



Full-Stack Service Programming

# Lecture 1

## Dart 언어의 이해 (심화)

2023. 09. 01

Sungwon Lee  
Department of Software Convergence

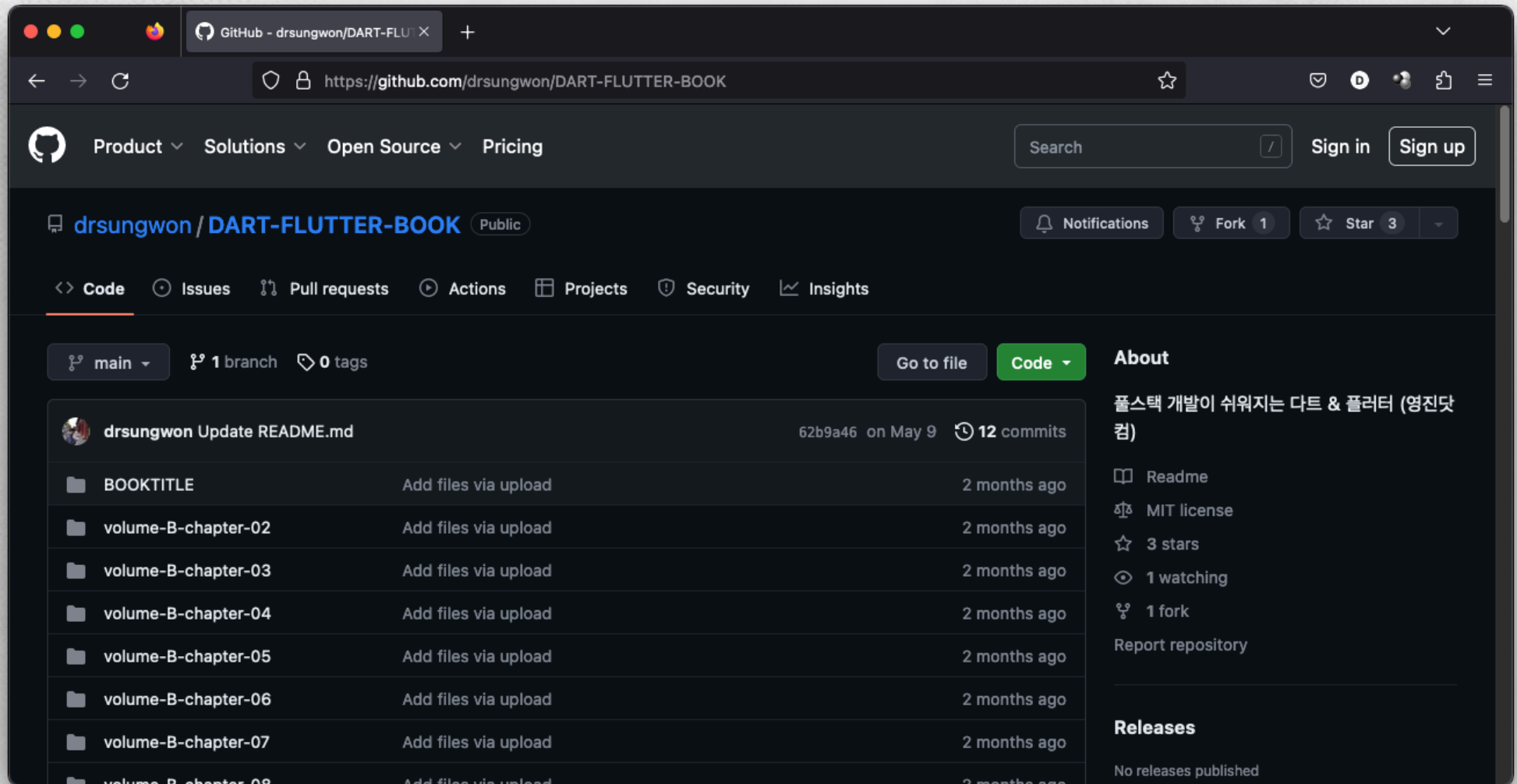
● Volume.C Dart 언어 심화 이해



# TEXT BOOK & SOURCE CODE

소스 코드

● <https://github.com/drsungwon/DART-FLUTTER-BOOK>





## Class 직접 만들기 Part.1

- class 키워드

```
class 클래스 이름 {  
  
}
```

- 인스턴스 변수와 메소드 (instance variable & methods)

- ✱ 객체(혹은 인스턴스)에 포함된 변수 및 함수(메소드)

### ONE MORE : late

- Null-Safety에 의해서 null 값을 가질수 없는 변수에 대해서,
- “일부러 지금 초기화하지 않고, 나중에 초기화를 하겠다”고 선언하는 문법임

## Class 직접 만들기 Part.1

---

- 생성자 (constructor)
  - ✘ 클래스의 객체가 생성되는 최초 시점에 호출되는 함수(메소드)
  - ✘ 클래스의 이름과 메소드의 이름이 동일함
  - ✘ 리턴 값을 갖지 않음
- Get 유형 및 Set 유형 메소드
  - ✘ 객체에 저장된 인스턴스 변수의 값을 읽거나 쓰기 위함 메소드
- getter 및 setter 문법
  - ✘ 간단한 Get/Set 유형 메소드를 만드는 문법
  - ✘ 예시) String get asString => "\$\_value"
  - ✘ 예시) set value(int givenValue) => \_value = givenValue



## Class 직접 만들기 Part.1

---

- 연산자 오버로딩 (operator overloading)
  - ✘ 기존 클래스에서 지원하는 메소드의 이름을 동일하게 사용함
  - ✘ 새로운 기능을 클래스에 추가함
- 유전의 법칙 (sub-class, inheritance)
  - ✘ extends 문법을 사용함  
class DERIVED-CLASS **extends** BASE-CLASS {  
}
  - ✘ Base class를 override 할 때, 명시적으로 @override 사용함  
**@override**  
void overriddenMethod(int givenValue) {  
}

## Class 직접 만들기 Part.1 (리뷰 및 실습)

---

- volume-C-chapter-01.dart

### ONE MORE : Initialization List

- 메소드의 이름 오른쪽에 ':' 문법을 사용해서, 인스턴스 변수를 초기화 함  
`Integer([int givenValue = 0]) : _value = givenValue {  
 }`

## Class 직접 만들기 Part.2

### ● mixin 클래스

- ✖ 사실상 클래스이나 스스로 객체가 되어 사용되기 보다는,
- ✖ 다른 클래스의 부속품으로 사용되는 클래스임
- ✖ class 문법 대신 mixin 문법을 사용함

```
25 mixin ActivationFlag {  
26     bool _flag = true;  
27  
28     bool get activated => _flag;  
29     set activated(bool givenFlag) => (_flag = givenFlag);  
30 }  
31
```



## Class 직접 만들기 Part.2

---

- mixin 클래스의 활용

- ✖ extends가 아닌 with 문법으로 다른 클래스에 포함됨

```
32 class TimemachineInteger extends Integer with ActivationFlag {
```

## Class 직접 만들기 Part.2 (리뷰 및 실습)

### ● volume-C-chapter-02.dart

#### NOTE

다양한 클래스에서 자주 사용할 만한 기능들을 mixin으로 만들어 놓은 후, 필요할 때 with 문법으로 클래스에 적용하면, 클래스의 개발이 보다 빨라지게 됩니다. 자주 사용하는 mixin들은 코드의 에러가 점점 줄어들 테니 프로그램의 안정성도 증가한다고 볼 수 있습니다.

#### NOTE

mixin을 아무 클래스나 적용하지 않고 특정 클래스에만 적용하였으면 좋겠다고 생각할 수 있습니다. 이런 경우는 'on' 문법을 사용합니다. 만약에 이번 챕터의 TimemachineInteger에만 적용시키고 싶다면 **25**를 다음과 같이 수정하면 됩니다.

```
mixin ActivationFlag on TimemachineInteger {
```

많이 사용하는 문법은 아니지만, 범용적으로 쓰이지 않고 제한된 클래스에서만 사용되는 mixin을 개발하고자 한다면 기억해 둘 만한 문법입니다.

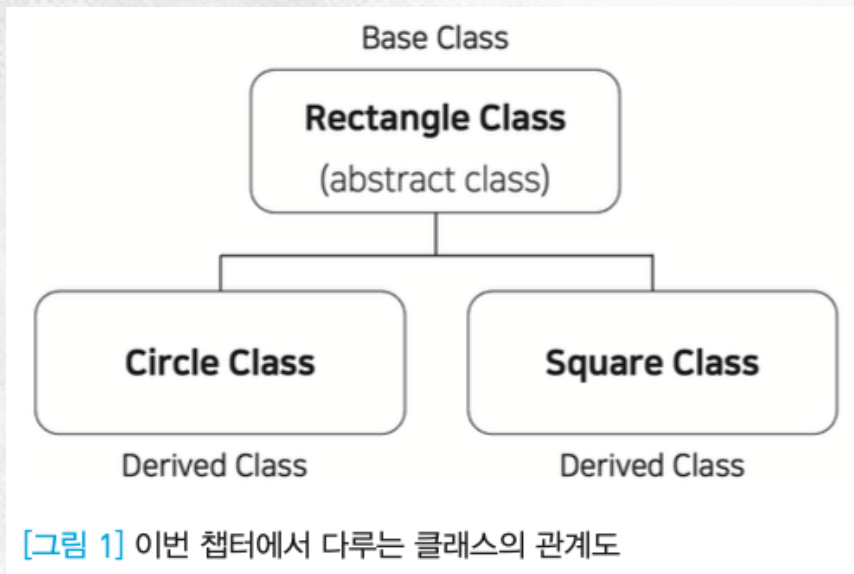
#### NOTE

mixin은 유용한 도구로서 이미 다른 사람들이 만들어 놓은 경우가 많습니다. 프로그램을 만들 때 다른 사람이 개발한 mixin을 사용하는 것도 좋은 방법입니다.

# Class 직접 만들기 Part.3

## ● abstract 클래스

- ✖ 객체를 만들수 없는 Base 클래스
- ✖ Derived 클래스에 공통적인 데이터 및 '메소드 타입' 만 정의함



```
1 abstract class Rectangle {
2     int cx = 0, cy = 0;
3     void draw();
4 }
5
```



## Class 직접 만들기 Part.3

## ● abstract 클래스의 활용

✖ implements 문법을 사용해서 Derived 클래스를 생성함

```
6 class Circle implements Rectangle {
7     @override
8     int cx = 0, cy = 0;
9
10    late int radius
11
12    @override
13    void draw() {
14        print("> Circle.draw(): center($cx,$cy) with r[$radius]");
15    }
16
17    Circle([int givenRadius = 1]) : radius = givenRadius;
18 }
```

## Class 직접 만들기 Part.3 (리뷰 및 실습)

---

- volume-C-chapter-03.dart

## Class 직접 만들기 Part.4

---

- Generic Class (Class Template)
- Static Variable



## Class 직접 만들기 Part.4 (리뷰 및 실습)

---

- volume-C-chapter-04.dart

## 비동기 입출력 기능 활용하기

- 비동기 작업 (asynchronous operation, async)
  - ✘ 두 개 이상의 작업을 동시에 수행함
  - ✘ 동기 작업 (synchronous operation)에 반대되는 개념임
  - ✘ 주로 동작이 느린 외부 저장 장치에 대한 작업을 수행함
  - ✘ 느린 저장 장치는 Stream 클래스로 비동기 동작을 지원함
  - ✘ Main 함수에 의해 수행되는 작업은 foreground, 비동기 함수에 의해 수행되는 작업은 background로 명명함
- Future.delayed( 시간(초), 작업 ) 메소드
  - ✘ '시간'이 지난 후에 '작업'을 수행함
  - ✘ '작업'을 비동기적으로 처리하도록 함
  - ✘ '작업'이 주어지지 않으면, '시간' 만큼 프로그램을 대기 시킴



## 비동기 입출력 기능 활용하기

---

### ● await 문법

- ❖ 비동기 작업이 종료될때까지 인위적으로 대기하도록 함
- ❖ await 문법을 사용하는 함수/메소드는 async 구문을 사용함

### ● Future<> 리턴 타입

- ❖ await 문법을 사용하는 함수/메소드 혹은 Future.delayed() 메소드를 사용하는 함수/메소드가 리턴 값을 갖는 경우, 리턴 값의 타입은 반드시 Future<>로 정의함



## 비동기 입출력 기능 활용하기 (리뷰 및 실습)

---

- volume-C-chapter-05-A.dart
- volume-C-chapter-05-B.dart
- volume-C-chapter-05-C.dart
- volume-C-chapter-05-D.dart
- volume-C-chapter-05-E.dart
- volume-C-chapter-05-F.dart
- volume-C-chapter-05-G.dart
- volume-C-chapter-05-H.dart
- volume-C-chapter-05-Z.dart

## 예외 상황 처리를 통한 안정성 강화하기

---

- try
- on
- catch
- finally
- throw
- rethrow
- Exception (abstract class)



## 예외 상황 처리를 통한 안정성 강화하기 (리뷰 및 실습)

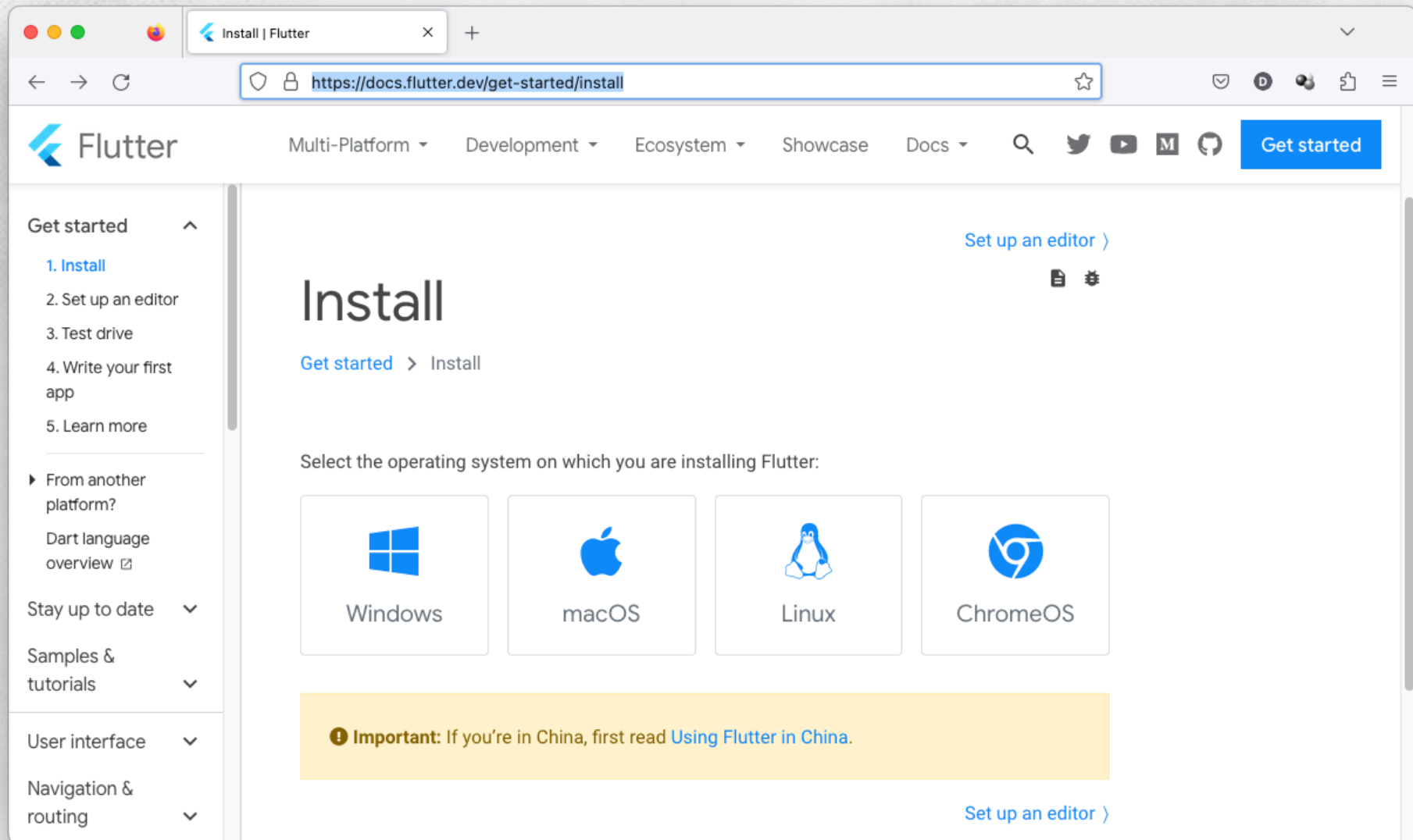
---

- volume-C-chapter-06-A.dart
- volume-C-chapter-06-B.dart
- volume-C-chapter-06-C.dart
- volume-C-chapter-06-D.dart
- volume-C-chapter-06-E.dart
- volume-C-chapter-06-F.dart
- volume-C-chapter-06-G.dart
- volume-C-chapter-06-H.dart
- volume-C-chapter-06-I.dart
- volume-C-chapter-06-J.dart
- volume-C-chapter-06-K.dart
- volume-C-chapter-06-L.dart



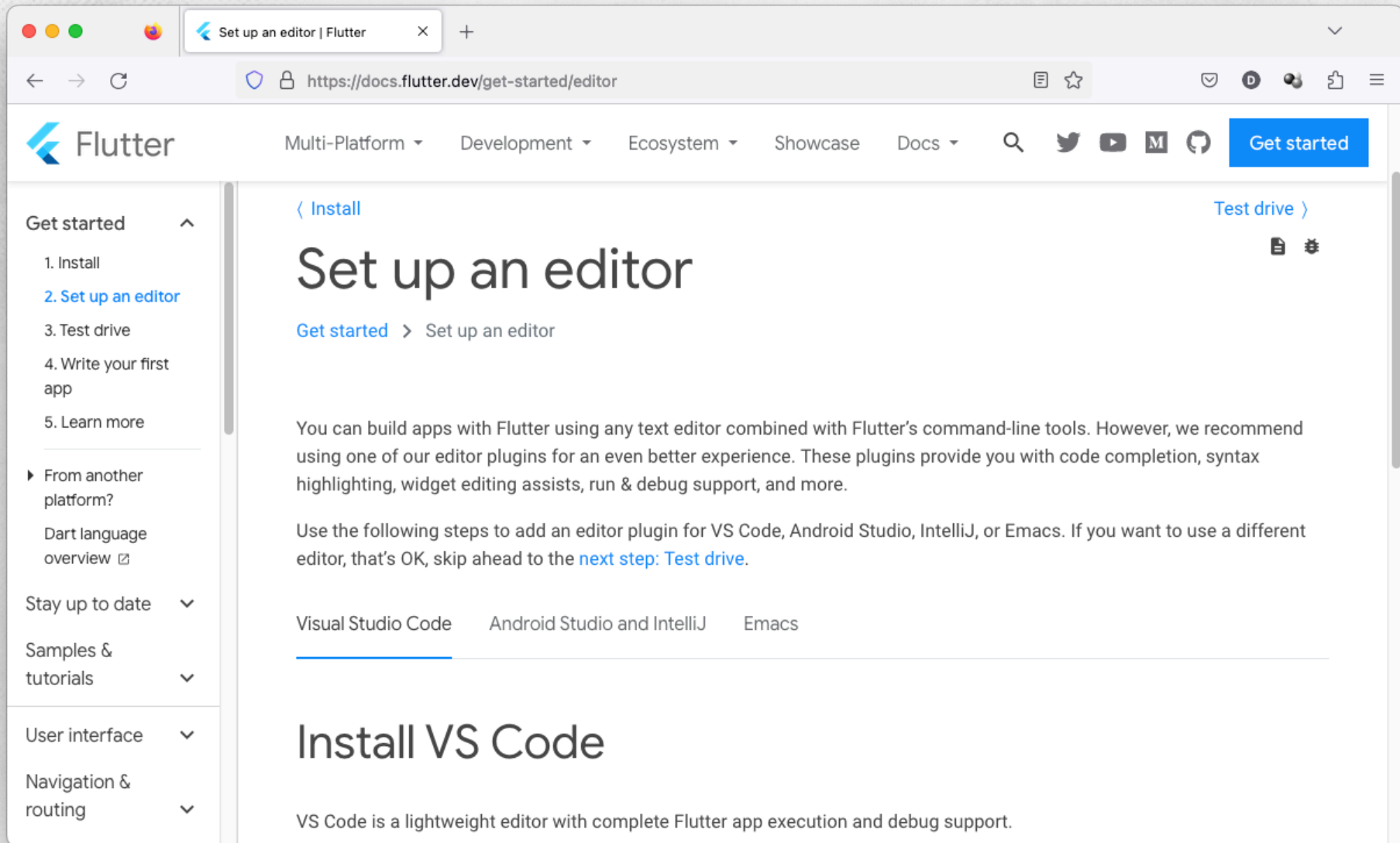
# Dart 개발 환경 설치하기

- Flutter SDK 설치 : <https://docs.flutter.dev/get-started/install>



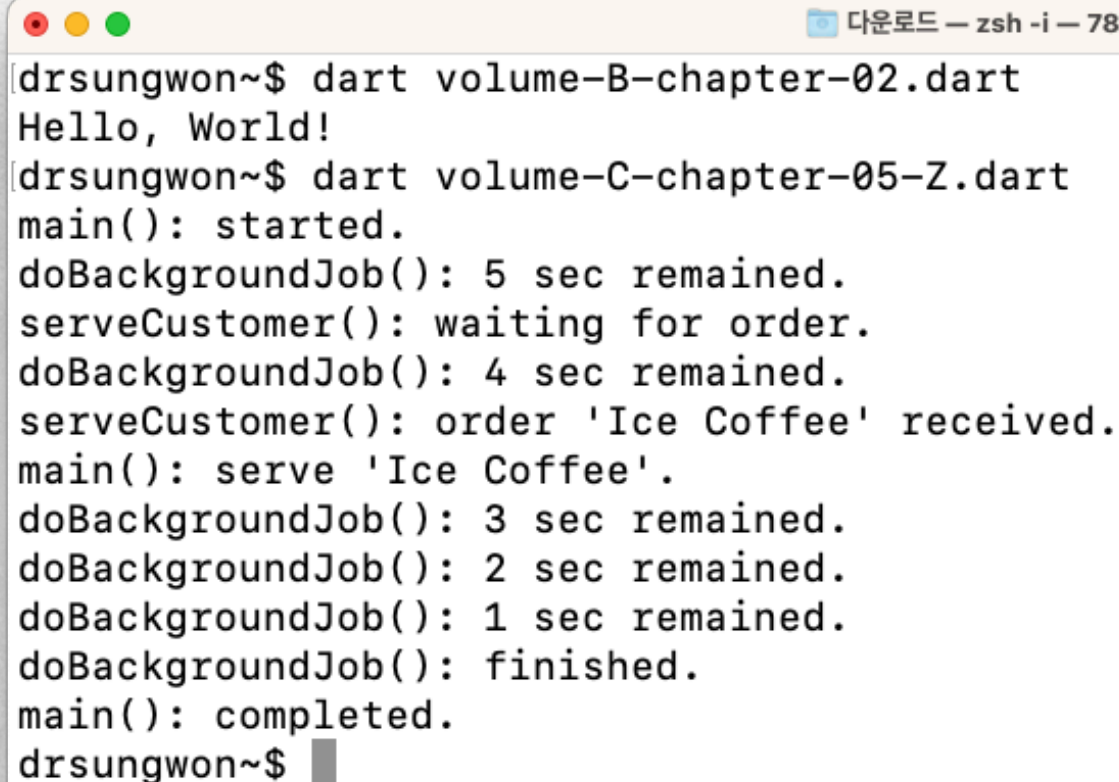
# Dart 개발 환경 설치하기

- Visual Code 설정 : <https://docs.flutter.dev/get-started/editor>



## Dart 개발 환경 설치하기 (리뷰 & 실습)

### ● Flutter SDK 검증 하기

A terminal window titled '다운로드 — zsh -i — 78x17' showing the execution of two Dart programs. The first program, 'dart volume-B-chapter-02.dart', prints 'Hello, World!'. The second program, 'dart volume-C-chapter-05-Z.dart', simulates a coffee shop order system with a main loop and background jobs.

```
drsungwon~$ dart volume-B-chapter-02.dart
Hello, World!
drsungwon~$ dart volume-C-chapter-05-Z.dart
main(): started.
doBackgroundJob(): 5 sec remained.
serveCustomer(): waiting for order.
doBackgroundJob(): 4 sec remained.
serveCustomer(): order 'Ice Coffee' received.
main(): serve 'Ice Coffee'.
doBackgroundJob(): 3 sec remained.
doBackgroundJob(): 2 sec remained.
doBackgroundJob(): 1 sec remained.
doBackgroundJob(): finished.
main(): completed.
drsungwon~$
```



## 키보드 입출력 기능 활용하기

---

- dart:io

- ❖ DartPad에서 실행이 안되며, Native/App/Desktop 지원
- ❖ `import 'dart:io';` 문법으로 활용함
- ❖ <https://api.dart.dev/> 접속 후, dart:io 라이브러리 확인함

- stdin

- `stdin.readLineSync()` : 리턴 값은 String? 타입임

- stdout

- `stdout.write()`

- `stdout.writeln()`

## 키보드 입출력 기능 활용하기

The screenshot shows a web browser window displaying the Dart API documentation for the `dart:io` library. The browser's address bar shows the URL `https://api.dart.dev/stable/3.0.5/dart-io/dart-io-library.html`. The page has a light gray header with a search bar labeled "Search API Docs" and a "Dart" link. The main content area is titled "dart:io library" and includes a description: "File, socket, HTTP, and other I/O support for non-web applications." It also features an "Important" note stating that browser-based apps cannot use this library. A list of supported environments (Servers, Command-line scripts, Flutter mobile apps, Flutter desktop apps) is provided. The page explains that the library allows working with files, directories, sockets, processes, HTTP servers, and clients, and that many operations are asynchronous, using `Futures` or `Streams` from the `dart:async` library. A code snippet shows the import statement: `import 'dart:io';`. A link to the "dart:io library tour" is also present. The right sidebar lists various classes available in the library, including `BytesBuilder`, `CompressionOptions`, `ConnectionTask`, `ContentType`, `Cookie`, `Datagram`, `Directory`, `File`, `FileLock`, `FileMode`, `FileStat`, `FileSystemCreateOptions`, `FileSystemDeleteOptions`, and `FileSystemEntity`. The footer of the page indicates "Dart 3.0.5 • Site CC BY 4.0".

Dart SDK

LIBRARIES

CORE

- [dart:async](#)
- [dart:collection](#)
- [dart:convert](#)
- [dart:core](#)
- [dart:developer](#)
- [dart:math](#)
- [dart:typed\\_data](#)

VM

- [dart:cli](#)
- [dart:ffi](#)
- [dart:io](#)
- [dart:isolate](#)
- [dart:mirrors](#)

## dart:io library

File, socket, HTTP, and other I/O support for non-web applications.

**Important:** Browser-based apps can't use this library. Only the following can import and use the `dart:io` library:

- Servers
- Command-line scripts
- Flutter mobile apps
- Flutter desktop apps

This library allows you to work with files, directories, sockets, processes, HTTP servers and clients, and more. Many operations related to input and output are asynchronous and are handled using [Futures](#) or [Streams](#), both of which are defined in the [dart:async](#) library.

To use the `dart:io` library in your code:

```
import 'dart:io';
```

For an introduction to I/O in Dart, see the [dart:io library tour](#).

### dart:io library

#### CLASSES

- [BytesBuilder](#)
- [CompressionOptions](#)
- [ConnectionTask](#)
- [ContentType](#)
- [Cookie](#)
- [Datagram](#)
- [Directory](#)
- [File](#)
- [FileLock](#)
- [FileMode](#)
- [FileStat](#)
- [FileSystemCreateOptions](#)
- [FileSystemDeleteOptions](#)
- [FileSystemEntity](#)

Dart 3.0.5 • Site CC BY 4.0

## 키보드 입출력 기능 활용하기

---

- volume-C-chapter-08.dart

### ONE MORE : ! 문법

- 변수 이름 뒤에 ! 문법이 있으면, Nullable 변수를 Null-Safety하게 처리함
- 예시) tmpNullable!



## 파일 입출력 기능 활용하기

- dart:io (재확인)
  - ✘ DartPad에서 실행이 안되며, Native/App/Desktop 지원
  - ✘ import 'dart:io'; 문법으로 활용함
  - ✘ <https://api.dart.dev/> 접속 후, dart:io 라이브러리 확인함
- File
- openRead()
- openWrite()
- readAsString()
- write()
- close()
- 참조:
  - ✘ 비동기 작업을 통해서 읽고/쓰는 것이 일반적임
  - ✘ 반복문 & in 문법을 통한 읽기 가능

## 파일 입출력 기능 활용하기

The screenshot shows the Dart API documentation for the `File` class in the `dart:io` library. The browser address bar shows the URL `https://api.dart.dev/stable/3.0.5/dart-io/File-class.html`. The page title is "File class" with labels "abstract" and "interface".

**dart:io library**

**CLASSES**

- BytesBuilder
- CompressionOptions
- ConnectionTask
- ContentType
- Cookie
- Datagram
- Directory
- File
- FileLock
- FileMode
- FileStat
- FileSystemCreateEvent
- FileSystemDeleteEvent
- FileSystemEntity

**File class** abstract interface

A reference to a file on the file system.

A `File` holds a `path` on which operations can be performed. You can get the parent directory of the file using `parent`, a property inherited from `FileSystemEntity`.

Create a new `File` object with a pathname to access the specified file on the file system from your program.

```
var myFile = File('file.txt');
```

The `File` class contains methods for manipulating files and their contents. Using methods in this class, you can open and close files, read to and write from them, create and delete them, and check for their existence.

When reading or writing a file, you can use streams (with `openRead`), random access operations (with `open`), or convenience methods such as `readAsString`,

Most methods in this class occur in synchronous and asynchronous pairs, for example, `readAsString` and `readAsStringSync`. Unless you have a specific reason for using the

**CONSTRUCT...**

- File
- fromRawPath
- fromUri

**PROPERTIES**

- absolute
- hashCode
- isAbsolute
- parent
- path
- runtimeType
- uri

**METHODS**

- copy
- copySync

Dart 3.0.5 • Site CC BY 4.0

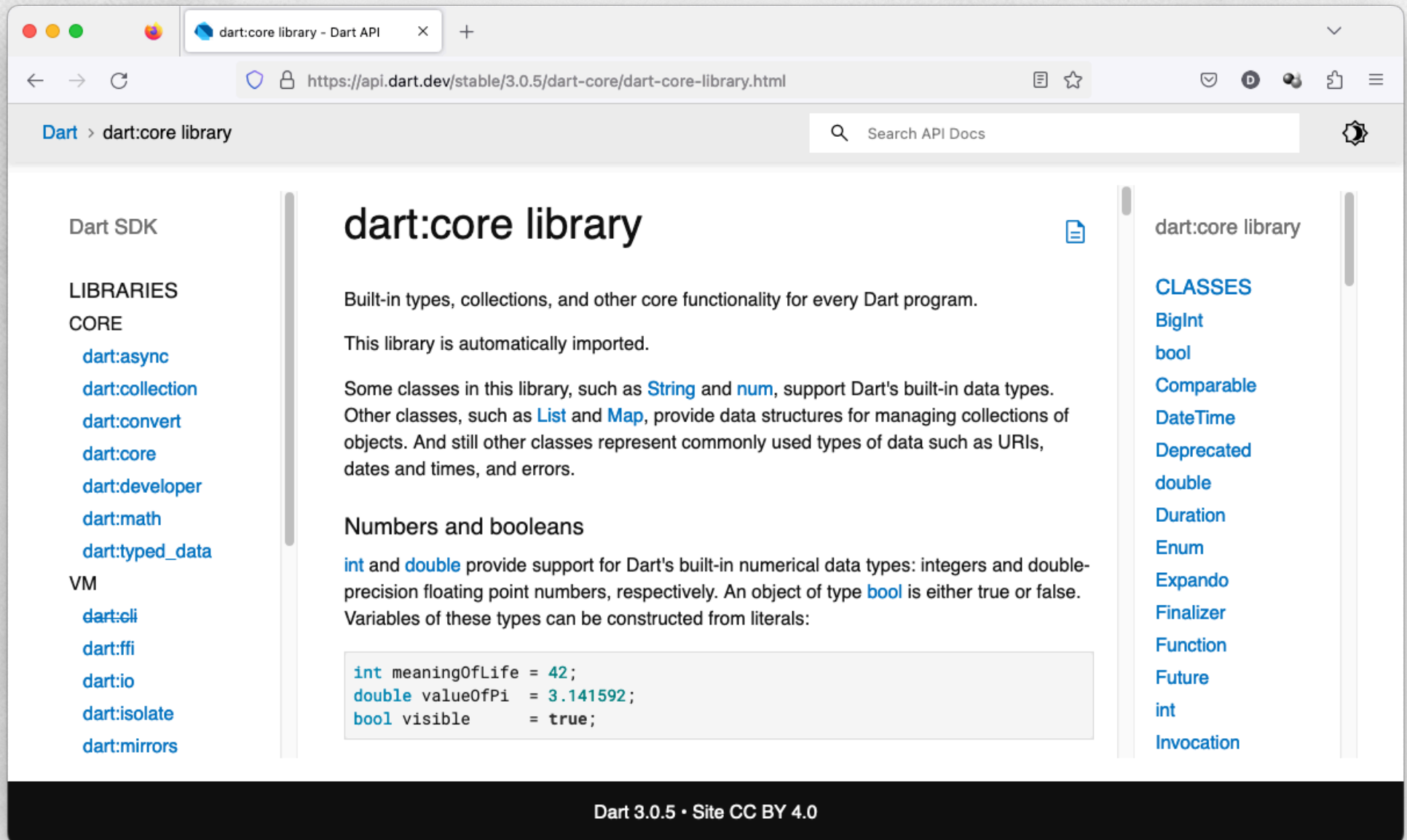


## 파일 입출력 기능 활용하기

---

- volume-C-chapter-09.dart





The screenshot shows a web browser window displaying the Dart API documentation for the `dart:async` library. The browser's address bar shows the URL `https://api.dart.dev/stable/3.0.5/dart-async/dart-async-library.html`. The page title is "dart:async library".

**Left Sidebar:**

- Dart SDK
- LIBRARIES
- CORE
  - [dart:async](#)
  - [dart:collection](#)
  - [dart:convert](#)
  - [dart:core](#)
  - [dart:developer](#)
  - [dart:math](#)
  - [dart:typed\\_data](#)
- VM
  - [dart:cli](#)
  - [dart:ffi](#)
  - [dart:io](#)
  - [dart:isolate](#)
  - [dart:mirrors](#)

**Main Content:**

## dart:async library

Support for asynchronous programming, with classes such as `Future` and `Stream`.

`Futures` and `Streams` are the fundamental building blocks of asynchronous programming in Dart. They are supported directly in the language through `async` and `async*` functions, and are available to all libraries through the `dart:core` library.

This library provides further tools for creating, consuming and transforming futures and streams, as well as direct access to other asynchronous primitives like `Timer`, `scheduleMicrotask` and `Zone`.

To use this library in your code:

```
import 'dart:async';
```

**Future**

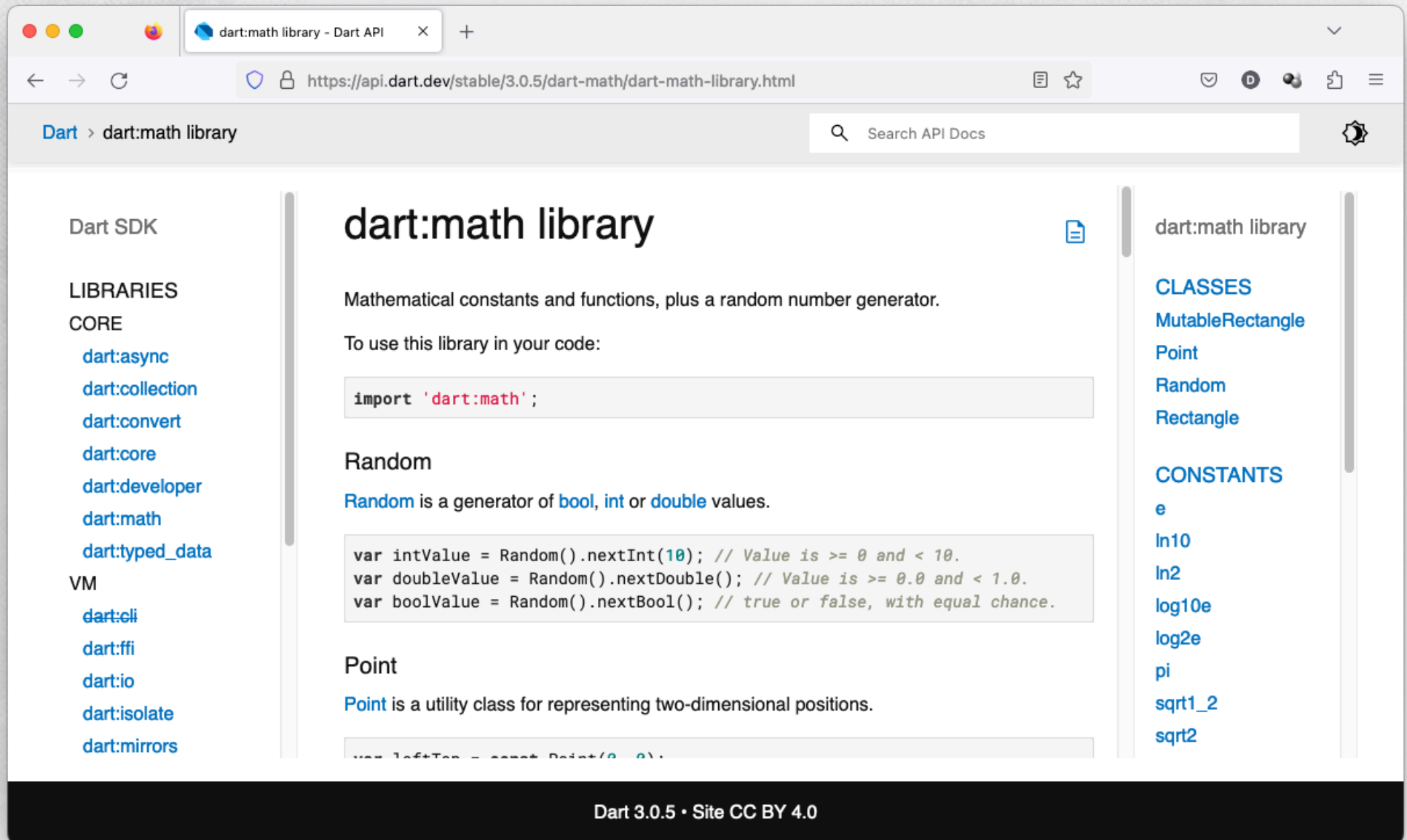
A `Future` object represents a computation whose return value might not yet be available. The `Future` returns the value of the computation when it completes at some time in the future. Futures are often used for APIs that are implemented using a different thread or

**Right Sidebar:**

- dart:async library
- CLASSES
  - [Completer](#)
  - [EventSink](#)
  - [Future](#)
  - [FutureOr](#)
  - [MultiStreamContr...](#)
  - [Stream](#)
  - [StreamConsumer](#)
  - [StreamController](#)
  - [StreamIterator](#)
  - [StreamSink](#)
  - [StreamSubscription](#)
  - [StreamTransformer](#)
  - [StreamTransform...](#)
  - [StreamView](#)

**Footer:** Dart 3.0.5 • Site CC BY 4.0





## Core 라이브러리

---

- `dart:async` : Support for asynchronous programming, with classes such as `Future` and `Stream`
- `dart:collection` : Classes and utilities that supplement the collection support in `dart:core`
- `dart:convert` : Encoders and decoders for converting between different data representations, including JSON and UTF-8
- `dart:core` : Built-in types, collections, and other core functionality for every Dart program
- `dart:developer` : Interact with developer tools such as the debugger and inspector
- `dart:math` : Mathematical constants and functions, plus a random number generator
- `dart:typed_data` : Lists that efficiently handle fixed sized data (for example, unsigned 8 byte integers) and SIMD numeric types



- dart:ffi : Foreign Function Interface for interoperability with the C programming language
- dart:io : File, socket, HTTP, and other I/O support for non-web applications
- dart:isolate : Concurrent programming using isolates: independent workers that are similar to threads but don't share memory, communicating only via messages
- dart:mirrors : Basic reflection in Dart, with support for introspection and dynamic invocation



## Web 라이브러리

---

- `dart:html` : HTML elements and other resources for web-based applications that need to interact with the browser and the DOM (Document Object Model)
- `dart:indexed_db` : A client-side key-value store with support for indexes
- `dart:js` : Low-level support for interoperating with JavaScript
- `dart:js_util` : Utility methods to manipulate `package:js` annotated JavaScript interop objects in cases where the name to call is not known at runtime
- `dart:svg` : Scalable Vector Graphics: Two-dimensional vector graphics with support for events and animation
- `dart:web_audio` : High-fidelity audio programming in the browser
- `dart:web_gl` : 3D programming in the browser

## 표준 라이브러리 활용하기

---

- volume-C-chapter-10.dart





**Thank you**



**경희대학교**  
KYUNG HEE UNIVERSITY