Data 저장

강사 주영민

Property list

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)
Item 0	String	armv7
▶ Supported interface orientati...	Array	(3 items)

Property list - plist

- 심플한 “파일” 저장 방법 중 하나.
- Key,Value구조로 데이터 저장
- File 형태로 저장되다 보니 외부에서 Access가능(보안 취약)

파일 위치

- 파일이 저장되는곳 Bundle & Documents 폴더
- Bundle은 프로젝트에 추가된 Resorce가 모인 곳
- 프로그램이 실행되며 저장하는 파일은 Documents폴더에 저장된다.
- **즉! plist파일의 데이터만 불러오는 역할은 Bundle을 통해서, plist파일에 데이터를 쓰고 불러오는 역할은 Documents폴더에 저장된 파일로!**

Plist File In Bundle

1. bundle에 있는 파일 Path 가져오기
2. Path를 통해 객체로 변환, 데이터 불러오기
3. 사용

Bundle

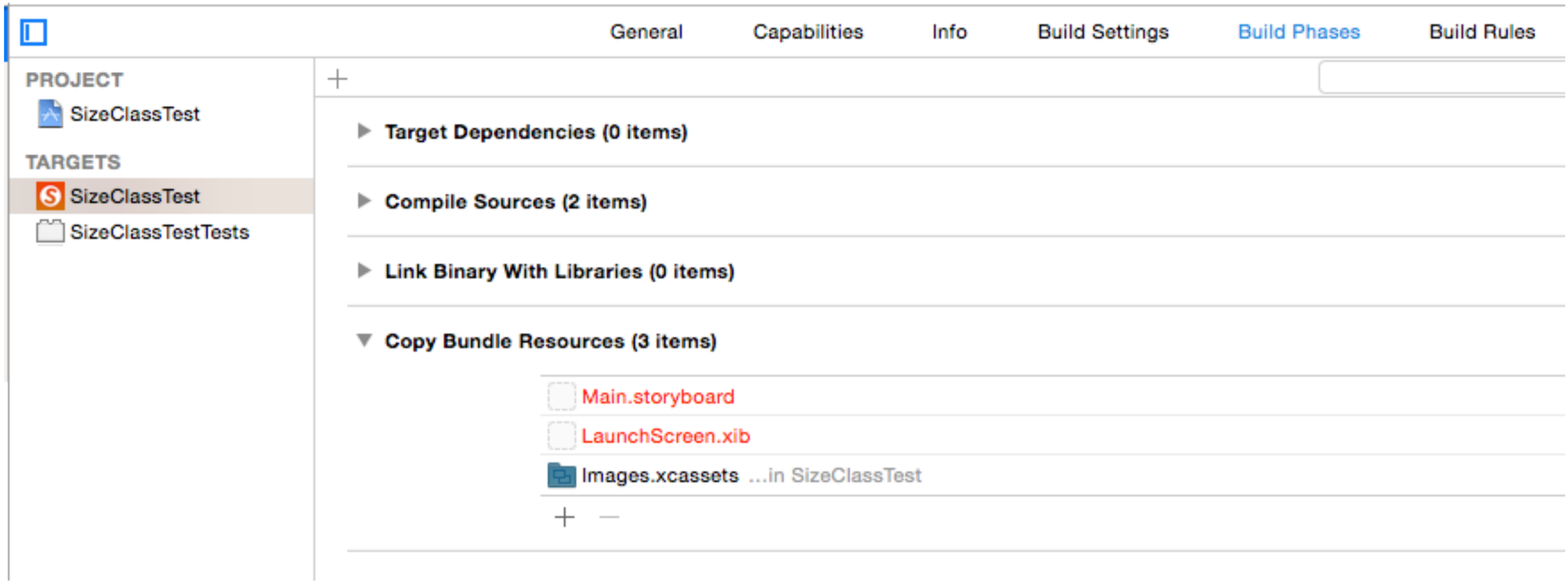
강사 주영민

Fast campus

Bundle

- 실행코드, 이미지, 사운드, nib 파일, 프레임 워크,설정파일 등 코드와 리소스가 모여있는 file system내의 Directory

Bundle 리소스 확인



Main Bundle 가져오기

```swift
// Get the main bundle for the app.
let mainBundle = Bundle.main
```

리소스 파일 주소 가져오기

```
// Get the main bundle for the app.
let mainBundle = Bundle.main
let filePaht = mainBundle.path(forResource: "rName", ofType:
"rType")
```

데이터 불러오기

```
if let path = filePaht {

    let image = UIImage(contentsOfFile: filePaht!)

}
```

Plist File In Bundle

```swift
if let filePaht = mainBundle.path(forResource: "rName", ofType: "rType"),
   let dict = NSDictionary(contentsOfFile: filePaht) as? [String: Any]
{
   // use swift dictionary as normal
}
```

Plist파일 불러오기

- 앱내 저장되어 있는 plist 리소스 파일의 데이터는 번들을 통해 가져올 수 있다.

Plist File In Document

1. Document folder Path 찾기

2. Document folder에 plist 파일이 있는지 확인
   - 만약 없다면 : bundle에 있는 파일 Document에 복사

3. Path를 통해 객체로 변환, 데이터 불러오기

4. writeToFile 메소드로 파일 저장

1. 파일 불러오기 (NSSearchPathForDirectoriesInDomains)

```
let path:[String] =
NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask,
true)
let basePath = path[0] + "/fileName.plist"
```

- doucument 폴더에 Path구하기

2. Document folder에 파일 있는지 확인

```
if !FileManager.default.fileExists(atPath: basePath)
{

}
```

- doucument 폴더에 plist파일이 존재 하지는지 확인

3. bundle에 있는 파일 Document에 복사

```
 if let fileUrl = Bundle.main.path(forResource: "fileName", ofType: "plist")
{
    do {
        try FileManager.default.copyItem(atPath: fileUrl, toPath: basePath)
    } catch  {
        print("fail")
    }
}
```

- 만약 doucument에 해당 plist파일이 존재 하지 않을때, bundle에 있는 파일을 document폴더로 복사

4. Dictionary 인스턴스 불러오기

```swift
if let dict = NSDictionary(contentsOfFile: basePath) as? [String: Any]
{
   // use swift dictionary as normal
}
```

- document 폴더에 있는 파일을 NSDictionary을 통해서 Dictionary인스턴스로 불러오기

5. write(toFile)메소드를 통해 파일 저장

```
if let dict = NSDictionary(contentsOfFile: basePath) as? [String: Any]
{
    var loadData = dict
    loadData.updateValue("addData", forKey: "key")

    let nsDic:NSDictionary =  NSDictionary(dictionary: loadData)
    nsDic.write(toFile: basePath, atomically: true)
}
```

- dictionary를 수정
- NSDictionary로 변경후 writeTofile 메소드를 통해 파일에 저장

실습

- 한번 같이 만들어 볼까요?

Singletone Pattern

강사 주영민

Singleton Pattern

1. 싱글톤 이란? 어플리케이션 전 영역의 걸쳐 하나의 클래스의 단 하나의 인스턴스만(객체)을 생성하는 것을 싱글톤 패턴이라고 한다.

2. 사용이유 : 어플리케이션 내부에서 유일하게 하나만 필요한 객체에서 사용(셋팅, 데이터 등)

3. 클래스 메소드로 객체를 만들며 static을 이용해서 단 1개의 인스턴스만 만든다.

4. 앱 내에서 공유하는 객체를 만들 수 있다.

Singleton Pattern 인스턴스 만들기

```
class SingletonClass {
    // MARK: Shared Instance
    static var sharedInstance:SingletonClass = SingletonClass()

    // Can't init is singleton
    private init()
    {
        //초기화가 필요하면 private로 생성
    }

}
```

Singleton Pattern 예제

```
//스크린 정보를 가지고 있는 객체
let screen = UIScreen.main

//사용자 정보를 저장하는 객체
let data = UserDefaults.standard

//어플리케이션 객체
let app = UIApplication.shared

//파일 시스템 정보를 가지고 있는 객에
let fileManager = FileManager.default
```

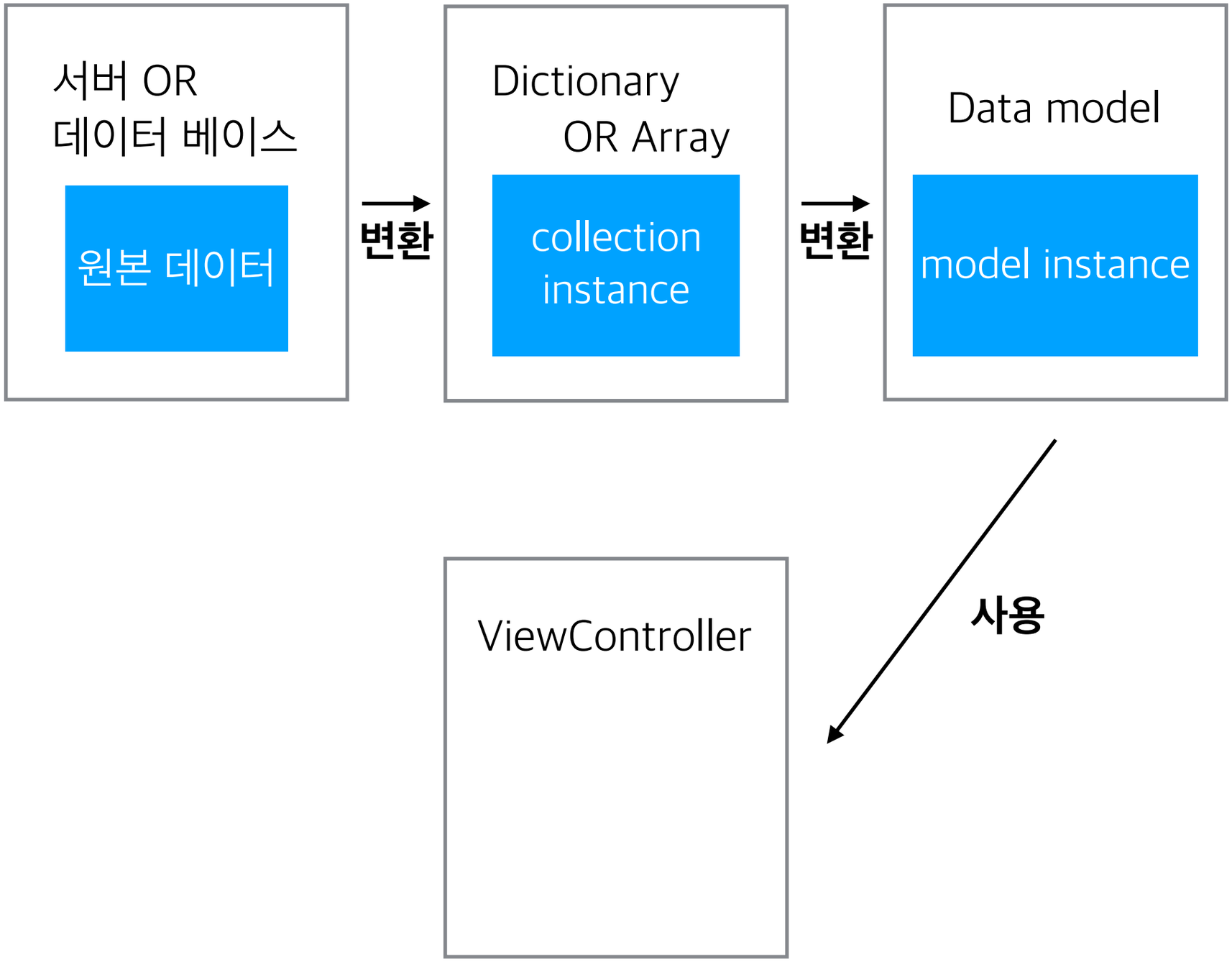
“싱글턴 패턴은 왜 사용하는 것일까?”

데이터 모델 만들기

Struct로 데이터 모델 만들기

- 서버나 데이터베이스에서 가져온 데이터를 바로 사용하지 않고 데이터 모델을 만들어서 사용
- Struct로 데이터 모델을 만들어 사용한다.(데이터의 경우 참조형 보다는 값 복사형 타입이 좋기때문)

데이터 컨트롤 흐름



왜 데이터 모델을 사용하나?

- 복잡한 구조의 Collection Type을 주로 사용하게 되며, 매번 사용할때마다 데이터를 끄집어 내기위해 Array & Dictionary instance를 만들어야 한다. (사용 편의성)

- Dictionary의 데이터는 key값을 통해 데이터에 접근 한다. String type인 key값은 다양한 곳에서 사용하게 되면 오타의 위험성이 커진다. (안정성)

- 데이터중 Dictionary의 key값 변경시 코드내 사용된 모든곳을 찾아서 직접 바꿔줘야 한다. (수정 용의)

데이터 모델 만들기

- 데이터 하나당 Property 하나로 매칭
- 필수 데이터와 기타 데이터를 접근 제한자로 구분
- 데이터 구조 단순화 작업
- 가공이 필요한 데이터 만들기

예제 : 로그인 정보

- 로그인 데이터

1. 사용자 ID
2. 비밀번호
3. 이메일
4. 추가 정보 : 생년월일, 성별,

- 실제 더미 데이터

```
let userDic:Dictionary<String,Any>
  = ["userID":"joo",
     “userPW":"12345!@",
     "email":"knightjym@gmail.com",
     "birthDay":"2017/10/15",
     "gender":1
    ]
```

예제 : 로그인 정보

- 실제 더미 데이터

```
let userDic:Dictionary<String,Any>
  = ["userID":"joo",
     “userPW":"12345!@",
     "email":"knightjym@gmail.com",
     "birthDay":"2017/10/15",
     "gender":1
]
```

- Model

key	valueType
userID	String
userPW	String
email	String
birthday	String
gender	Int

예제 : 로그인 정보

- Model

key	valueType
userID	String
userPW	String
email	String
birthday	String
gender	Int

- DataModel Struct

```
struct UserModel
{
    var userID:String
    var pw:String
    var email:String
    var birthday:String?
    var gender:Gender?

    init?(dataDic:[String:Any])
    {
        //다음 페이지

    }
}
```

로그인 정보 초기화 메소드 예제

```
init?(dataDic:[String:Any])
{
    //필수 항목 모델화
    guard let userID = dataDic["UserID"] as? String else {return nil}
    self.userID = userID

    guard let pw = dataDic["userPw"] as? String else {return nil}
    self.pw = pw

    guard let email = dataDic["email"] as? String else {return nil}
    self.email = email
   //옵셔널 항목 모델화
    self.birthday = dataDic["birthDay"] as? String

    if let rawData = dataDic["gender"] as? Int,
           (rawData == 1 || rawData == 2)
    {
        self.gender = Gender(rawValue:rawData)
    }
}
```

복잡한 데이터 모델

- 음악앨범 정보

1. 앨범 정보
   - 앨범 타이틀
   - 가수 이름
   - 장르

2. [노래 정보 리스트]
   - 노래 정보
     - 노래제목
     - 가수
     - 작곡가, 작사가
     - 총 플레이 시간
     - 노래 URL

- 실제 더미 데이터

```
let album:Dictionary<String,Any> =
    ["albumInfo":["albumTitle":"2집 Oh!",
                  "artist":"소녀시대",
                  "genre":"댄스"],
     "songList":[["songTitle":"Oh!",
                  "trackNum":1,
                  "artist":"소녀시대",
                  "writer":"김정배",
                  "playTime":12340,
                  "playURL":"http://music.naver.com/123"],
                 ["songTitle":"Show! Show! Show!",
                  "trackNum":2,
                  "artist":"소녀시대",
                  "writer":"김부민",
                  "playTime":10130,
                  "playURL":"http://music.naver.com/124"],
                 ["songTitle":"웃자 (Be Happy)",
                  "trackNum":4,
                  "artist":"소녀시대",
                  "writer":"이트라이브",
                  "playTime":12134,
                  "playURL":"http://music.naver.com/126"]
          ]
]
```

- Model

key	valueType		
albumInfo	Dictionary		
	key	valueType	
	albumTitle	String	
	artist	String	
	genre	String	
songList	Array		
	[	Dictionary	
		key	valueType
		songTitle	String
		trackNum	Int
		artist	String
		writer	String
		playTime	Int
		playURL	String
	]		

DataModel Struct

- 복잡한 구조의 데이터를 구분하여 만들기 위해선?!
- 키-Value가 있는 Dictionary구조마다 데이터 모델 만들기
- AlbumModel, AlbumInfo, SongData

DataModel Struct

```
struct AlbumInfo
{
    var albumTitle:String
    var artist:String
    var genre:String

    init?(dataDic:[String:Any])
    {
        //이전방법과 동일
    }
}
```

```
struct SongData
{
    var songTitle:String
    var trackNum:Int
    var artist:String
    var writer:String
    var playTime:Int
    var playURL:String

    init?(dataDic:[String:Any])
    {
        //이전방법과 동일
    }
}
```

DataModel Struct

```
struct AlbumModel
{
    var albumInfo:AlbumInfo
    var songList:[SongData] = []

    init?(dataDic:[String:Any])
    {
        guard let infoDic = dataDic["albumInfo"] as?
                            Dictionary<String,Any> else {return nil}
        //데이터 인스턴스 만들기
        albumInfo = AlbumInfo(dataDic: infoDic)!

        guard let list = dataDic["songList"] as?
                         [Dictionary<String,Any>] else {return nil}
        //for문을 통해 각 데이터를 모델로 만든후 Array에 추가
        for songDic in list
        {
            songList.append(SongData(dataDic: songDic)!)
        }
    }
}
```