

Before

```

@immutable
class Person {
  const Person({
    required this.firstName,
    required this.lastName,
    required this.age,
  });

  factory Person.fromJson(Map<String, Object?> json) {
    return Person(
      firstName: json['firstName'] as String,
      lastName: json['lastName'] as String,
      age: json['age'] as int,
    );
  }

  final String firstName;
  final String lastName;
  final int age;

  Person copyWith({
    String? firstName,
    String? lastName,
    int? age,
  }) {
    return Person(
      firstName: firstName,
      lastName: lastName,
      age: age,
    );
  }

  Map<String, Object?> toJson() {
    return {
      'firstName': firstName,
      'lastName': lastName,
      'age': age,
    };
  }

  @override
  String toString() {
    return 'Person('
      'firstName: $firstName, '
      'lastName: $lastName, '
      'age: $age'
    ')';
  }

  @override
  bool operator ==(Object other) {
    return other is Person &&
      person.runtimeType == runtimeType &&
      person.firstName == firstName &&
      person.lastName == lastName &&
      person.age == age;
  }

  @override
  int get hashCode {
    return Object.hash(
      runtimeType,
      firstName,
      lastName,
      age,
    );
  }
}

```

After

```

@freezed
class Person with _$Person {
  const factory Person({
    required String firstName,
    required String lastName,
    required int age,
  }) = _Person;

  factory Person.fromJson(Map<String, Object?> json)
    => _$PersonFromJson(json);
}

```

출처 - <https://pub.dev/packages/freezed>

## ▼ 실습 준비



다음 패키지들을 설치해 봅시다.

- build\_runner : 코드 생성
- freezed & freezed\_annotation : `==`, `hashCode`, `copyWith()`, `toString()`
- json\_serializable & json\_annotation : `fromJson()`, `toJson()`

1. Terminal을 열고 다음 명령어를 실행하여 필요한 패키지들을 설치해 주세요.

▼ 코드스니펫 - **터미널** : dev\_dependency 설치

```
dart pub add -d freezed build_runner json_serializable
```



freezed & build\_runner & json\_serializable 패키지는 배포 전 개발 환경에서만 필요하므로 dev\_dependency에 추가하도록 `-d` 옵션을 붙였습니다.

▼ 코드스니펫 - **터미널** : dependency 설치

```
dart pub add freezed_annotation json_annotation
```



freezed\_annotation & json\_annotation 패키지는 배포 환경에서 사용하므로 dependency에 추가하였습니다.



Dart 프로젝트가 아닌 Flutter 프로젝트라면 `flutter pub add <패키지명>` 으로 설치를 진행하면 됩니다.

```
! analysis_options.yaml
🕒 CHANGELOG.md
☰ pubspec.lock M
! pubspec.yaml M
📘 README.md

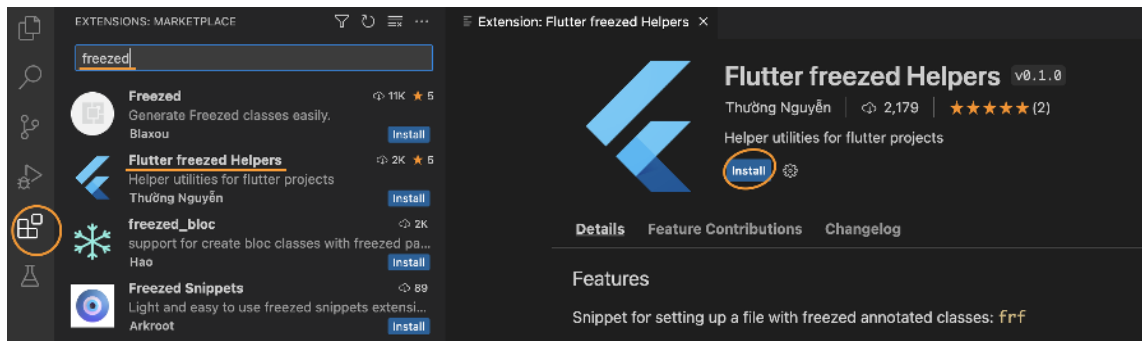
12 dev_dependencies:
13   build_runner: ^2.3.3
14   freezed: ^2.3.2
15   json_serializable: ^6.5.4
16   lints: ^2.0.0
17   test: ^1.16.0
18 dependencies:
19   equatable: ^2.0.5
20   freezed_annotation: ^2.2.0
21   json_annotation: ^4.7.0
22
```

설치 명령어를 실행하는 시점에 따라 버전이 다를 수 있습니다.



다음으로, 코드 생성기를 쓸 때 함께 사용하면 유용한 VSCode 확장 프로그램을 설치해 봅시다.

2. VSCode Extensions을 열고 `freezed` 라고 검색한 뒤 `Flutter freezed Helpers` 를 설치해 주세요.






**Flutter frozen Helpers** 는 frozen 코드를 쉽게 작성하도록 도와주는 확장 프로그램입니다.

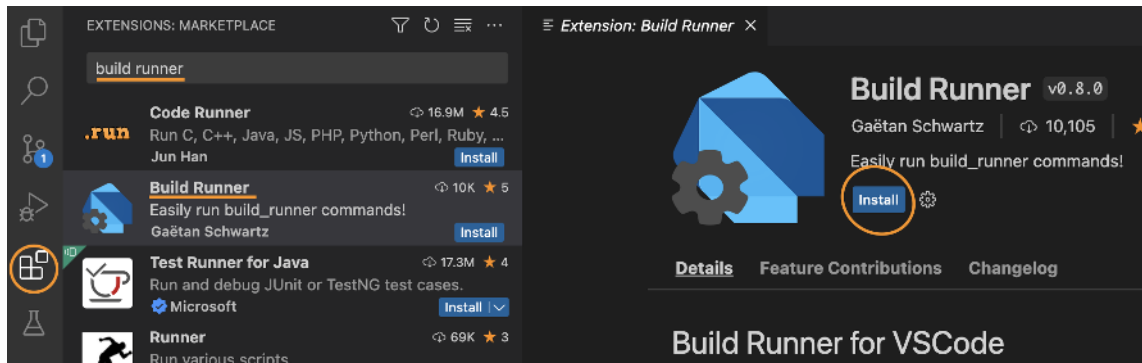
#### Flutter frozen Helpers - Visual Studio Marketplace

This extension currently helps you to easily write frozen annotated classes and allows you to run code generation for those classes. You can also watch the files so that

 <https://marketplace.visualstudio.com/items?itemName=mthuong.vscode-flutter-frozen-helper>




3. **build runner** 라고 검색해 주세요. 그리고 아래 이미지와 같이 Dart 로고가 있는 **Build Runner** 를 선택하여 설치해 주세요.



**Build Runner** 는 코드가 변경된 경우 자동으로 코드를 생성하도록 도와주는 확장 프로그램입니다.

#### Build Runner - Visual Studio Marketplace

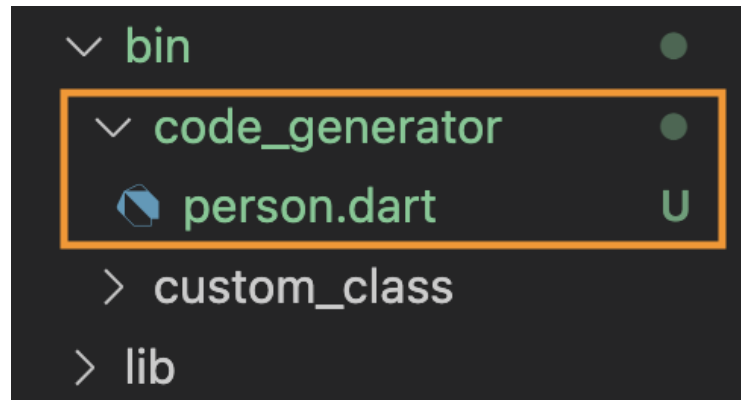
Launch build\_runner build with CTRL+SHIFT+B (CMD+SHIFT+B on Mac). Launch build\_runner build with --build-filter by using CTRL+ALT+B (

 <https://marketplace.visualstudio.com/items?itemName=GaetanSchwartz.build-runner>



### ▼ 실습

1. **bin** 폴더 밑에 **code\_generator** 폴더를 만들고, 밑에 **person.dart** 파일을 만들어 주세요.





`person.dart` 파일에 아래와 같이 Person 클래스를 코드 생성 패키지들을 이용해 만들어 봅시다.

```

class Person {
  final String name;
  final int age;

  const Person({
    required this.name,
    required this.age,
  });

  Person copyWith({
    String? name,
    int? age,
  }) {
    return Person(
      name: name ?? this.name,
      age: age ?? this.age,
    );
  }

  factory Person.fromJson(Map<String, dynamic> json) {
    return Person(
      name: json['name'] ?? '',
      age: json['age'] ?? 0,
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'name': name,
      'age': age,
    };
  }

  @override
  bool operator ==(Object other) =>
    identical(this, other) ||
    other is Person &&
    runtimeType == other.runtimeType &&
    name == other.name &&
    age == other.age;

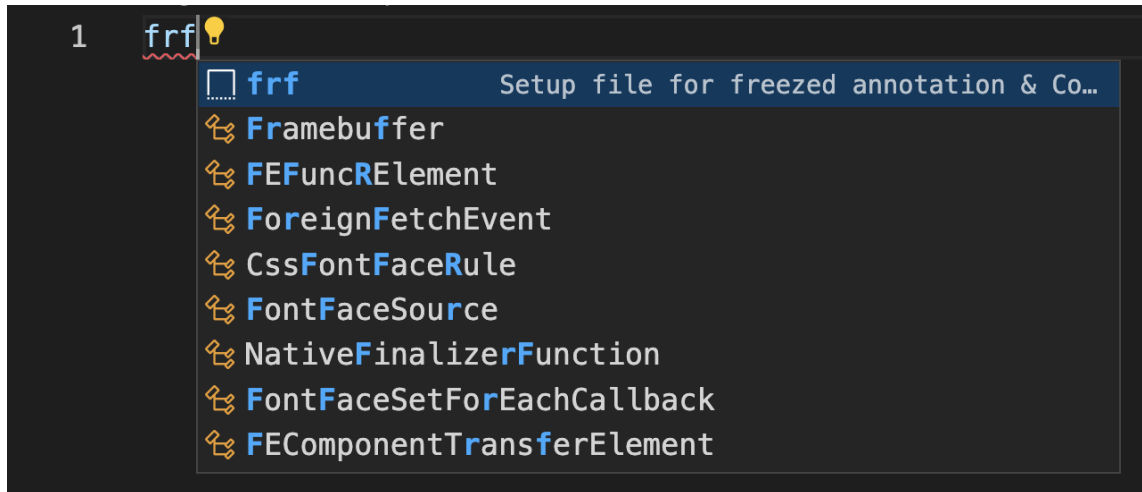
  @override
  int get hashCode => Object.hash(
    name,
    age,
  );

  @override
  String toString() {
    return 'Person(name: $name, age: $age)';
  }
}

```

<https://dartpad.dev/?id=6dbf0a9a1d97674ec72450e13bfe287a>

2. 앞서 설치한 Flutter freezed helpers 패키지를 이용해 freezed 자동 완성을 이용해 봅시다. `person.dart` 파일에 `frf` 라고 작성한 뒤, 아래와 같이 자동 완성이 출력되면 엔터를 눌러주세요.



그러면 아래 이미지와 같이 작성되고, 동일한 이름이 들어가야하는 자리에 커서가 깜빡거립니다.



3. `Person` 이라고 클래스 이름을 입력한 뒤, `ESC` 를 누르면 커서가 1개로 되돌아옵니다.



```

1 import 'package:freezed_annotation/freezed_annotation.dart';
2
3 part 'person.freezed.dart';
4 part 'person.g.dart';
5
6 @freezed
7 class Person with _$Person {
8
9   factory Person() = _Person;
10
11   factory Person.fromJson(Map<String, dynamic> json) => _$PersonFromJson(json);
12 }

```

4. 아래 이미지와 같이 `factory Person()` 생성자에 다음과 같이 이름 지정 매개변수로 `String name` 을 필수로 받도록 작성해 주세요.

▼ 코드스니펫 - `person.dart` : String name 속성 추가

```
{required String name}
```

```

1 import 'package:freezed_annotation/freezed_annotation.dart';
2
3 part 'person.freezed.dart';
4 part 'person.g.dart';
5
6 @freezed
7 class Person with _$Person {
8   factory Person({required String name}) = _Person;
9
10   factory Person.fromJson(Map<String, dynamic> json) => _$PersonFromJson(json);
11 }
12

```



위와 같이 생성자에 `String name` 매개변수를 작성하면, 코드 생성시 `String name;` 속성을 갖도록 만들어 줍니다.



아직 코드 생성 명령을 실행하지 않았기 때문에 에러가 발생합니다.

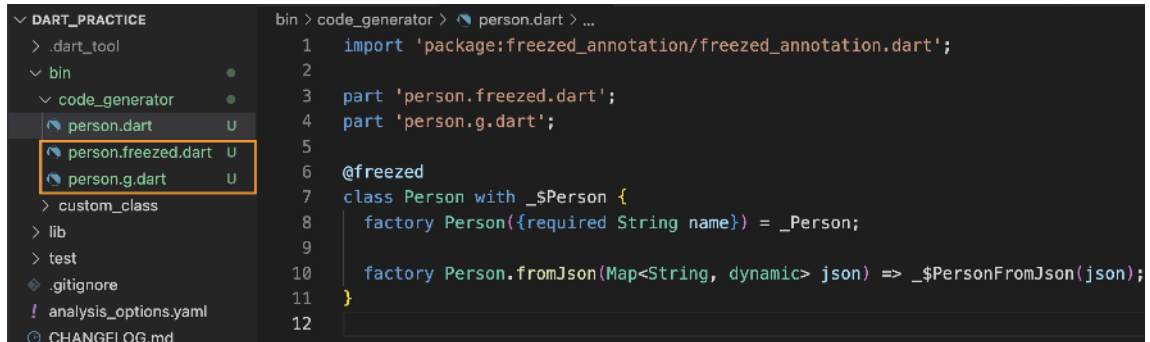
5. Terminal에서 다음 명령어를 실행하여 코드를 생성해 주세요.

▼ 코드스니펫 - `터미널` : 코드 생성 명령어

```
dart run build_runner build
```



Flutter 프로젝트인 경우 `flutter run build_runner build` 명령어를 실행하면 됩니다.



코드 생성 명령이 실행되면, `person.dart` 파일 밑에 `person.freezed.dart` 와 `person.g.dart` 파일이 생성됩니다.

- `person.freezed.dart` : `==`, `hashCode`, `copyWith()`, `toString()` 코드 생성
- `person.g.dart` : `fromJson()`, `toJson()` 코드 생성



코드 생성기로 생성된 파일은 수정하지 않고, `person.dart` 파일을 수정한 뒤 다시 코드 생성 명령어를 실행하여 파일을 재생성하는 방식으로 진행합니다.

6. `person.dart` 파일에 생성자에 `int age` 속성을 추가해 봅시다.

▼ 코드스니펫 - `person.dart` : `int age` 속성 추가

```
required int age,
```

```

1  import 'package:freezed_annotation/freezed_annotation.dart';
2
3  part 'person.freezed.dart';
4  part 'person.g.dart';
5
6  @freezed
7  class Person with _$Person {
8    factory Person({
9      required String name,
10     required int age,
11   }) = _Person;
12
13   factory Person.fromJson(Map<String, dynamic> json) => _Person
14 }
15

```

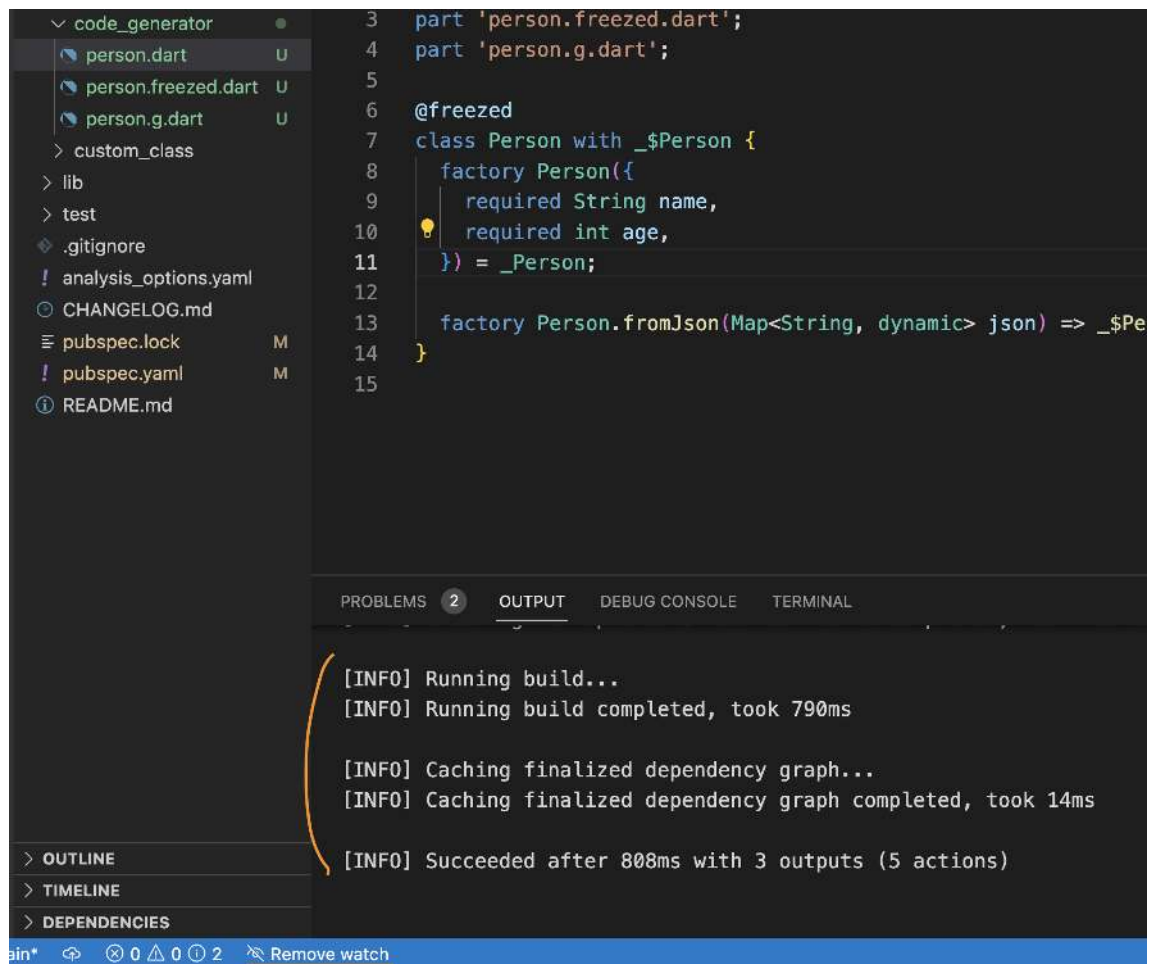


Terminal에서 `dart run build_runner build` 명령어 실행하면 다시 코드가 생성 되는데, 앞서 설치한 VSCode Build Runner 확장 프로그램을 이용해 코드 수정시 자동으로 위 명령어를 실행하도록 만들어 봅시다.

7. VSCode 좌측 하단에 `Watch` 버튼을 클릭해 주세요.



그러면 아래 이미지와 같이 `Remove watch` 로 변경되고, build가 실행된 것을 확인할 수 있습니다.



The screenshot shows the VS Code editor with a Dart file named `person.dart` open. The file contains the following code:

```
3 part 'person.freezed.dart';
4 part 'person.g.dart';
5
6 @freezed
7 class Person with _$Person {
8   factory Person({
9     required String name,
10    required int age,
11  }) = _Person;
12
13   factory Person.fromJson(Map<String, dynamic> json) => _$Pe
14 }
15
```

The Output console at the bottom shows the following logs:

```
[INFO] Running build...
[INFO] Running build completed, took 790ms

[INFO] Caching finalized dependency graph...
[INFO] Caching finalized dependency graph completed, took 14ms

[INFO] Succeeded after 808ms with 3 outputs (5 actions)
```

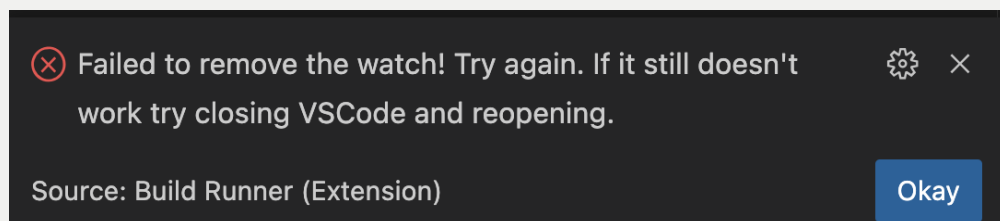
A yellow bracket highlights the logs from `[INFO] Running build...` to `[INFO] Succeeded after 808ms...`. The status bar at the bottom shows `ain*` and `Remove watch`.



`watch` 중인 경우에는 터미널에 직접 코드 생성 명령어를 내리지 않아도, 수정시 자동으로 코드 생성이 됩니다.



`Remove watch` 를 클릭하였을 때, 아래와 같이 에러가 뜬다면 VSCode를 재시작 하여 `watch` 를 중지할 수 있습니다.



8. `code_generator` 폴더 밑에 `main.dart` 파일을 만들고 정말로 앞서 살펴본 함수들이 만들어져 있는지 확인해 봅시다.

▼ 코드스니펫 - `main.dart` : 시작 코드

```
import 'dart:convert';

import 'person.dart';

void main() {
  Person a = Person(name: '철수', age: 10);
  Person b = a;

  // a.name = '영희'; // 불변 객체이므로 name 수정 불가

  /// copyWith()
  a = a.copyWith(name: '영희');

  /// toString()
  print("toString() : $a");
  print("toString() : $b");

  /// 값 비교(Value Equality)
  bool valueEquality = Person(name: "철수", age: 1) == Person(name: "철수", age: 1);
  print("값 비교 : $valueEquality");

  /// JSON 직렬화(Serialization)
  Map<String, dynamic> map = a.toJson();
  print("toJson() : $map");
  String jsonString = jsonEncode(map);

  /// JSON 역직렬화(Deserialization)
  Map<String, dynamic> jsonMap = jsonDecode(jsonString);
  Person person = Person.fromJson(jsonMap);
  print("fromJson() : $person");
}
```

```

1  import 'dart:convert';
2
3  import 'person.dart';
4
5  Run | Debug
6  void main() {
7      Person a = Person(name: '철수', age: 10);
8      Person b = a;
9
10     // a.name = '영희'; // 불변 객체이므로 name 수정 불가
11
12     /// copyWith()
13     a = a.copyWith(name: '영희');
14
15     /// toString()
16     print("toString() : $a");
17     print("toString() : $b");
18
19     /// 값 비교(Value Equality)
20     bool valueEquality = Person(name: "철수", age: 1) == Person(name: "철수", age: 1);
21     print("값 비교 : $valueEquality");
22
23     /// JSON 직렬화(Serialization)
24     Map<String, dynamic> map = a.toJson();
25     print("toJson() : $map");
26     String jsonString = jsonEncode(map);
27
28     /// JSON 역직렬화(Deserialization)
29     Map<String, dynamic> jsonMap = jsonDecode(jsonString);
30     Person person = Person.fromJson(jsonMap);
31     print("fromJson() : $person");
32 }

```



코드 생성기는 객체를 다음과 같이 생성합니다.


- Line 9 : 불변 객체로 생성하기 때문에, 객체의 속성을 수정할 수 없습니다.
- Line 12 : `copyWith()` 를 생성합니다.
- Line 15, 16 : 로깅시 객체 속성을 볼 수 있도록 `toString()` 을 생성합니다.
- Line 19 : 값 비교를 위해 `==`, `hashCode` 를 생성합니다.
- Line 23 : `toJson()` 를 생성합니다.
- Line 29 : `fromJson()` 을 생성합니다.



보다 자세한 사용 방법은 공식 문서를 참고해 주세요.

#### freezed | Dart Package


English | 한국어 Welcome to Freezed, yet another code generator for data-classes/unions/pattern-matching/cloning. Dart is awesome, but defining a

 <https://pub.dev/packages/freezed>



#### json\_serializable | Dart Package

Provides Dart Build System builders for handling JSON. The builders generate code when they find members annotated with classes defined in

 [https://pub.dev/packages/json\\_serializable](https://pub.dev/packages/json_serializable)



**Run** 버튼을 눌러 실행하면 다음과 같이 출력 됩니다.

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
toString() : Person(name: 영화, age: 10)
toString() : Person(name: 철수, age: 10)
값 비교 : true
toJson() : {name: 영화, age: 10}
fromJson() : Person(name: 영화, age: 10)
Exited
```



여기까지 코드 생성기 패키지가 어떤 코드를 생성해 주고, 어떤 이유로 해당 코드들을 사용해야 하는지 배워보았습니다.

## 06. Dart 키워드

▼ final & const

### final & const 공통점



### 변수 재할당 방지 기능

```
final a = 1;  
a = 2; // 불가능  
  
const b = 1;  
b = 2 ; // 불가능
```

## final & const 차이점



### 용도 차이

- **final** : 변수의 참조 재할당을 방지 역할
- **const** : **컴파일 타임**에 계산될 수 있는 상수를 선언하는 역할

```
1 ▼ void main() {  
2   final a = DateTime.now(); // 가능  
3  
4   const b = DateTime.now(); // 불가능 : DateTime.now()는 런타임에 결정됨  
5 }  
6
```

<https://dartpad.dev/?id=c46bc591121cbf0484796c2d0ddb1595>





### late 키워드 사용 가능 여부

`late` 는 변수 초기값을 나중에 할당하겠다는 의미의 키워드입니다.

- **final**: `late` 키워드 사용 가능
- **const**: `late` 키워드 사용 불가능 (**컴파일 타임**에 선언되므로 나중에 없음)

```
void main() {  
  late final a; // late 가능  
  a = 1;  
  
  late const b; // late 불가능  
  b = 1;  
}
```

<https://dartpad.dev/?id=5d7ca168ceb481ee88954ceabafa81d3>



## 값 변경 여부

- **final** : 값을 변경할 수 있습니다.
- **const** : 값을 변경할 수 없습니다.

```
void main() {
  final a = [];
  a.add(1); // 값 변경 가능
  print(a); // [1]

  const b = [];
  b.add(1); // 런타임 에러 : 값 변경 불가능
  print(b);
}
```

Console

```
[1]
Uncaught Error: Unsupported operation: add
```

<https://dartpad.dev/?id=13bf01d2a8a45c53a6749b6d8b1822a8>

**final**은 변수 재할당 방지만 담당할 뿐, 객체 생성과 관련이 없기 때문에 다음과 같은 결과가 나옵니다.

```
1 void main() {
2   final a1 = [];
3   final a2 = [];
4   print(a1 == a2); // false (메모리 주소 다름, 가변 배열)
5
6   const b1 = [];
7   const b2 = [];
8   print(b1 == b2); // true (메모리 주소 동일, 불변 배열)
9
10  final c1 = const [];
11  final c2 = const [];
12  print(c1 == c2); // true (메모리 주소 동일, 불변 배열)
13 }
```

<https://dartpad.dev/?id=1add1fc45cdf650cceb1d30cdf736dcb>

따라서 **final**은 네 개의 각기 다른 배열을 만들고, **const**는 값이 동일하기 때문에 배열을 하나만 만든 뒤 참조를 공유합니다.

```
final a1 = [];
final a2 = [];
final a3 = [];
final a4 = [];
```

```
const b1 = [];  
const b2 = [];  
const b3 = [];  
const b4 = [];
```

## ▼ getter & setter



VSCode에서 Getter & Setter를 배워봅시다.

1. `bin` 폴더 밑에 `getter_setter` 폴더를 만들고 `document.dart` 와 `main.dart` 파일을 만들어 주세요.

### ▼ 코드스니펫 - `document.dart` : 시작 코드

```
class Document {  
  // 내용  
  String content;  
  
  // 읽은 횟수  
  int readCount = 0;  
  
  // 수정 횟수  
  int updateCount = 0;  
  
  // 통계  
  late String statistic = "readCount : $readCount / updat  
  
  Document(this.content);  
}
```

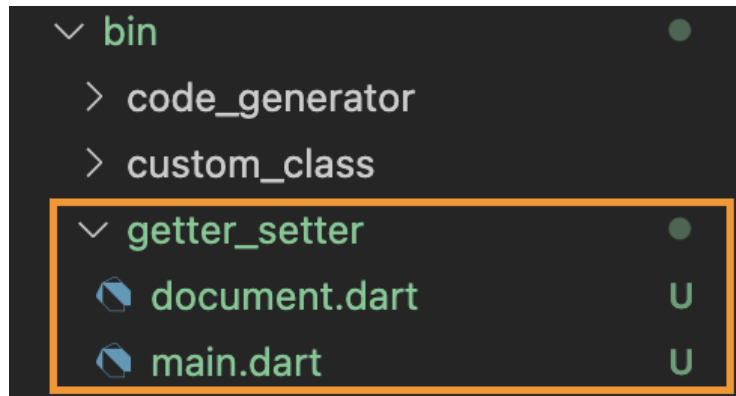
### ▼ 코드스니펫 - `main.dart` : 시작 코드

```
import 'document.dart';  
  
void main() {
```

```
final doc = Document("1");

// 조회
String content = doc.content;
doc.readCount += 1;
print(doc.statistic);

// 수정
doc.content = "2";
doc.updateCount += 1;
print(doc.statistic);
}
```





#### document.dart 파일

- `content` : 문서 내용
- `readCount` : `content` 를 읽은 횟수
- `updateCount` : `content` 를 수정한 횟수
- `statistic` : `readCount` & `updateCount` 출력 문자열

```
bin > getter_setter > document.dart > ...
1  class Document {
2      // 내용
3      String content;
4
5      // 읽은 횟수
6      int readCount = 0;
7
8      // 수정 횟수
9      int updateCount = 0;
10
11     // 통계
12     late String statistic = "readCount : $readCount / updateCount : $updateCount";
13
14     Document(this.content);
15 }
16
```

`statistic` 같이 다른 속성을 초기값으로 사용하려면 앞에 `late` 키워드를 붙여야 합니다. `late` 키워드가 붙은 변수는 해당 변수를 처음 호출할 때 값 할당이 이뤄집니다.



#### main.dart 파일

```
bin > getter_setter > main.dart > ...  
1  import 'document.dart';  
2  
Run | Debug  
3  void main() {  
4      final doc = Document("1");  
5  
6      // 조회  
7      String content = doc.content;  
8      doc.readCount += 1;  
9      print(doc.statistic);  
10  
11     // 수정  
12     doc.content = "2";  
13     doc.updateCount += 1;  
14     print(doc.statistic);  
15 }  
16
```

- 7번째 줄에서 `doc.content` 를 조회했기 때문에, 8번째 줄에서 `readCount` 를 1 증가시킵니다.
- 12번째 줄에서 `doc.content` 를 수정했기 때문에, 13번째 줄에서 `updateCount` 를 1 증가시킵니다.

2. `getter_setter/main.dart` 파일을 열고 `main()` 함수 위의 `Run` 버튼을 실행하면 다음과 같이 나옵니다.

```
bin > getter_setter > main.dart > ...
1  import 'document.dart';
2
   Run | Debug
3  void main() {
4      final doc = Document("1");
5
6      // 조회
7      String content = doc.content;
8      doc.readCount += 1;
9      print(doc.statistic);
10
11     // 수정
12     doc.content = "2";
13     doc.updateCount += 1;
14     print(doc.statistic);
15 }
16
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

2 readCount : 1 / updateCount : 0  
Exited



Debug Console에 `readCount : 1 / updateCount : 0` 만 두 번 출력됩니다. 위와 같은 결과가 출력된 이유는 `late` 키워드로 인해 `statistic` 변수에 처음 접근하는 9번째 줄에서 `statistic` 의 값이 할당되고(늦은 초기화), 이후 14번째 줄에서 접근하는 경우 이미 할당된 값을 반환하기 때문입니다.

```
// 읽은 횟수
int readCount = 0;

// 수정 횟수
int updateCount = 0;

// 통계
late String statistic = "readCount : $readCount / updateCount : $updateCount";
```

`statistic` 속성에 접근할 때 마다 매번 `readCount` 와 `updateCount` 값을 가져오도록 수정해 봅시다.

3. `document.dart` 파일의 `statistic` 을 다음과 같이 수정해 주세요.

▼ 코드스니펫 - `document.dart` : statistic Getter

```
String get statistic => "readCount : $readCount / updateC
```

```
11 // 통계
12 String get statistic => "readCount : $readCount / updateCount : $updateCount";
13
```





`get` 키워드를 사용하는 함수를 Getter라고 부릅니다.

```
// 일반 형태
반환타입 get 이름 {
    return 반환값;
}

// 화살표 구문 사용
반환타입 get 이름 => 반환값;
```

변수와 같이 이름을 불러 접근할 수 있습니다.

이름

함수를 호출하면 내부를 실행하듯, Getter도 호출 시 내부 로직을 매번 실행합니다.

4. `main.dart` 파일을 다시 실행하면 `statistic` Getter가 매번 값을 가져오는 것을 확인할 수 있습니다.

```
bin > getter_setter > main.dart > ...
1  import 'document.dart';
2
Run | Debug
3  void main() {
4      final doc = Document("1");
5
6      // 조회
7      String content = doc.content;
8      doc.readCount += 1;
9      print(doc.statistic);
10
11     // 수정
12     doc.content = "2";
13     doc.updateCount += 1;
14     print(doc.statistic);
15 }
16
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

```
readCount : 1 / updateCount : 0
readCount : 1 / updateCount : 1
Exited
```



위 코드는 다음과 같은 의존 관계를 가지고 있습니다.

- `content`에 접근하는 경우, 항상 `readCount += 1`을 해줘야 합니다.
- `content`를 수정하는 경우, 항상 `updateCount += 1`을 해줘야 합니다.

누군가 실수로 count를 갱신하지 않는다면, 데이터가 신뢰성을 잃는 문제가 있습니다.



Getter & Setter를 이용하면 보다 방어적인 코드를 작성해 인적 오류를 방지할 수 있습니다.

5. `document.dart` 파일에 `content` 속성을 클릭 → 전구 아이콘 클릭 → `Encapsulate field`를 선택해 주세요.

```
bin > getter_setter > document.dart > Document > content
1 class Document {
2   // 내용
3   String content;
4
5   More Actions...
6   Encapsulate field ✓
7
```

💡 그러면 아래와 같이 `content`에 대한 Getter와 Setter가 생성 됩니다.

```
bin > getter_setter > document.dart > Document > _content
1 class Document {
2   // 내용
3   String _content;
4
5   String get content => _content;
6
7   set content(String content) {
8     _content = content;
9   }
10
```

- 3번째 줄 `content` → `_content` : `_`로 시작하는 변수는 다른 파일에서 접근할 수 없는 비공개(private) 변수가 됩니다. (함수, 클래스 이름에서도 동일합니다)
- 5번째 줄 Getter : 외부에서 `content` 읽기를 수행하는 경우 호출되며, `_content`에 담긴 값을 반환합니다.
- 7번째 줄 Setter : 외부에서 `content = ?`와 같이 쓰기를 수행하는 경우 호출되며, 값을 가져와 `_content`에 할당합니다.



`set` 키워드를 사용하는 함수를 Setter라고 부릅니다.

```
set 이름(값) {
    // 할당시 수행하고 싶은 로직
}
```

이름에 값을 할당하는 경우 호출 됩니다.

```
이름 = 값
```



`content` Getter가 호출될 때 `readCount += 1` 을 실행하고, `content` Setter가 호출될 때 `updateCount += 1` 을 실행하도록 수정해 봅시다.

- 5번째 줄의 Getter에 추가 로직을 작성하기 위해 `=>` 클릭 → 전구 아이콘 클릭 → `Convert to block body` 를 선택해 주세요.

```
bin > getter_setter > document.dart > Document > content
1  class Document {
2      // 내용
3      String _content;
4
5      String get content => _content;
6
7      Quick Fix...
8      ⚡ Make field '_content' public
9      ⚡ Ignore 'unnecessary_getters_setters' for this line
10     ⚡ Ignore 'unnecessary_getters_setters' for this file
11     More Actions...
12     ⚡ Convert to async function body
13     ⚡ Convert to block body
14     // ...
```

그러면 화살표 구문이 기본 함수 형태로 변경됩니다.

```
bin > getter_setter > document.dart > ...
1  class Document {
2      // 내용
3      String _content;
4
5      String get content {
6          return _content;
7      }
8  }
```

7. 아래 이미지와 같이 Getter와 Setter에 count를 갱신하는 코드를 추가해 주세요.

▼ 코드스니펫 - `document.dart` : content Getter & Setter

```
readCount += 1;
```

```
updateCount += 1;
```

```
bin > getter_setter > document.dart > ...
1  class Document {
2      // 내용
3      String _content;
4
5      String get content {
6          readCount += 1;
7          return _content;
8      }
9
10     set content(String content) {
11         _content = content;
12         updateCount += 1;
13     }
14
15     // 읽은 횟수
16     int readCount = 0;
17
18     // 수정 횟수
19     int updateCount = 0;
20 }
```

8. 아래 이미지와 같이 `main.dart` 파일에서 8번째 줄과, 13번째 줄 코드를 삭제해 주세요.

```
bin > getter_setter > main.dart > ...
1  import 'document.dart';
2
   Run | Debug
3  void main() {
4      final doc = Document("1");
5
6      // 조회
7      String content = doc.content;
8      doc.readCount += 1;
9      print(doc.statistic);
10
11     // 수정
12     doc.content = "2";
13     doc.updateCount += 1;
14     print(doc.statistic);
15 }
16
```

그리고 실행해 보면 정상적으로 통계가 올라가는 것을 확인할 수 있습니다.

```
bin > getter_setter > main.dart > ...
1  import 'document.dart';
2
   Run | Debug
3  void main() {
4      final doc = Document("1");
5
6      // 조회
7      String content = doc.content;
8      print(doc.statistic);
9
10     // 수정
11     doc.content = "2";
12     print(doc.statistic);
13 }
14
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

```
readCount : 1 / updateCount : 0
readCount : 1 / updateCount : 1
Exited
```



`document.dart` 파일에서 `readCount` 와 `updateCount` 는 현재 외부에서 변경이 가능한 상태입니다.

```
bin > getter_setter > document.dart > ...
1  class Document {
2      // 내용
3      String _content;
4
5      String get content {
6          readCount += 1;
7          return _content;
8      }
9
10     set content(String content) {
11         _content = content;
12         updateCount += 1;
13     }
14
15     // 읽은 횟수
16     int readCount = 0;
17
18     // 수정 횟수
19     int updateCount = 0;
20
21     // 통계
22     String get statistic => "readCount : $readCount / updateCount : $updateCount";
23
24     Document(this._content);
25 }
26
```

미래의 나 또는 동료가 실수하지 않도록 외부에서 변경을 못하도록 만들어 봅시다.

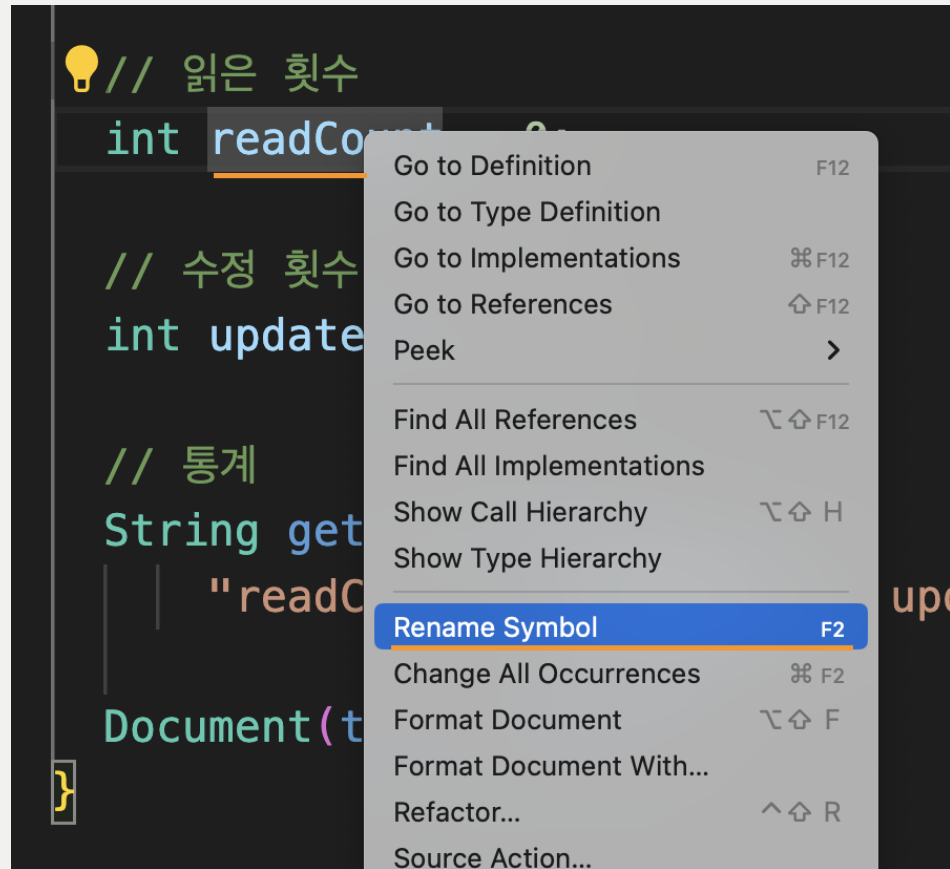
9. 변수의 이름 앞에 `_` (underbar)를 붙여 비공개 변수로 만들어 주세요.

- `readCount` → `_readCount`
- `updateCount` → `_updateCount`

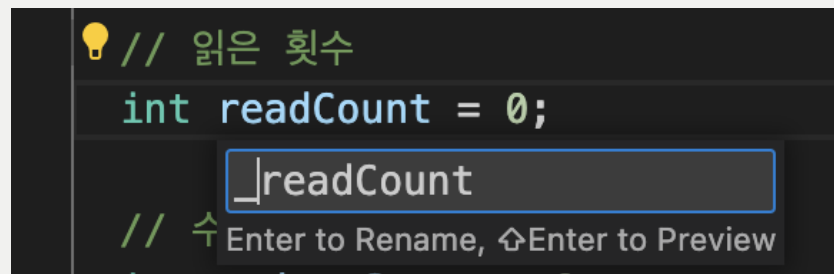


변수의 이름을 변경은 다음과 같은 방법으로 바꾸면 편리합니다.

1. 이름을 변경하려는 변수 클릭 → 마우스 우클릭 → **Rename Symbol** 클릭



2. 새로운 변수명 입력 후 엔터



해당 변수가 사용된 모든 곳이 수정됩니다.



```

bin > getter_setter > document.dart > ...
1  class Document {
2    // 내용
3    String _content;
4
5    String get content {
6      _readCount += 1;
7      return _content;
8    }
9
10   set content(String content) {
11     _content = content;
12     updateCount += 1;
13   }
14
15   // 읽은 횟수
16   int _readCount = 0;
17
18   // 수정 횟수
19   int updateCount = 0;
20
21   // 통계
22   String get statistic => "readCount : $_readCount / updateCount : $updateCount";
23
24   Document(this._content);
25 }
26

```

변수명이 중복되는 경우에는, 에디터에서 수정이 잘못될 수 있으므로 주의가 필요합니다.

▼ 코드스니펫 - `document.dart` : 최종

```

class Document {
  // 내용
  String _content;

  String get content {
    _readCount += 1;
    return _content;
  }

  set content(String content) {
    _content = content;
    _updateCount += 1;
  }

  // 읽은 횟수

```

```

    int _readCount = 0;

    // 수정 횟수
    int _updateCount = 0;

    // 통계
    String get statistic =>
        "readCount : $_readCount / updateCount : $_updateCo

    Document(this._content);
}

```

```

bin > getter_setter > document.dart > ...
1  class Document {
2      // 내용
3      String _content;
4
5      String get content {
6          _readCount += 1;
7          return _content;
8      }
9
10     set content(String content) {
11         _content = content;
12         _updateCount += 1;
13     }
14
15     // 읽은 횟수
16     int _readCount = 0;
17
18     // 수정 횟수
19     int _updateCount = 0;
20
21     // 통계
22     String get statistic =>
23         "readCount : $_readCount / updateCount : $_updateCount";
24
25     Document(this._content);
26 }
27

```



`main.dart` 파일에서 `doc.` 이라고 입력했을 때 외부에서 접근 가능한 속성 및 메소드가 노출되는데, `_readCount` 와 `_updateCount` 는 비공개 속성이라 접근이 불가능합니다.

```
bin > getter_setter > main.dart > main
1  import 'document.dart';
2
Run | Debug
3  void main() {
4      final doc = Document("1");
5
6      // 조회
7      String content = doc.content;
8      doc.
9      print content String
10     hashCode
11     // 수 runtimeType
12     doc. statistic
13     print toString()
14     noSuchMethod(...)
15 }
```

외부에서 접근하지 못하도록 데이터를 숨기는 과정을 **캡슐화(Encapsulation)** 라고 부릅니다.

## ▼ extends



`extends` 키워드를 사용하면 특정 클래스의 기능을 상속받을 수 있습니다.

1. `bin` 폴더 밑에 `extends.dart` 파일을 만들고 다음과 같이 작성해 주세요.

▼ 코드스니펫 - `extends.dart` : 시작

```
class Scanner {
    void scanning() => print("scanning...");
}

class Printer {
    void printing() => print("printing...");
}
```

```

}

class Machine extends Printer {}

void main() {
  final machine = Machine();
  machine.printing();
  // machine.scanning();
}

```

The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'DART\_PRACTICE' with a file named 'extends.dart' selected. The code editor shows the following Dart code:

```

1 class Scanner {
2   void scanning() => print("scanning...");
3 }
4
5 class Printer {
6   void printing() => print("printing...");
7 }
8
9 class Machine extends Printer {}
10
11 void main() {
12   final machine = Machine();
13   machine.printing();
14   // machine.scanning();
15 }
16

```

Below the code editor, there is a 'Run | Debug' button. The 'DEBUG CONSOLE' tab is active, showing the output:

```

printing...
Exited

```



`Machine` 이 `Printer` 를 상속 받았기 때문에, `printing()` 메소드를 사용할 수 있습니다.



`Scanner` 까지 상속 받아 `Machine` 을 복합기로 만들어 봅시다.

```
class Machine extends Printer, Scanner {}
```

Dart는 다중 상속이 안되므로 위와 같이 작성할 수 없습니다.

2. `Printer` 클래스가 `Scanner` 를 상속받으면, `Printer` 를 상속받은 `Machine` 에서 `scanning()` 도 가능합니다.

▼ 코드스니펫 - `extends.dart` : 최종

```
class Scanner {
  void scanning() => print("scanning...");
}

class Printer extends Scanner {
  void printing() => print("printing...");
}

class Machine extends Printer {}

void main() {
  final machine = Machine();
  machine.printing();
  machine.scanning();
}
```

```
1 class Scanner {
2   void scanning() => print("scanning...");
3 }
4
5 class Printer extends Scanner {
6   void printing() => print("printing...");
7 }
8
9 class Machine extends Printer {}
10
11 Run | Debug
12 void main() {
13   final machine = Machine();
14   machine.printing();
15   machine.scanning();
16 }
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL

printing...  
scanning...  
Exited



다중 상속이 안되어 `Printer` 가 `Scanner` 를 상속받아 구현은 가능했지만, 이로 인해 `Printer` 기능만 상속받을 수 없게 됩니다.

## ▼ mixin



`mixin` 을 이용하면 보다 자유롭게 여러 클래스의 속성과 메소드를 섞을 수 있습니다.

1. `bin` 폴더 밑에 `mixin.dart` 파일을 만들고 다음과 같이 작성해 주세요.

▼ 코드스니펫 - `mixin.dart` : 시작

```
class Scanner {
  void scanning() => print("scanning...");
}

class Printer {
```

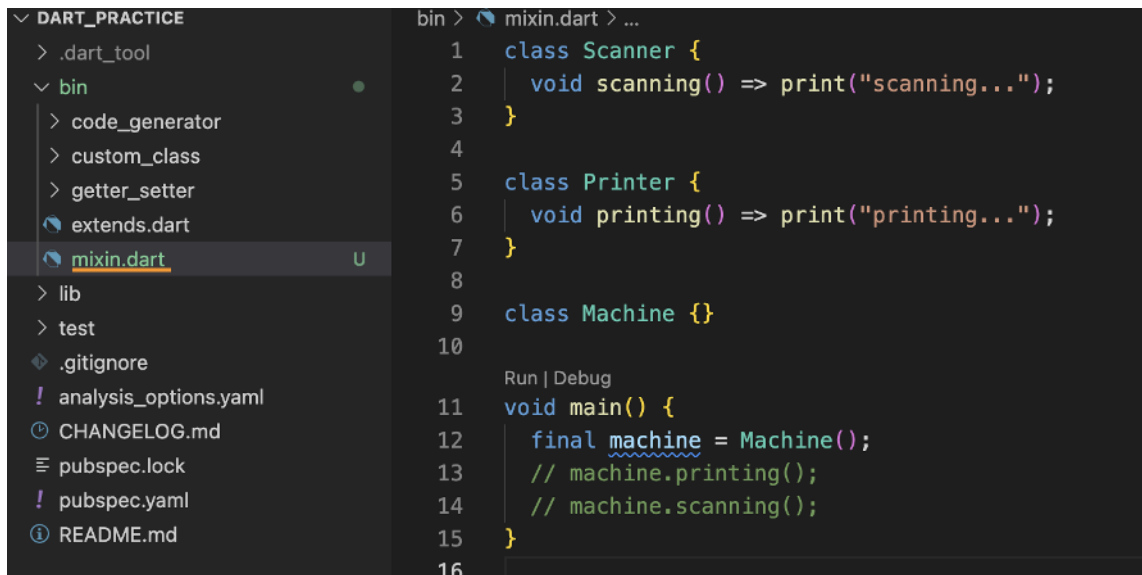
```

    void printing() => print("printing...");
  }

  class Machine {}

  void main() {
    final machine = Machine();
    // machine.printing();
    // machine.scanning();
  }

```



2. `class Scanner` 와 `class Printer` 앞에 `mixin` 키워드를 추가하면 `Machine` 클래스에 `with Printer, Scanner` 라고 추가하여 두 클래스의 기능을 `Machine` 클래스로 합칠 수 있습니다. `main()` 함수의 주석도 해제해 주세요.

▼ 코드스니펫 - `mixin.dart` : mixin class

```

mixin class Scanner {
  void scanning() => print("scanning...");
}

mixin class Printer {
  void printing() => print("printing...");
}

```

```
class Machine with Printer, Scanner {}

void main() {
  final machine = Machine();
  machine.printing();
  machine.scanning();
}
```

```
1  mixin class Scanner {
2    void scanning() => print("scanning...");
3  }
4
5  mixin class Printer {
6    void printing() => print("printing...");
7  }
8
9  class Machine with Printer, Scanner {}
10
    Run | Debug
11  void main() {
12    final machine = Machine();
13    machine.printing();
14    machine.scanning();
15  }
16
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text,

```
printing...
scanning...
Exited
```



위 문법을 **mixin**이라고 부르며, 두 클래스에 동일한 이름의 메서드나 속성이 존재하는 경우, 더 뒤에 선언된 클래스의 것이 사용됩니다.





`extends` 와 `with` 를 함께 사용하는 것도 가능합니다.

```
class Machine extends Printer with Scanner {}
```



Dart 3.0.0 이전 버전

```
class Scanner {}           // mixin 가능 & 인스턴스 생성 가능  
mixin Scanner {}          // mixin 가능 & 인스턴스 생성 불가능
```

Dart 3.0.0 이상 버전

```
class Scanner {}           // mixin 불가능 & 인스턴스 생성 가능  
mixin Scanner {}          // mixin 가능 & 인스턴스 생성 불가능  
mixin class Scanner {}     // mixin 가능 & 인스턴스 생성 가능
```



`mixin class` 키워드는 Dart 3.0.0 버전 이상부터 사용할 수 있습니다. Dart 버전은 터미널에서 다음 명령어를 통해 확인할 수 있습니다.

```
flutter --version
```

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
• devclass@DevStory dart_practice % flutter --version
Flutter 3.10.0 • channel unknown • unknown source
Framework • revision 84a1e904f4 (5 days ago) • 2023-05-09 07:41:44 -0700
Engine • revision d44b5a94c9
Tools • Dart 3.0.0 • DevTools 2.23.1
○ devclass@DevStory dart_practice %
```

#### ▼ Dart 버전 올리는 방법

1. 다음 명령어를 통해 flutter 버전을 올려주세요.

```
flutter upgrade
```

2. 업그레이드가 완료되면 VSCode를 다시 시작해 주세요.
3. `pubspec.yaml` 파일에서 Dart 버전을 확인해주세요.  
만약 아래와 같이

`sdk`의 버전이 `3.0.0` 미만으로 설정되어 있다면,

```
5
6   environment:
7     | sdk: '>=2.18.4 <3.0.0'
8
```

다음과 같이 변경해 주세요.

```
6   environment:
7     | sdk: '>=3.0.0 <4.0.0'
8
```


4. 그리고 저장( **Ctrl/Cmd + S** )해 주시면 Dart3 문법을 이용하실 수 있습니다.



**mixin**에 대한 보다 상세한 설명은 아래 공식 문서 링크를 참고해 주세요.

#### A tour of the Dart language

This page shows you how to use each major Dart feature, from variables and operators to classes and libraries, with the assumption that you already know

 <https://dart.dev/guides/language/language-tour#adding-features-to-a-class-mixins>




Dar-

▼ extension



`extension` 을 이용하면 특정 클래스의 기능을 추가할 수 있습니다.

DartPad

 <https://dartpad.dev/?id=2c7aeea85389dbd4303d389b239c9365>

```
1▼ void main() {
2    /// extension function 호출
3    print('HELLO'.equalsIgnoreCase('hello')); // true
4  }
5
6
7  /// String 클래스를 확장
8▼ extension MyStringExt on String {
9
10   /// Function
11   /// 대소문자 무시 비교
12▼ bool equalsIgnoreCase(String other) {
13     return toLowerCase() == other.toLowerCase();
14   }
15 }
16
```

String 클래스는 직접 수정할 수 없지만, `extension` 을 이용해 기능을 추가했습니다.



확장 클래스의 이름은 마음대로 지을 수 있으며, 아래 문법을 따라 작성하면 됩니다.

```
extension 이름 on 클래스명 {  
  
    /// function, getter, setter... 등 추가 가능  
}
```

`extension`에 대한 보다 상세한 사항은 링크를 참고해 주세요.

#### Extension methods

Extension methods add functionality to existing libraries. You might use extension methods without even knowing it. For example, when you use code completion in an

 <https://dart.dev/guides/language/extension-methods>

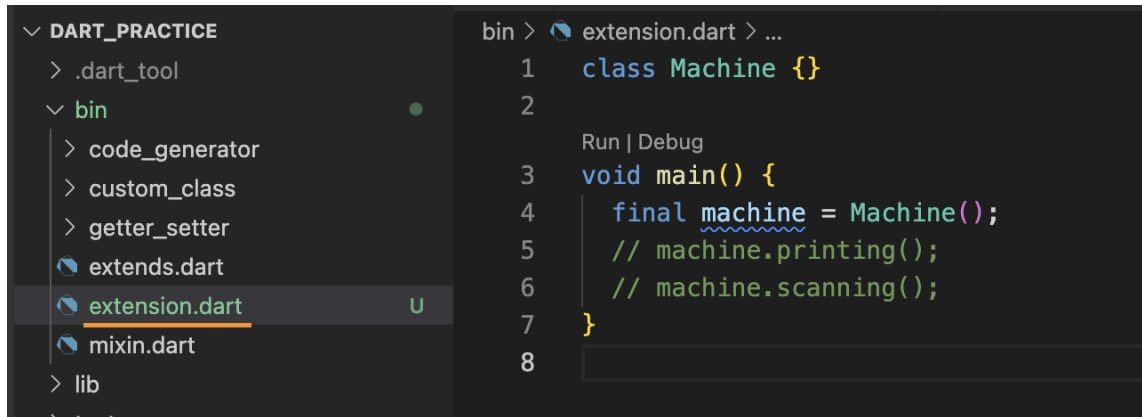


# Dart

1. `bin` 폴더 밑에 `extension.dart` 파일을 만들고 다음과 같이 작성해 주세요.

▼ 코드스니펫 - `extension.dart` : 시작

```
class Machine {}  
  
void main() {  
    final machine = Machine();  
    // machine.printing();  
    // machine.scanning();  
}
```



`Machine` 클래스에 `extension` 을 이용해 `printing()` 과 `scanning()` 기능을 추가해 봅시다.

2. 아래 이미지와 같이 `extension`을 추가하고, 주석을 해제해 주세요.

▼ 코드스니펫 - `extension.dart` : 최종

```
extension MyMachineExt on Machine {
  void scanning() => print("scanning...");
  void printing() => print("printing...");
}

class Machine {}

void main() {
  final machine = Machine();
  machine.printing();
  machine.scanning();
}
```

```
1 extension MyMachineExt on Machine {
2   void scanning() => print("scanning...");
3   void printing() => print("printing...");
4 }
5
6 class Machine {}
7
8 Run | Debug
9 void main() {
10   final machine = Machine();
11   machine.printing();
12   machine.scanning();
13 }
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, lexclude)

printing...  
scanning...  
Exited



**extension** 을 이용하면 클래스 수정 없이 기능을 확장할 수 있습니다.

## 최종 코드

GitHub - nero-angela/dart\_practice

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

[https://github.com/nero-angela/dart\\_practice](https://github.com/nero-angela/dart_practice)

nero-angela/  
**dart\_practice**



1 0 0 1  
Clone with GitHub Issues Stars Forks

© 2023 [DevStory](#). All rights reserved.