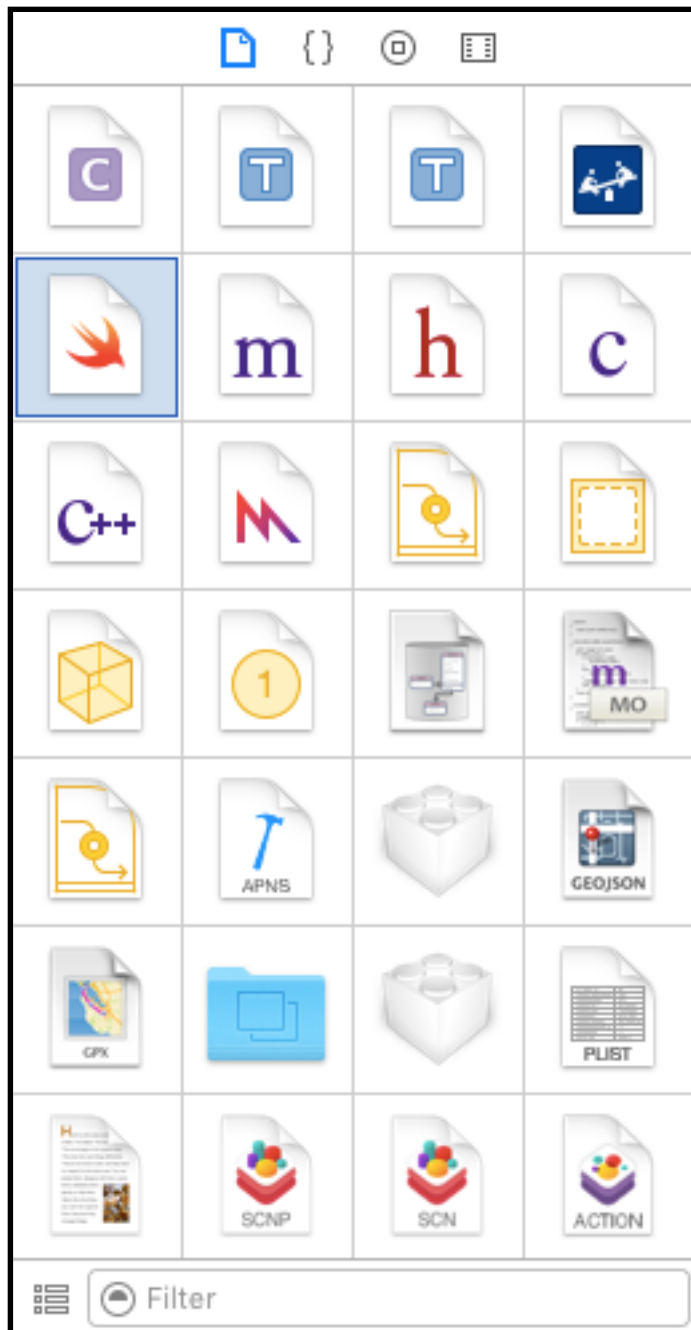
Quick Help Inspector

Utilities - Inspector2 ver UI

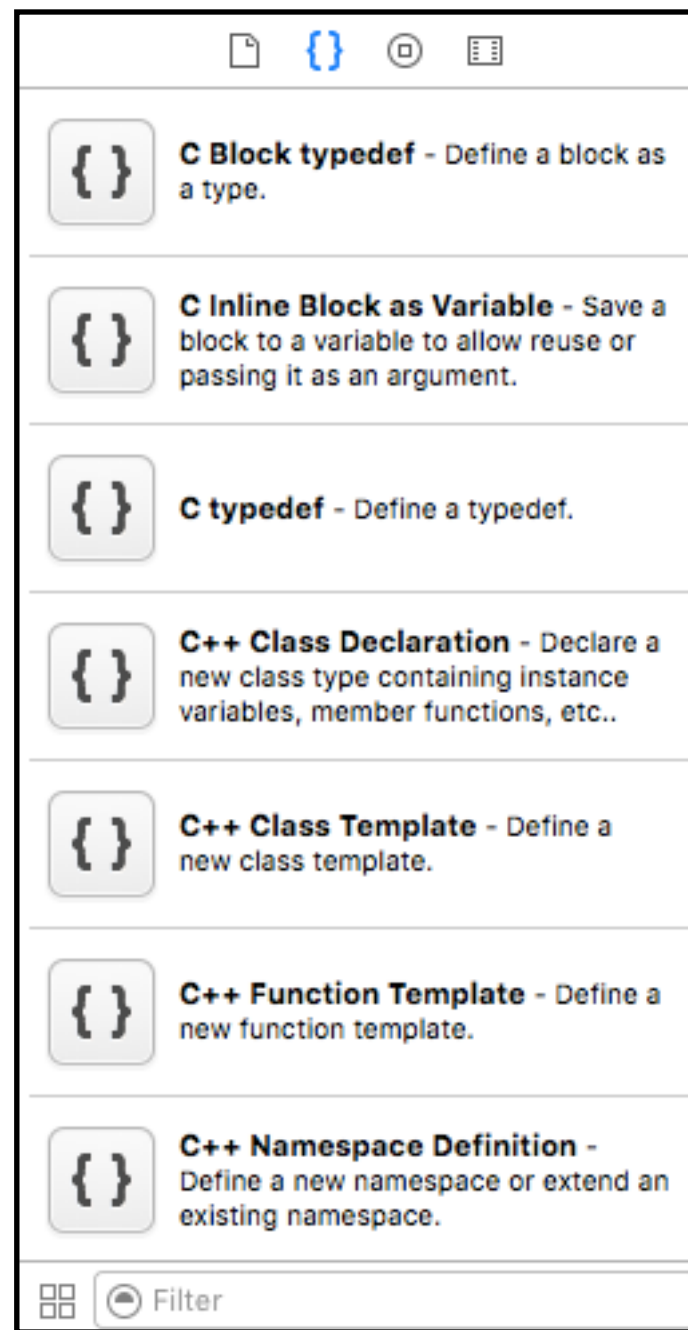


Utilities - library

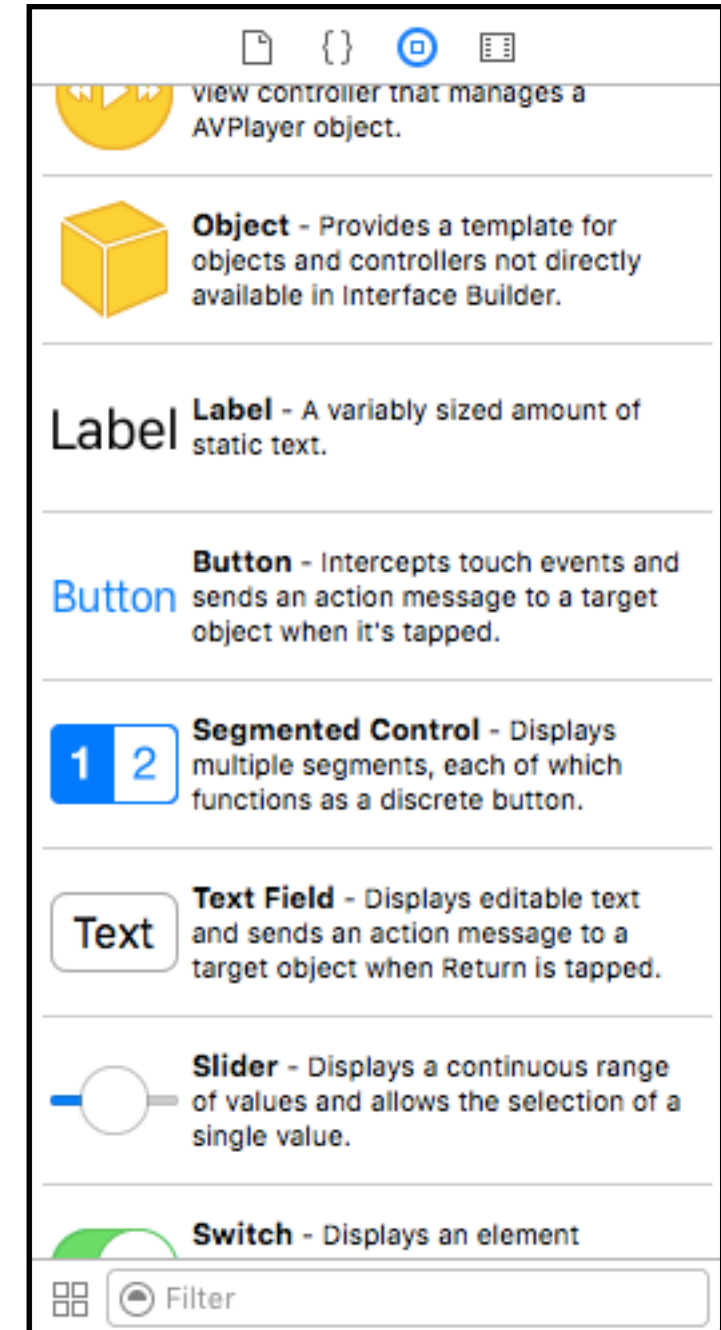
File Template



Code Snippet

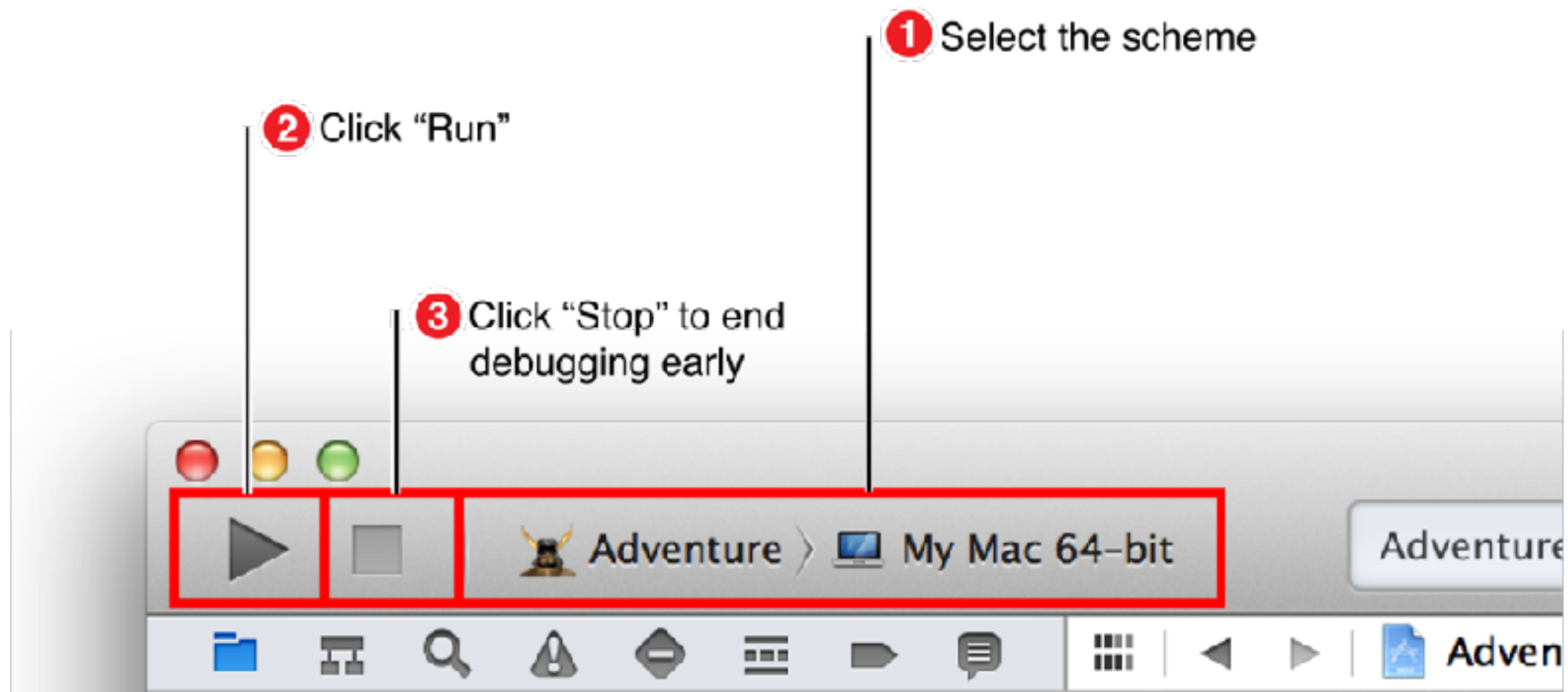


Object



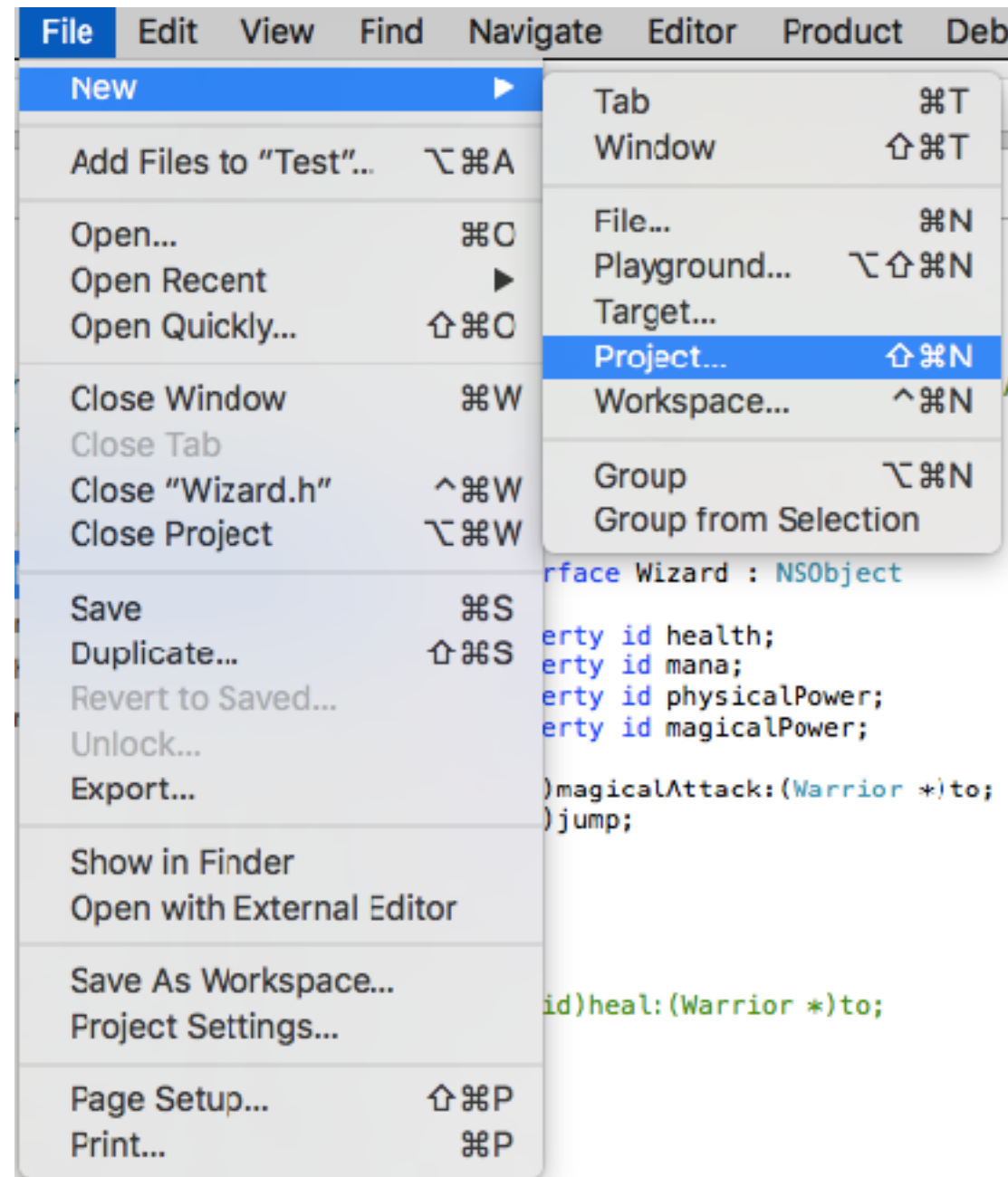
빌드 & 런

1. Select an active scheme and destination.
2. Click Run to build and run your code with the active scheme.
3. Use the Stop button to stop an in-progress build or end the current debugging session.



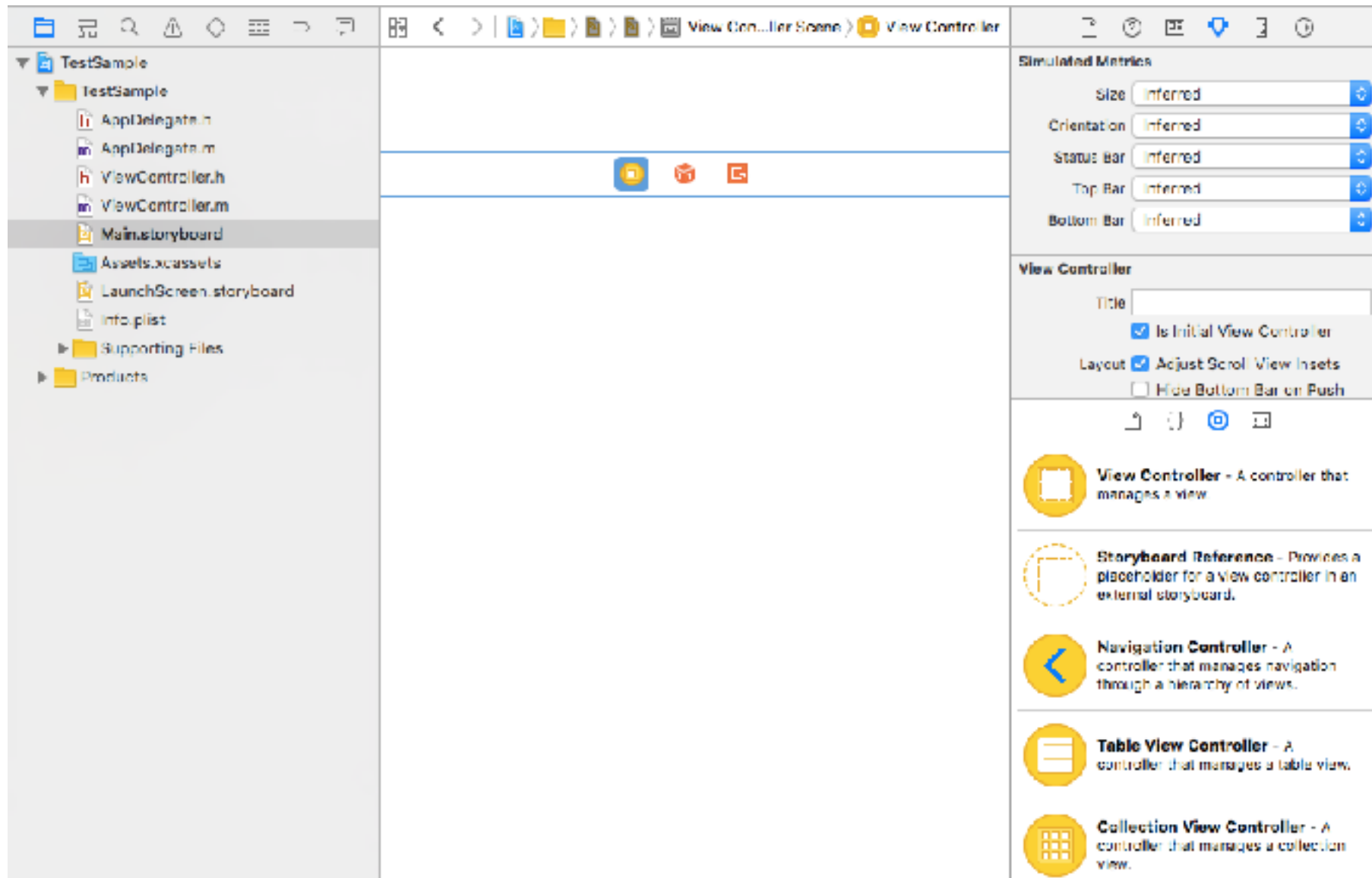
Xcode가지고 놀기

새프로젝트 만들기



Storyboard가지고 놀기

- Utilities - library - Object들을 가지고 놀아보세요



따라해봐요

- break point 찍어보기
- build 해보기
- run 해보기

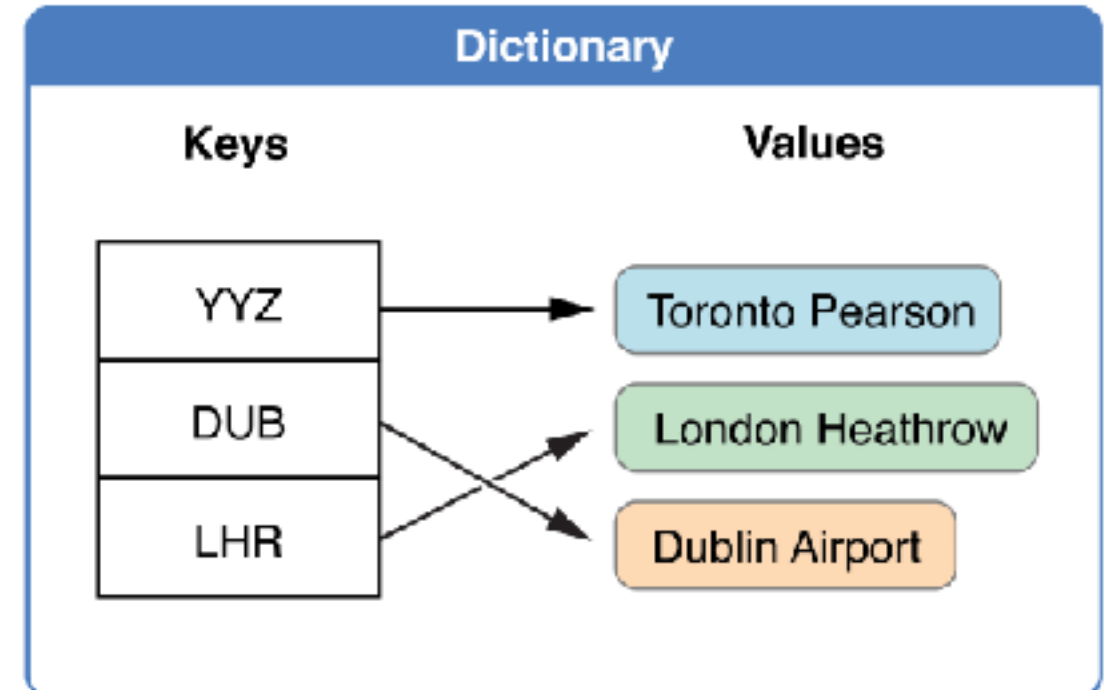
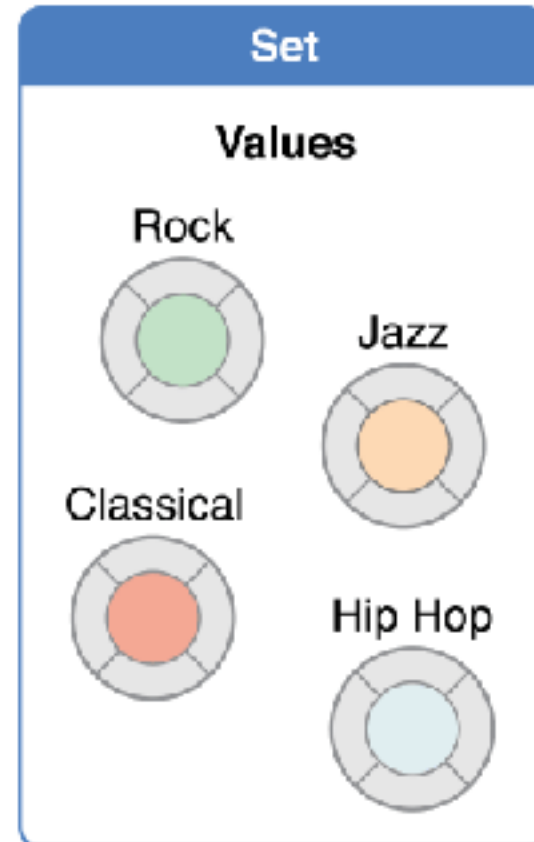
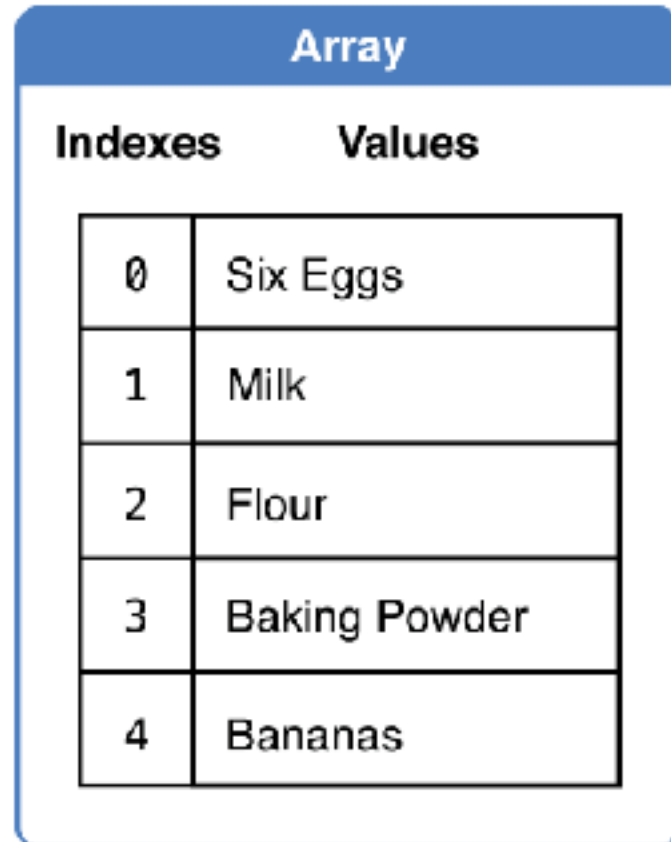
Hello World

- hello world 찍어보기 (Log And UI)

Collection Type

Collection Type

- Swift는 값의 모음을 저장하기 위한 배열, 집합 및 사전이라는 세 가지 기본 형식을 제공 합니다. 배열은 정렬 된 값 모음입니다. 집합은 고유 한 값의 정렬되지 않은 모음입니다. 사전은 키 - 값 연관의 정렬되지 않은 모음입니다.



Mutability of Collections

- 변수(var) 에 할당하면 Collection를 변경가능하다.
- 즉 Collection에 추가, 제거, 수정할수 있다.
- 하지만 상수(let)에 할당하면 Collection를 변경 불가능 하다.

Array

- 배열(영어: array)은 번호(인덱스)와 번호에 대응하는 데이터들로 이루어진 자료 구조를 나타낸다. 일반적으로 **배열에는 같은 종류의 데이터들이 순차적으로 저장**되어, 값의 번호가 곧 배열의 시작점으로부터 값이 저장되어 있는 상대적인 위치가 된다.

Array 문법

- 기본 표현은 `Array<Element>`로 Array Type을 나타낸다.
- 여기에서 `Element`는 배열에 저장할수 있는 타입이다.
- 또 다른 축약 문법으로 `[Element]`로 표현할 수 있다.

```
var someInts:[Int] = [Int]()  
var someInts:Array<Int> = Array<Int>()
```

배열 리터럴

- 배열 리터럴 문법은 대괄호 [] 를 사용한다.

[값 1 , 값 2 , 값 3]

```
var someInts: [Int] = [1,2,3,4]  
someInts = []
```

배열 Element 가져오기

- index를 통해 배열의 값을 가져올수 있다.
- index는 0부터 시작된다.

```
var someInts:[Int] = [1,2,3,4]
print("\ (someInts[0] )")
print("\ (someInts[3] )")
```

배열 추가 기능

- 현재 배열의 element count
- 빈 배열 확인
- element 추가
- element 삽입
- element 삭제

Quick Help

- command + shift + O

Set

- Set은 같은 타입의 데이터가 순서없이 모여있는 자료구조, 각 항목의 순서가 중요치 않거나 한번만 표시해야하는 경우 배열 대신 사용된다.

Set 문법

- 기본 표현은 `Set<Element>`로 Set Type을 나타낸다.
- 여기에서 `Element`는 배열에 저장할수 있는 타입이다.
- Set은 Array와 다르게 축약 문법이 없다.

```
var someInts:Set<Int> = Set<Int>()
```

Set 리터럴 사용

- Set Type으로 설정된 변수는 배열 리터럴을 이용해서 값을 설정할 수 있다.

[값 1 , 값 2 , 값 3]

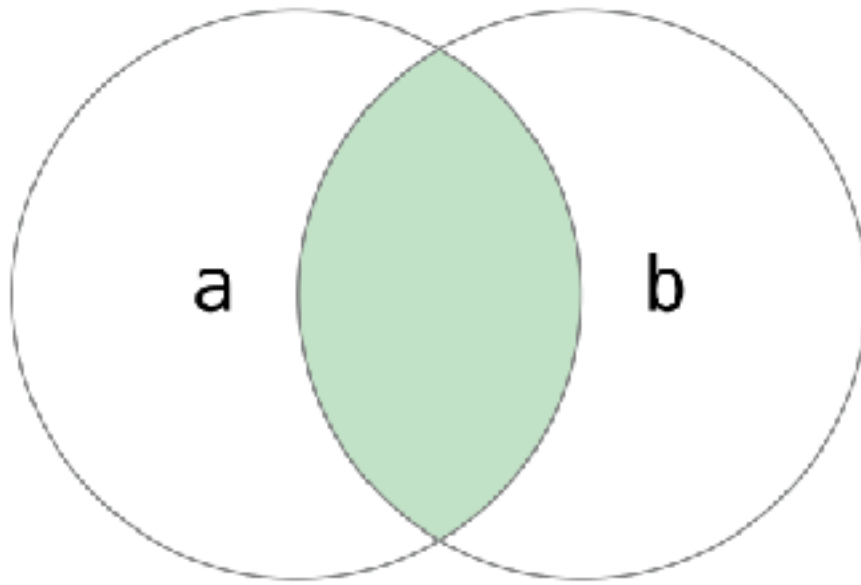
```
var someInts:Set<Int> = [1,2,3,4]  
someInts = []  
var someStrings:Set = ["joo", "young"]
```


Set Element 가져오기

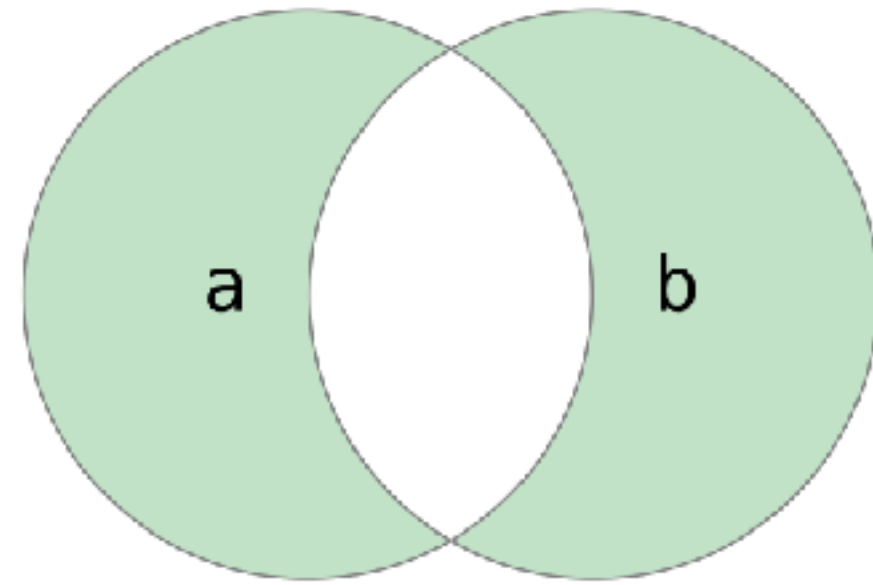
- Set은 순서가 정해져 있지 않기 때문에 for-in 구문을 통해서 데이터를 가져와야 한다.
- 순서는 정해져 있지 않지만 정렬을 통해 데이터를 원하는 순서대로 가져올수 있다.

Set 집합 연산

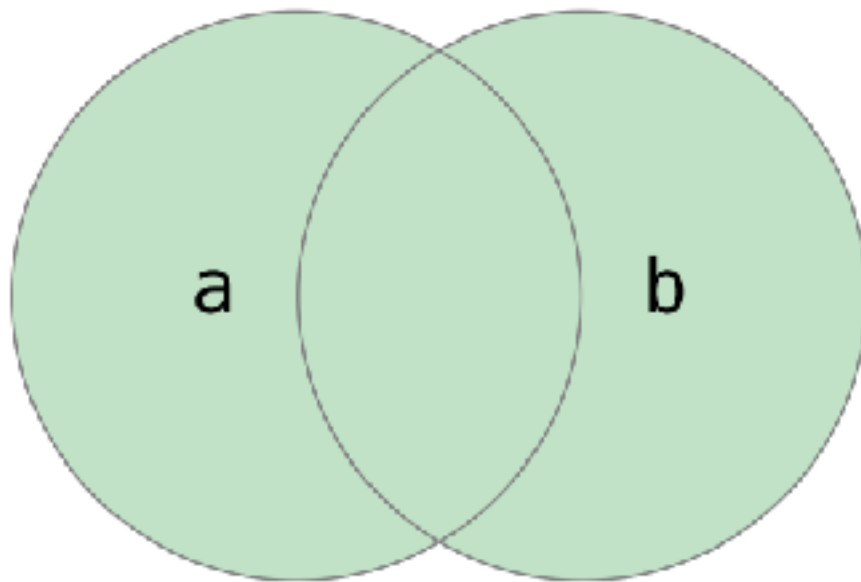
`a.intersection(b)`



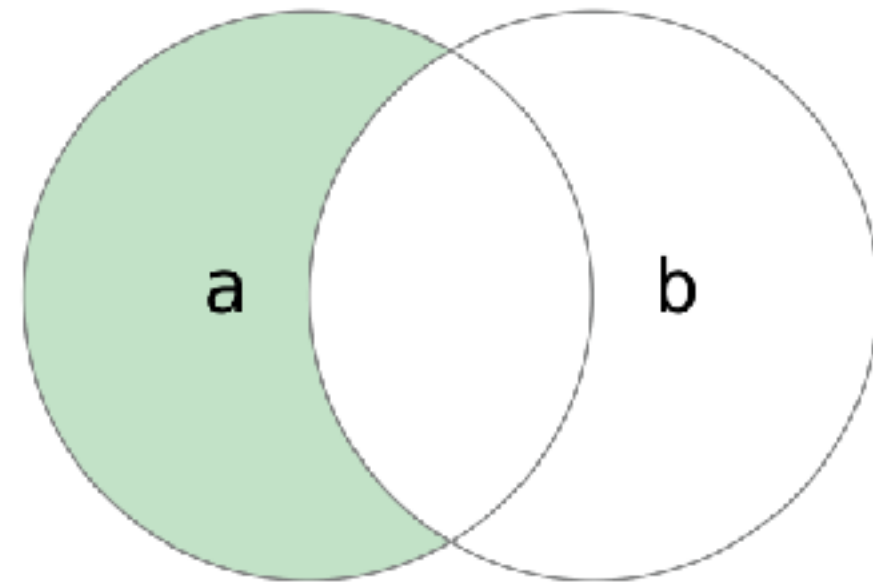
`a.symmetricDifference(b)`



`a.union(b)`



`a.subtracting(b)`



Set 집합 연산

```
var oddDigits : Set = [ 1, 3, 5, 7, 9 ]  
let evenDigits : Set = [2, 4, 6, 8]  
let primeDigits : Set = [2, 3, 5, 7]  
  
oddDigits.intersection(evenDigits)  
  
oddDigits.symmetricDifference(primeDigits)  
  
oddDigits.union(evenDigits).sort()  
  
oddDigits.subtract(primeDigits).sort()
```

Dictionary

- Dictionary는 순서가 정해져 있지 않은 데이터에 키값을 통해 구분할수 있는 자료구조. 항목의 순서가 중요치 않고 key값을 통해서 데이터를 접근할때 사용합니다.

Dictionary 문법

- 기본 표현은 `Dictionary<key, value>`로 Dictionary Type을 나타낸다.
- `Key`값 은 Dictionary에서 value를 가져오는데 사용되는 값이다.
- 또 다른 축약 문법으로 `[key:value]` 로 표현할 수 있다.

```
var someInts: [String: Int] = [String: Int]()  
var someInts: Dictionary<String, Int> = [:]
```

딕셔너리 리터럴

- 딕셔너리의 리터럴 문법은 [:] 를 사용한다.

[키 1 : 값 1 , 키 2 : 값 2 , 키 3 : 값 3]

```
var airports: [String:String] = ["ICH": "인천공항", "CJU": "제주공항"]
```

딕셔너리 Value 가져오기

- key값을 통해 Value값을 가져올수 있다.

```
var airports: [String:String] = ["ICH": "인천공항", "CJU": "제주공항"]  
print("\n(airports["ICH"])\n")  
print("\n(airports["CJU"])\n")
```