

**dart/flutter에 template method 적용하기**

# 1. Template Method에 효과적인 유즈케이스

템플릿 메서드 패턴은 다음과 같은 특징을 가진 유즈케이스에서 매우 효과적

- 공통된 알고리즘 구조를 공유하지만 세부 구현이 다른 경우
  - 모든 구현체가 동일한 순서의 단계를 따르며
  - 각 단계의 구체적인 로직만 다름
- 코드 중복을 제거하고 싶은 경우
  - 여러 클래스에서 유사한 로직이 반복됨
  - 공통 로직을 한 곳에서 관리하고 싶음
- 확장 가능한 프레임워크를 만들고 싶은 경우
  - 사용자가 특정 단계만 커스터마이징할 수 있도록 함
  - 전체 알고리즘의 제어는 프레임워크가 유지

## 2. 입력 유효성 검증 유즈케이스가 효과적인 이유

- 고정된 검증 순서 🖱️ 모든 validator는 동일한 검증 단계를 따름
- 다양한 검증 규칙 🖱️ 각 필드(이메일, 비밀번호 등)마다 규칙이 다름
- 재사용 가능한 공통 로직 🖱️ null 체크, 길이 검증 등은 공통
- 확장성 🖱️ 새로운 validator 추가가 쉬움
- 일관성 🖱️ 모든 검증이 동일한 인터페이스 사용

### 3. 입력 유효성 검증의 비즈니스 요구사항

## 3.1. Username (사용자명)

- 3-20자 길이
- 영문자로 시작
- 영문자, 숫자, 언더스코어(\_), 하이픈(-)만 허용
- 예약어 사용 불가 (admin, root, system, user)

## 3.2. Email (이메일)

- 5-254자 길이 (RFC 5321 표준)
- 올바른 이메일 형식 (local@domain.tld)
- 대소문자 구분 없음 (자동 소문자 변환)

### 3.3. Password (비밀번호)

- 8-128자 길이
- 최소 1개의 대문자 포함
- 최소 1개의 소문자 포함
- 최소 1개의 숫자 포함
- 최소 1개의 특수문자 포함
- 공백만으로 구성 불가

### 3.4. Phone (전화번호)

- 10-15자 길이
- 한국 전화번호 형식 (010-XXXX-XXXX)
- 구분자 자동 제거 (-, (), 공백)



## 4. 입력 유효성 검증의 기술적 요구사항

### 1. 일관된 검증 프로세스

☞ 모든 validator는 동일한 순서로 검증 수행하며, 실패 시 즉시 에러 반환

### 2. 명확한 에러 메시지

☞ 필드명과 구체적인 실패 이유 포함해 사용자가 이해하기 쉬운 메시지

### 3. 확장 가능한 구조

☞ 기존 코드 수정 최소화하면서, 새로운 validator 추가 용이

### 4. 테스트 가능성

☞ 각 validator를 독립적으로 테스트 가능해야하며, 특히 Edge case 검증 가능

## 5. 단위테스트로 요구사항 리뷰

## 5.1. EmailValidator 테스트 케이스 : 유효한 입력 검증

표준 이메일 형식 허용, null, 빈 문자열 거부

```
test('should accept valid email', () {  
    final result = validator.validate('test@example.com');  
    expect(result.isValid, true);  
    expect(result.errorMessage, isNull);  
});  
  
test('should reject null email', () {  
    final result = validator.validate(null);  
    expect(result.isValid, false);  
    expect(result.errorMessage, 'Email is required');  
});
```

## 5.2. EmailValidator 테스트 케이스 : 형식 검증

@ 기호, 도메인, TLD 필수

```
test('should reject email without @', () {  
    final result = validator.validate('testexample.com');  
    expect(result.isValid, false);  
    expect(result.errorMessage, 'Email must be a valid email address');  
});  
  
test('should reject email without TLD', () {  
    final result = validator.validate('test@example');  
    expect(result.isValid, false);  
});
```

## 5.3. EmailValidator 테스트 케이스 : 길이 검증

최소/최대 길이 제한

```
test('should reject email that is too short', () {  
    final result = validator.validate('a@b');  
    expect(result.isValid, false);  
    expect(result.errorMessage, 'Email must be at least 5 characters');  
});
```

## 5.4 PasswordValidator 테스트 케이스 : 복잡도 요구사항 검증

대문자, 소문자, 숫자, 특수문자 필수

```
test('should reject password without uppercase', () {  
    final result = validator.validate('password123!');  
    expect(result.isValid, false);  
    expect(result.errorMessage, 'Password must contain at least one uppercase letter');  
});  
  
test('should reject password without digit', () {  
    final result = validator.validate('Password!');  
    expect(result.isValid, false);  
    expect(result.errorMessage, 'Password must contain at least one digit');  
});  
  
test('should reject password without special character', () {  
    final result = validator.validate('Password123');  
    expect(result.isValid, false);  
    expect(result.errorMessage, 'Password must contain at least one special character');  
});
```

## 5.5. PasswordValidator 테스트 케이스 : 동적 요구사항

생성자 파라미터로 요구사항 커스터마이징

```
test('should accept password without uppercase when not required', () {  
    final customValidator = PasswordValidator(requireUppercase: false);  
    final result = customValidator.validate('password123!');  
    expect(result.isValid, true);  
});
```

## 5.6. PasswordValidator 테스트 케이스 : 공백 처리

비밀번호는 공백을 보존하지만, 공백만으로는 불가

```
test('should preserve leading and trailing spaces in password', () {  
    final result = validator.validate(' Password123! ');  
    expect(result.isValid, true);  
});  
  
test('should reject password that is only whitespace', () {  
    final result = validator.validate(' ');  
    expect(result.isValid, false);  
    expect(result.errorMessage, 'Password cannot be only whitespace');  
});
```



## 6. 코드 리뷰

## 6.1. BaseValidator 클래스의 알고리즘 스켈레톤 구현

lib/validators/base\_validator.dart

```
abstract class BaseValidator {
  final String fieldName;
  BaseValidator(this.fieldName);

  /// Template Method - 검증 알고리즘의 골격 정의
  ValidationResult validate(String? value) {
    // Step 1: null/빈 문자열 체크
    if (!checkNotEmpty(value)) return ValidationResult.failure(errorMessage: '$fieldName is required', fieldName: fieldName);

    final processedValue = preprocess(value!);

    if (!checkMinLength(processedValue)) return ValidationResult.failure(errorMessage: getMinLengthErrorMessage(), fieldName: fieldName);

    // Step 4: 최대 길이 검증
    if (!checkMaxLength(processedValue)) return ValidationResult.failure(errorMessage: getMaxLengthErrorMessage(), fieldName: fieldName);

    // Step 5: 형식 검증 (각 validator마다 다름)
    if (!checkFormat(processedValue)) return ValidationResult.failure(errorMessage: getFormatErrorMessage(), fieldName: fieldName);

    // Step 6: 추가 커스텀 검증 (선택적)

    final customValidationResult = performCustomValidation(processedValue);
    if (customValidationResult != null) return customValidationResult;

    // 모든 검증 통과
    return ValidationResult.success(fieldName: fieldName);
  }
}
```

## 6.2. BaseValidator 클래스의 추상메서드와 훅메서드

lib/validators/base\_validator.dart

```
abstract class BaseValidator {  
  // 추상 메서드들 (서브클래스에서 필수 구현)  
  bool checkMinLength(String value);  
  String getMinLengthErrorMessage();  
  bool checkMaxLength(String value);  
  String getMaxLengthErrorMessage();  
  bool checkFormat(String value);  
  String getFormatErrorMessage();  
  
  // 훅 메서드들 (서브클래스에서 선택적 오버라이드)  
  bool checkNotEmpty(String? value) => value != null && value.isNotEmpty;  
  
  String preprocess(String value) => value.trim();  
  
  // 기본적으로 추가 검증 없음  
  ValidationResult? performCustomValidation(String value) => null;  
}
```

## 6.3. BaseValidator의 역할과 특징

역할:

- 검증 알고리즘의 **\*\*골격(skeleton)\*\***을 구현
- 6단계의 검증 순서를 고정
- 각 단계는 abstract/hook 메서드 호출

특징:

- `final` 로 선언하여 서브클래스에서 오버라이드 불가 (Dart에서는 명시적으로 표시 권장)
- 모든 validator가 동일한 흐름을 따르도록 강제
- Hollywood Principle 적용: "우리가 당신을 호출할게요"

## 6.4. BaseValidator의 구현체 중 EmailValidator

lib/validators/email\_validator.dart

```
class EmailValidator extends BaseValidator {
  static const int minLength = 5; // a@b.c
  static const int maxLength = 254; // RFC 5321
  EmailValidator({String fieldName = 'Email'}) : super(fieldName);

  @override
  bool checkMinLength(String value) => value.length >= minLength;
  @override
  String getMinLengthErrorMessage() => '$fieldName must be at least $minLength characters';
  @override
  bool checkMaxLength(String value) => value.length <= maxLength;
  @override
  String getMaxLengthErrorMessage() => '$fieldName must not exceed $maxLength characters';
  @override
  bool checkFormat(String value) => RegExp(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$').hasMatch(value);
  @override
  String getFormatErrorMessage() => '$fieldName must be a valid email address';
  @override
  String preprocess(String value) => super.preprocess(value).toLowerCase();
}
```

## 6.5. BaseValidator의 구현체 중 PasswordValidator

```
lib/validators/password_validator.dart
```

구현 포인트:

- `preprocess()` 오버라이드로 trim 방지 (공백 보존)
- `performCustomValidation()` 활용하여 복잡도 검증
- 생성자 파라미터로 요구사항 동적 설정 가능
- 4가지 복잡도 규칙을 순차적으로 검증

## 6.6. BaseValidator의 구현체 중 UsernameValidator

```
lib/validators/username_validator.dart
```





구현 포인트:

- 기본 `preprocess()` 사용 (trim만 수행)
- 정규표현식으로 형식 검증 (영문자 시작 강제)
- `performCustomValidation()` 으로 비즈니스 규칙 검증 (예약어)
- 생성자로 예약어 목록 커스터마이징 가능

## 6.7. BaseValidator의 구현체 중 PhoneValidator

```
lib/validators/phone_validator.dart
```

구현 포인트:

-  `preprocess()` 오버라이드로 구분자 자동 제거
-  `countryCode` 에 따라 다른 형식 검증
-  한국 전화번호 특화 검증 (010, 011, 016, 017, 018, 019)
-  확장 가능한 구조 (다른 국가 코드 추가 용이)



## 6.8. BaseValidator의 구현체 비교

Validator	preprocess()	checkFormat()	performCustomValidation()
EmailValidator	trim + toLowerCase	이메일 정규식	사용 안 함
PasswordValidator	그대로 반환	공백만 체크	대소문자/숫자/특수문자 검증
UsernameValidator	기본 trim	영문자 시작 + 허용 문자	예약어 체크
PhoneValidator	trim + 구분자 제 거	국가별 전화번호 형 식	사용 안 함

## 7. Template Method 패턴 고려사항

- 고정된 알고리즘 순서: 단계가 명확하고 변경되지 않음
- 공통 로직 존재: 여러 구현체가 공유하는 코드가 있음
- 변형이 필요한 단계: 일부 단계는 구현체마다 다름
- 확장 가능성: 향후 새로운 구현체 추가 예상
- 제어 역전: 프레임워크가 전체 흐름을 제어해야 함

감사합니다.

[aiiiiiiden@gmail.com](mailto:aiiiiiiden@gmail.com)