

『Swift実践入門』 サンプルコード

技術評論社刊『Swift実践入門』（石川洋資、西山勇世著）のサンプルコードについて説明します。

PlaygroundファイルとXcodeプロジェクト

本書のサンプルコードはPlaygroundファイル、もしくはXcodeプロジェクトとして提供しています。

基本的にはPlaygroundのファイルを利用しますが、次のケースでは例外的にXcodeプロジェクトを利用しています。

- Objective-Cのコードと連携するケース
- 複数のソースコードをコンパイルして、1つのプログラムにするケース

具体的には第1章、第14章、第15章が該当します。

サンプルコードの構成

ここでは、サンプルコードの構成について説明します。

Playgroundファイルの構成

Playgroundファイルは `playground` ディレクトリ内に配置しています。

1つの章のに対して1つのPlaygroundファイルを、1つのコードブロックに対して1つのPlaygroundのページを用意しています。

Playgroundファイル名は、 `04_function.playground` のように該当する章の英語名と同じになっています。

Playgroundのページ名は、 `節番号-連番` という形式になっています。たとえば、4.2節の3つ目のサンプルコードであれば `4.2-3` となります。

Xcodeプロジェクトの構成

Xcodeプロジェクトは `xcode` ディレクトリ内に配置しています。

`xcode` ディレクトリは章ごとのサブディレクトリに分かれており、サブディレクトリ名は `01_introduction` のように該当する章の英語名と同じになっています。

第1章、第15章

第1章、第15章の一部のコードは、対応するコード片ごとにXcodeプロジェクトを提供しています。

各章のXcodeプロジェクトは、1つのWorkspaceにまとめられています。たとえば、第1章のWorkspaceは `01_introduction.xcworkspace` です。このWorkspaceを開くことで、その章のすべてのXcodeプロジェクトにアクセスできます。

各章のディレクトリは、コード片ごとに `節番号-連番` という名前のサブディレクトリに分かれています。各サブディレクトリには、 `節番号-連番` という名前のXcodeプロジェクトと、すべてのソースコードをまとめる `Sources` というディレクトリがあります。なお、ここでの連番は、Playgroundで用いている連番とは独立しています。

これらの構成を図示すると、次のようになります。

```
章名 (例: 01_introduction)
├─ 章名.xcworkspace (例: 01_introduction.xcworkspace)
├─ 連番 (例: 1.1-1)
│   └─ Sources
│       ├── ソースコード1
│       ├── ソースコード2
│       └─ .....
└─ 連番.xcodeproj (例: 1.1-1.xcodeproj)
.....
```

第14章

第14章は全体を通じて1つのプログラムの作り方について説明する章なので、1つのXcodeプロジェクトだけを用意しています。

サンプルコードの実行方法

ここでは、PlaygroundファイルとXcodeプロジェクトそれぞれの実行方法を説明します。

Playgroundファイルの実行方法

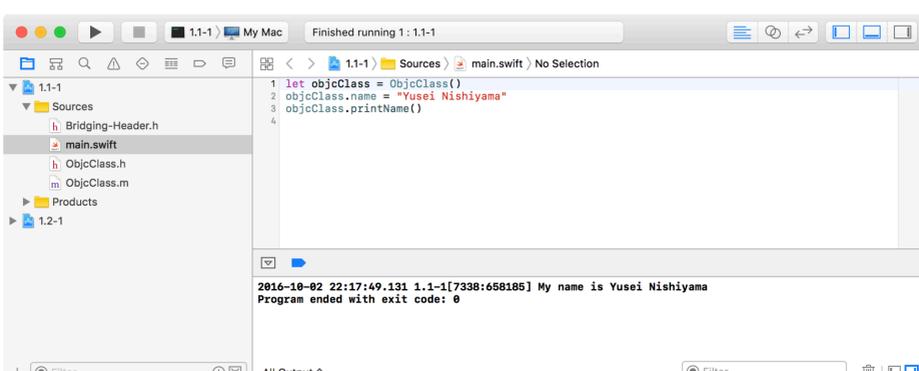
対応する章のPlaygroundを開きます。そして実行したいコード片のページを開きます。次の図のように、右ペインにはPlaygroundでの評価結果、下ペインには標準出力への出力結果が表示されます。



Playgroundの実行結果

Xcodeプロジェクトの実行方法

対応する章のWorkspaceを開きます。そしてXcodeのメニューの「Product」→「Scheme」から実行したいコード片に対応する連番を選択します。連番の選択後、Xcodeのメニューから「Product」→「Run」（command+Rキー）を選択することで、コンパイルとプログラムの実行が行われます。次の図のように、出力結果は下ペインに表示されます。



Xcodeプロジェクトの実行結果

サンプルコード内でのコメントについて

サンプルコード内では、次の2つのケースでコメントを使用しています。

- 紙面でコメントを用いた補足を行っているケース
- コンパイルエラーや実行時エラーを避けるケース

以下、それぞれについて説明します。

紙面でコメントを用いた補足を行っているケース

次のような紙面でのコメントによる補足は、サンプルコードにもそのまま掲載しています。

```
var a: Int // 変数はInt型

a = 456 // Int型の代入はOK
a = "abc" // String型の代入はコンパイルエラー
```

コンパイルエラーや実行時エラーを避けるケース

成功が期待されるコードとエラーが発生するコードが併記されている場合、エラーが発生するコードをコメントアウトしています。

たとえば次の例では、 `a = "abc"` がコード全体をコンパイルできなくしてしまい、実行結果を確認できないので、エラーとなる箇所をコメントアウトしています。

```
var a: Int // 変数はInt型

a = 456 // Int型の代入はOK
// a = "abc" // String型の代入はコンパイルエラー
```

ただし、次のコードのように全体がコンパイルエラーになっても意図が正しく伝わる場合は、コメントアウトしていません。

```
let integers1 = [1, 2, 3]
integers1.removeAtIndex(1) // コンパイルエラー
```