

# Resp.java 제네릭(Generic) 이해하기

Resp.java에 나오는 클래스 제네릭 **T**, 메서드 제네릭 **B**, 와일드카드 **?** 를 예제와 함께 정리합니다.

## 1. 클래스 제네릭 **T** — **Resp<T>**

의미: "이 클래스는 어떤 타입 하나를 담는 상자다"라고 선언하는 것.

- **T**는 타입 파라미터 이름(보통 T = Type).
- **Resp<User>**, **Resp<String>**, **Resp<List<Board>>** 처럼 구체적인 타입을 넣어서 사용합니다.

예제

```
// T = User 로 고정된 응답
Resp<User> userResp = new Resp<>(200, "성공", user);
User body = userResp.getBody(); // 반환 타입이 User

// T = String
Resp<String> msgResp = new Resp<>(200, "성공", "hello");
String msg = msgResp.getBody(); // 반환 타입이 String

// T = List<Board>
Resp<List<Board>> listResp = new Resp<>(200, "성공", boards);
List<Board> list = listResp.getBody(); // 반환 타입이 List<Board>
```

정리: **T**는 "나중에 이 클래스를 쓸 때 정해질 타입"이라서, 한 번 **Resp<User>**로 만들면 **body**는 항상 **User** 타입으로 다를 수 있습니다.

## 2. 메서드 제네릭 **B** — **ok(B body)**

의미: "이 메서드는 호출할 때 넘겨준 인자 타입을 그대로 응답 타입으로 쓴다"는 뜻.

- **B**는 메서드에만 쓰이는 타입 파라미터(보통 B = Body).
- 클래스의 **T**와는 별개: **ok()**를 호출할 때 전달한 타입이 **B**가 됩니다.

메서드 시그니처

```
public static <B> ResponseEntity<Resp<B>> ok(B body)
//      ↑ 메서드 제네릭 선언: "이 메서드 안에서 타입 B를 쓴다"
//      ↑ 인자 타입 B   ↑ 반환 타입에 B가 그대로 반영
```

예제

```
// B = User → ResponseEntity<Resp<User>> 반환
User user = userService.findById(1);
return Resp.ok(user); // body 타입이 User 이므로 B=User

// B = AuthResponse
AuthResponse auth = new AuthResponse(...);
return Resp.ok(auth); // B=AuthResponse → ResponseEntity<Resp<AuthResponse>>

// B = List<Board>
List<Board> boards = boardService.findAll();
return Resp.ok(boards); // B=List<Board> → ResponseEntity<Resp<List<Board>>>
```

## 클래스 T vs 메서드 B

구분	선언 위치	언제 정해지나
T	클래스 <code>Resp&lt;T&gt;</code>	<code>Resp&lt;User&gt;</code> 처럼 변수/반환 타입을 선언할 때
B	메서드 <code>static &lt;B&gt; ... ok(B body)</code>	메서드에 넘기는 인자 타입에 따라 컴파일러가 추론

그래서 `Resp` 클래스는 `T`를 쓰지만, `ok()` 메서드는 그때그때 다른 타입을 쓰기 위해 **메서드만의 제네릭 B**를 둔 것입니다.

## 3. 와일드카드 ? — `Resp<?>`, `ResponseEntity<?>`

**의미:** "구체적인 타입을 지정하지 않고, 아무 타입이 올 수 있다"고만 표현할 때 사용.

- `?`는 "알 수 없는 타입 하나"를 의미합니다.
- 읽기용으로 쓰거나, 타입을 굳이 밝히지 않아도 될 때 사용합니다.

### fail 메서드에서의 사용

```
public static ResponseEntity<?> fail(HttpStatus status, String msg) {
    Resp<?> resp = new Resp<>(status.value(), msg, null);
    // ↑ body에 null만 넣으므로 "어떤 타입이든 상관없음" → ?
    return new ResponseEntity<>(resp, status);
    // ↑ 반환 타입도 "ResponseEntity 안에 뭐가 들어가든 상관없음" → ?
}
```

- `body`가 항상 `null`이므로 "타입이 `User`든 `String`이든 상관없다"는 뜻으로 `Resp<?>`를 씁니다.
- 호출하는 쪽에서도 "실패 응답의 `body` 타입을 쓰지 않는다"면 `ResponseEntity<?>`로 받아도 됩니다.

### 예제

```
// 성공 시: 구체 타입 사용 (제네릭 B 활용)
ResponseEntity<Resp<User>> okResp = Resp.ok(user);
User data = okResp.getBody().getBody(); // User 로 받음
```

```
// 실패 시: body를 쓰지 않으므로 ? 로 충분
ResponseEntity<?> failResp = Resp.fail(HttpStatus.BAD_REQUEST, "잘못된 요청");
Object body = failResp.getBody(); // Object처럼 다루거나, 그냥 상태/메시지만 사용
```

? vs 구체 타입 정리

표현	의미
Resp<User>	body는 반드시 User 타입이다.
Resp<?>	body는 어떤 타입일 수 있다. 구체 타입을 모르거나, 신경 쓰지 않을 때.
ResponseEntity<?>	ResponseEntity 안의 body 타입을 특정하지 않겠다.

4. 한 줄 요약

개념	역할
클래스 제네릭 T	Resp<T> — "이 응답이 담는 데이터의 타입"을 나중에 정해서 쓴다.
메서드 제네릭 B	ok(B body) — 넘긴 인자 타입을 그대로 Resp<B>로 감싸서 반환한다.
와일드카드 ?	Resp<?>, ResponseEntity<?> — "타입을 하나로 고정하지 않고, 아무 타입이어도 된다"고 표현할 때 쓴다.

이렇게 쓰면 같은 Resp 클래스로 User, List<Board>, String 등 어떤 타입이든 타입 안전하게 담을 수 있습니다.