



It's Your Life

with





my computer



# DAO, DTO, VO

## 개념 및 차이



수학·통계학



컴퓨터 공학



머신러닝(ML)



딥러닝(DL)



etc.

**DAO, DTO, VO**

4년 차 유부녀 와

**그놈이 그놈이다**



# DAO, DTO, VO 개념 및 차이

자~ 드가자~



VO

(Value Object)



SBS 스페셜

SBS HD



사실은 저 말할 게 있어요

어제 했어요~! ㅎㅎㅎ

```
public class User { 18 usages  👤 kdtTetz +1
    private int id; 4 usages
    private String userid; 4 usages
    private String name; 4 usages
    private String password; 3 usages
    private int age; 4 usages
    private boolean membership; 4 usages
    private Timestamp signupDate; 4 usages
```

```
public User(int id, String userid, String name,
    this.id = id;
    this.userid = userid;
    this.name = name;
    this.password = password;
    this.age = age;
    this.membership = membership;
    this.signupDate = signupDate;
}
```

회원 정보를 매칭 시키기 위해서 +  
회원 정보를 객체로 편하게 관리하기 위해서  
만들었던 User 클래스 기억 하시나요!?

고게 사실상의 VO 입니다!

그런데  
말입니다

SBS

실제 서비스에서는  
VO 를 좀 더 제한적인 의미로 사용합니다!



## VO 가 뭐의 약자였죠!?



이러한 책의 데이터가 막 바뀌면 될까요?

그럼 책을 읽는 사람마다  
다른 내용으로 알게 되겠죠!?



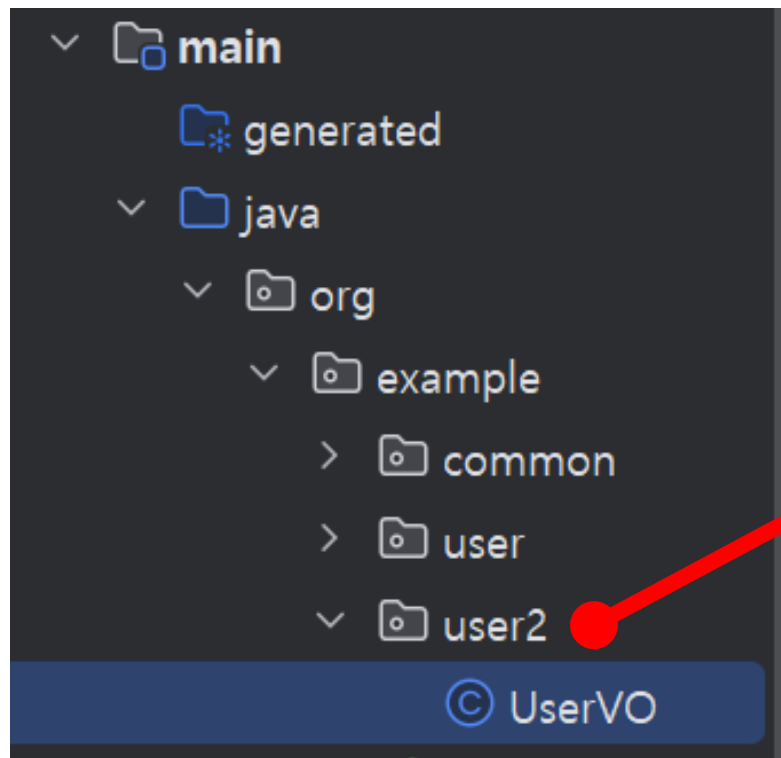
같은 내용이 담긴 데이터가 여러 개일 필요가 있을까요?

그래서 VO 간의 비교는  
동일성(identity, 객체 주소 값이 같음)이 아니라  
동등성(equality)에 기초하여 비교합니다!



# 따라서 VO는

- 기본적으로 데이터가 변경이 불가능 하도록 막습니다!
- 자바에서 데이터가 불가능하도록 하는 방법이 뭐죠!?
- 불변 객체로 만들기, Getter 만 설정하기의 테크닉이 들어 갑니다!
- 그리고 비교 메서드를 동등성 비교로 오버라이딩을 해서 사용합니다!



VO, DAO, DTO 를 위한  
user2 패키지 작성

```
public class UserVO { 2 usages new *
    private final int id; 4 usages
    private final String userid; 4 usages
    private final String name; 4 usages
    private final String password; 4 usages
    private final int age; 4 usages
    private final boolean membership; 4 usages
    private final Timestamp signupDate; 4 usages
}
```

객체의 필드를 전부 불변으로 설정

```
public UserVO(int id, String userid, String name, String password,
    this.id = id;
    this.userid = userid;
    this.name = name;
    this.password = password;
    this.age = age;
    this.membership = membership;
    this.signupDate = signupDate;
}
```



최초 생성시에만 값의 부여가 가능하므로  
생성자 설정

```
public int getId() { return id; } no usages new *
public String getUserId() { return userid; } no usages
public String getName() { return name; } no usages new
public String getPassword() { return password; } no usages
public int getAge() { return age; } no usages new *
public boolean isMembership() { return membership; } no usages
public Timestamp getSignupDate() { return signupDate; }
```

객체의 값을 변경하지 못하고  
값만 받을 수 있도록 Getter 만 설정



```
@Override new *
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    UserVO userVO = (UserVO) o;
    return id == userVO.id && age == userVO.age && membership == userVO.membership && Objects.equals
}
```



동일성(주소 값이 같음, == 로 비교)이 아닌  
동등성(실제 필드 값이 같음, equals 로 비교)을  
비교해야 하므로 equals 메서드 오버라이딩

아는형님

가인 | 형님들에게 가르쳐주고 싶은 것은?

D-12  
월드컵 최종예선  
대한민국 VS 카타르  
10/6(목) 저녁 7:00

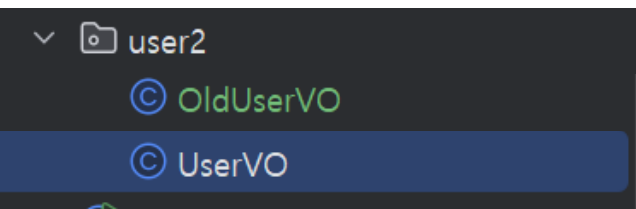
애써 쿨한 척  
이게

어제 배운 거랑 무슨 차이가 있냐!



Project  
Lombok

이 몸, 등장



```
8      R Rename usages
9      ▼ @Data
10     @AllArgsConstructor
11     public class UserLB {
12         private final int id;
13         private final String userid;
14         private final String name;
15         private final String password;
16         private final int age;
17         private final boolean membership;
18         private final Timestamp signupDate;
19     }
20
```

@Data 어노테이션은  
자동으로 Getter/Setter, equals,  
hashCode 메서드를 생성 해줍니다!

R Rename usages

9 `@Data new *`

10 `@AllArgsConstructor`

11 `public class UserLB {`

12  `private final int id;`

13  `private final String userid;`

14  `private final String name;`

15  `private final String password;`

16  `private final int age;`

17  `private final boolean membership;`

18  `private final Timestamp signupDate;`

19 `}`

`@AllArgsConstructor` 는  
모든 필드를 포함한  
생성자를 만들어 줍니다!



Project  
Lombok



짱이다..너무 짱인데..  
진짜 짱이다..오..  
완전 캡짱이다..







# DAO

## (Data Access Object)



애가 DAO 라고 생각하면 편합니다!

즉, 케익(= 데이터베이스)에는  
자기만 접근하겠다는 마인드!



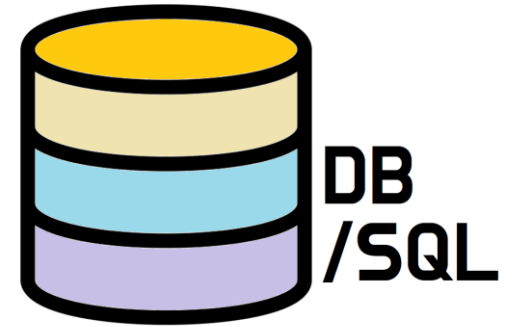


다른 클래스 1

다른 클래스 2

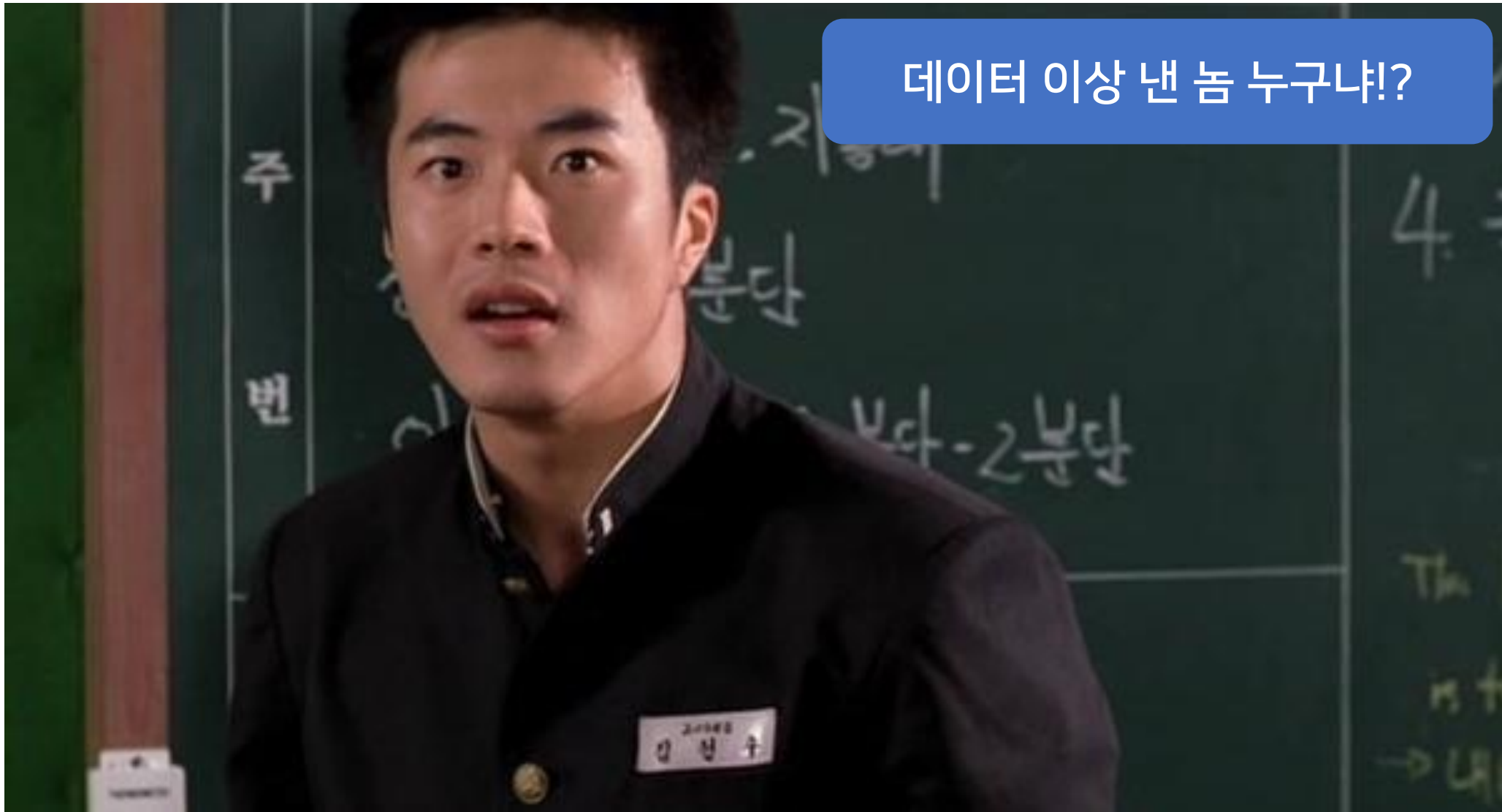
다른 클래스 3

다른 클래스 4





데이터 이상 낸 놈 누구냐!?







다른 클래스 1

다른 클래스 2

다른 클래스 3

다른 클래스 4







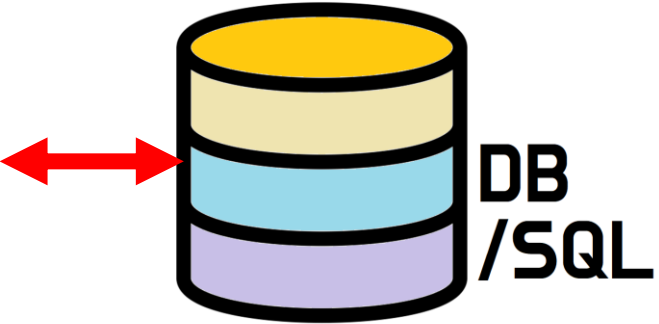
다른 클래스 1

다른 클래스 2

다른 클래스 3

다른 클래스 4

DAO





데이터 이상 낸 놈 누구냐!?

DAO







"저기요.. 잠깐만요 낮이 익는데.."

*As if I knew you already.. And came back to see you*



sample

common

user

ManageUser

ManageUserProc

ManageUserProc

ManageUserProc

ManageUserProc

User

user2

UserLB

UserVO

Main

1package org.example.user;

2

3> import ...

8

9public class ManageUser { 6 usages kdtTetz +1

10// 사용자 추가

11@public void addUser(User user) { 3 usages Tet

12Connection conn = JDBCUtil.getConnection();

13String sql = "INSERT INTO user\_table (userid

14

15try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

16pstmt.setString( parameterIndex: 1, user.getUserid()

17pstmt.setString( parameterIndex: 2, user.getName());

18pstmt.setString( parameterIndex: 3, user.getPassword

19pstmt.setInt( parameterIndex: 4, user.getAge());

20pstmt.setBoolean( parameterIndex: 5, user.isMemberst

21pstmt.executeUpdate();

22

23System.out.println("회원 추가 성공!");

24} catch (SQLException e) {


25throw new RuntimeException(e);

26}

27}

사실 어제 만들었던  
ManageUser 는 어떤 기능을 했죠?

user\_table 데이터 베이스에  
관련된 업무를 혼자서 전담했습니다!

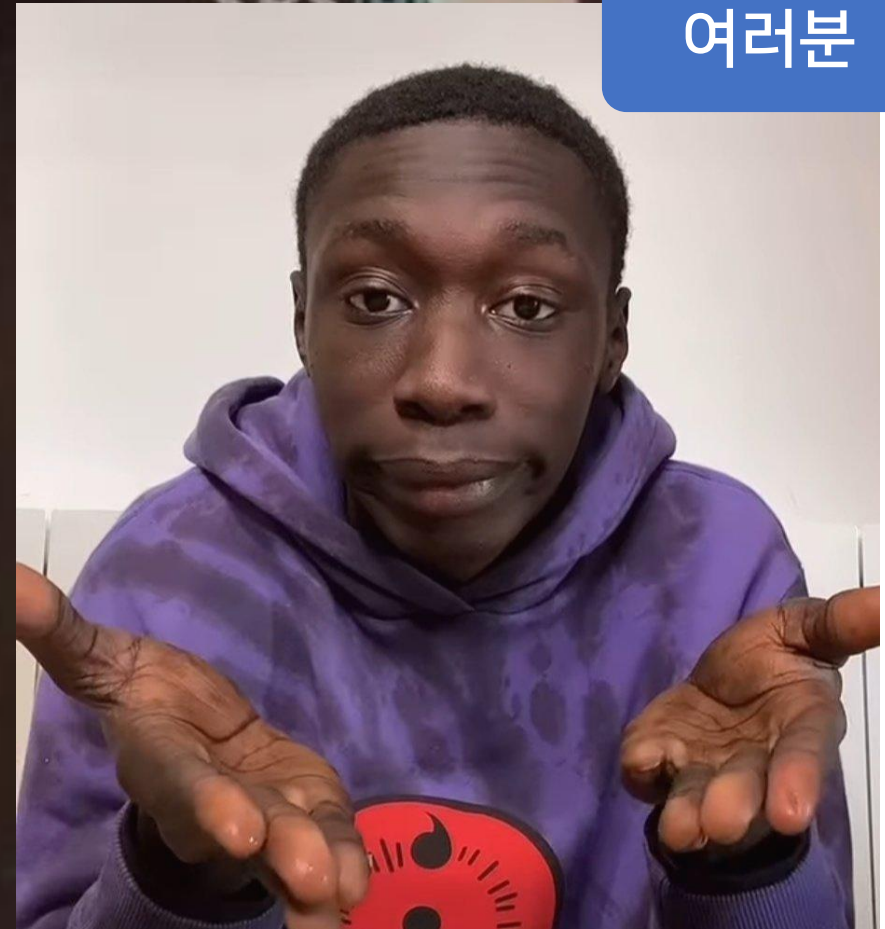




SBS 스페셜

사실은 저 말할 게 있어요

그게 그거이긴 합니다...



여러분





개선이라도 해야지

어떡해...



```
public class ManageUser { 6 usages 👤 kdtTetz +1
```

```
// 사용자 추가
```

```
public void addUser(User user) { 3 usages 👤 Tetz
```

```
Connection conn = JDBCUtil.getConnection();
```

```
String sql = "INSERT INTO user_table (userid, name, pas
```

메서드를 호출 할 때마다  
DB 접속을 매번  
새로 해줬습니다!

```
public void getAllUsers() { 2 usages 👤 kdtTetz +1
```

```
Connection conn = JDBCUtil.getConnection();
```

```
String sql = "SELECT * FROM user_table";
```

```
// 회원 삭제 기능
```

```
public void deleteUserById(int id) { 3 usages 👤 kdtTetz +1
```

```
Connection conn = JDBCUtil.getConnection();
```

```
String sql = "DELETE FROM user_table WHERE id = ?";
```



인간의 욕심은 끝이 없고



같은 실수를 반복한다



휴먼!!  
나는 그렇게 하지 않습니다!

```
public class UserDao { 3 usages new *  
    private final Connection conn; 2 usages  
  
    public UserDao() { 1 usage new *  
        this.conn = JDBCUtil.getConnection();  
    }  
}
```

접속 정보를 인스턴스가  
멤버로 가지고 있도록 변경!

접속 정보가 변경되면 안되므로  
불변으로 설정

해당 DAO 를 사용할 때  
접속 정보가 있어야만 하므로  
생성자에서 JDBCUtil 을 사용하여  
접속 정보를 멤버 변수에 저장하고  
사용 합니다!

이제 User 를 관리하는  
클래스가 변경 되었으므로  
UserVO 으로 변경

// 사용자 추가

```
public void addUser(UserVO user) { 1 usage new *  
    String sql = "INSERT INTO user_table (userid, name, password, age,  
  
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {  
        pstmt.setString(parameterIndex: 1, user.getUserid());  
        pstmt.setString(parameterIndex: 2, user.getName());  
        pstmt.setString(parameterIndex: 3, user.getPassword());  
        pstmt.setInt(parameterIndex: 4, user.getAge());  
        pstmt.setBoolean(parameterIndex: 5, user.isMembership());  
        pstmt.executeUpdate();  
  
        System.out.println("회원 추가 성공!");  
    } catch (SQLException e) {  
        throw new RuntimeException(e);  
    }  
}
```

// 모든 사용자 조회

```
public List<UserVO> getAllUsers() { 2 usages    new *
    String sql = "SELECT * FROM user_table";
    List<UserVO> users = new ArrayList<>();

    try (Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
        while (rs.next()) {
            UserVO user = new UserVO(
                rs.getInt(columnLabel: "id"),
                rs.getString(columnLabel: "userid"),
                rs.getString(columnLabel: "name"),
                rs.getString(columnLabel: "password"),
                rs.getInt(columnLabel: "age"),
                rs.getBoolean(columnLabel: "membership"),
                rs.getTimestamp(columnLabel: "signup_date")
            );
            users.add(user);
        }
        users.forEach(System.out::println);
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

전체 유저 조회도  
유저 데이터 타입을 UserVO 으로  
변경



```
5  ▶ public class ManageUserProgramV3 { new *
6  ▶  ▼ public static void main(String[] args) {
7      UserDAO userDAO = new UserDAO();
8      userDAO.getAllUsers();
9      userDAO.close();
10 }
11 }
12
```

변경된 UserDAO 의  
getAllUsers() 실행

```
UserLB(id=1, userid=xenosign, name=이효석2, password=12345, age=41, membership=true, signupDate=2024-07-16 15:10:23.0)
UserLB(id=2, userid=kimsiwan, name=김시완, password=1234, age=30, membership=true, signupDate=2024-07-16 15:10:38.0)
UserLB(id=3, userid=imsiwan, name=임시완, password=1234, age=38, membership=false, signupDate=2024-07-16 15:10:44.0)
UserLB(id=4, userid=pororo, name=뽀로로, password=1234, age=10, membership=false, signupDate=2024-07-16 15:10:45.0)
UserLB(id=6, userid=na, name=나건우, password=1234, age=28, membership=true, signupDate=2024-07-16 16:11:38.0)
UserLB(id=7, userid=faker, name=이상혁, password=goat, age=26, membership=false, signupDate=2024-07-16 16:31:30.0)
UserLB(id=9, userid=66, name=66, password=66, age=66, membership=true, signupDate=2024-07-17 12:01:57.0)
```

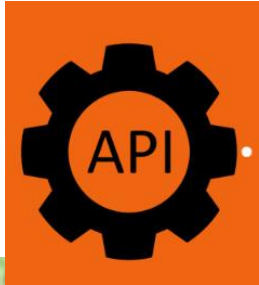


# DTO

## (Data Transfer Object)



*Backend*





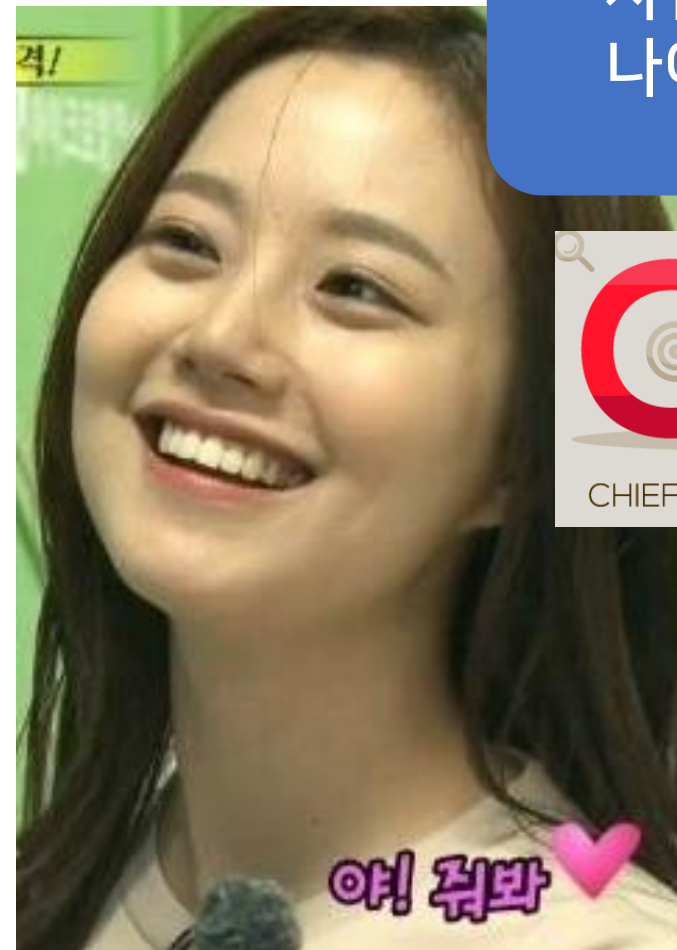
사람 이름이랑  
나이만 줘바바





```
> import ...  
  
✓ @Data 16 usages new *  
@AllArgsConstructor  
public class UserLB {  
    private final int id;  
    private final String userid;  
    private final String name;  
    private final String password;  
    private final int age;  
    private final boolean membership;  
    private final Timestamp signupDate;  
}
```





사람 이름이랑  
나이만 줘바바







```
public UserV0(int id, String userid, String name, String password, int age, boolean me
    this.id = id;
    this.userid = userid;
    this.name = name;
    this.password = password;
    this.age = age;
    this.membership = membership;
    this.signupDate = signupDate;
}
```

```
public int getId() { return id; } new *
public String getUserid() { return userid; } no usages new *
public String getName() { return name; } new *
public String getPassword() { return password; } new *
public int getAge() { return age; } new *
public boolean isMembership() { return membership; } new *
public Timestamp getSignupDate() { return signupDate; } no usages new *
```

```
@Override new *
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    UserV0 userV0 = (UserV0) o;
    return id == userV0.id && age == userV0.age && membership == userV0.membership &&
}
```



```
public UserVO(int id, String userid, String name, String password, int age, boolean me
    this.id = id;
    this.userid = userid;
    this.name = name;
    this.password = password;
    this.age = age;
    this.membership = membership;
    this.signupDate = signupDate;
}
```

데이터만 필요한 대상에게  
이러한 불필요한 코드들과  
메서드들 까지 보낼 필요가 있을까요?

```
public int getId() { return id; } new *
public String getUserid() { return userid; } no usages new *
public String getName() { return name; } new *
public String getPassword() { return password; } new *
public int getAge() { return age; } new *
public boolean isMembership() { return membership; } new *
public Timestamp getSignupDate() { return signupDate; } no usages new *
```

게다가 실제 현업이라면  
객체가 훨~~~~~씬  
더 복잡하겠죠!?

```
@Override new *
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    UserVO userVO = (UserVO) o;
    return id == userVO.id && age == userVO.age && membership == userVO.membership &&
}
```



```
public class UserDTO { no usages new *
```

```
private String name; 3 usages
```

```
private int age; 3 usages
```

데이터 전송에 불필요한 필드는  
제거합니다!

```
public UserDTO(String name, int age) { no usages new *
```

```
    this.name = name;
```

```
    this.age = age;
```

```
}
```

데이터 전송에 불필요한 로직 혹은  
비즈니스 로직도 전부 제거합니다!

```
public String getName() { return name; } new *
```

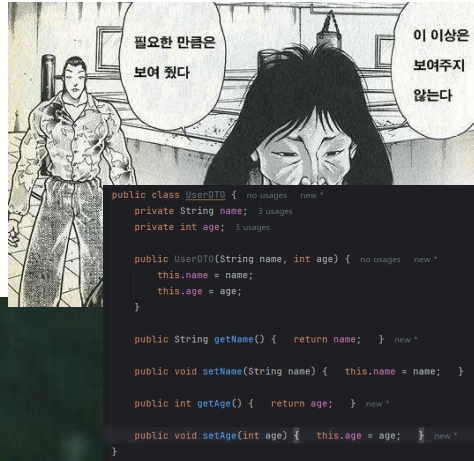
```
public void setName(String name) { this.name = name; } n
```

```
public int getAge() { return age; } new *
```

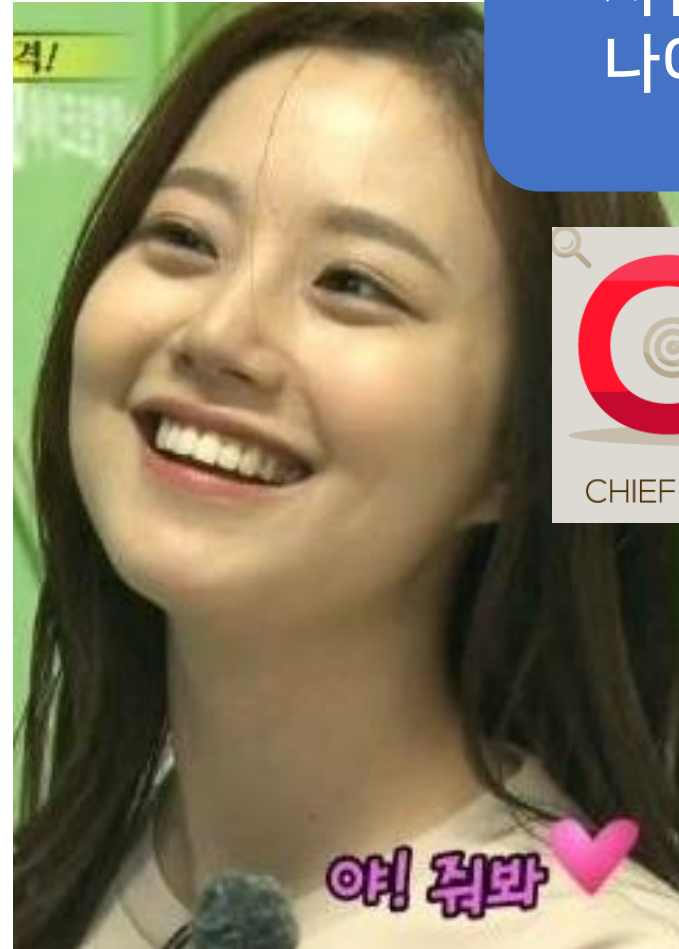
```
public void setAge(int age) { this.age = age; } new *
```

```
}
```

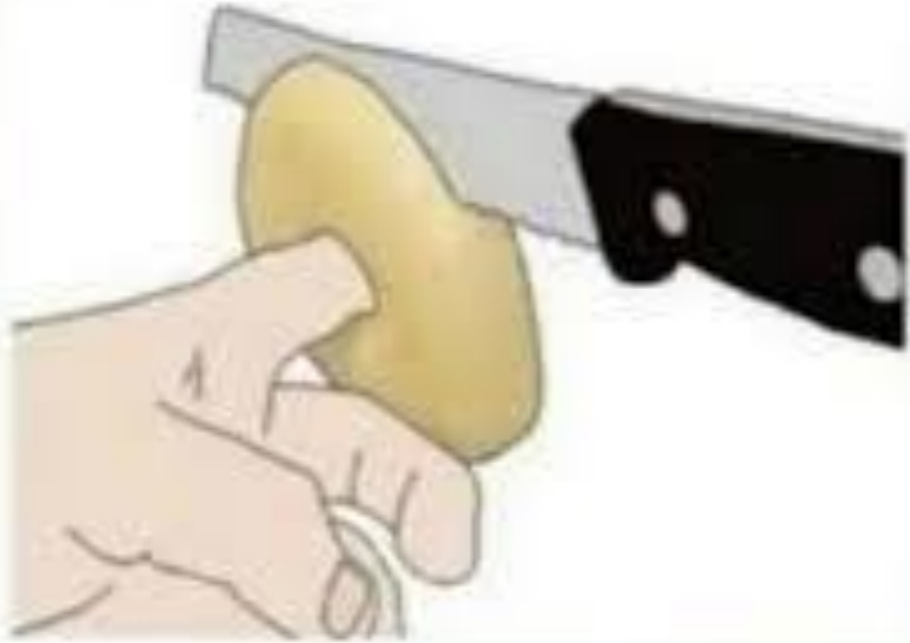




사람 이름이랑  
나이만 줘바바



베이글을 자를때 구멍에 손가락을  
넣으면 더욱 편리하게 자를 수  
있습니다



10번밖에 못쓴다는게 단점이네





## 그럼 DTO 를 쓰면 어떤 장점이 있을까요!?



1. 역할 분리 : 데이터를 처리하는 클래스와 전송하는 클래스를 구분하여  
역할 분리를 할 수 있습니다

2. 보안 및 데이터 무결성 확보 : 정보 객체의 모든 필드를 전달할 필요 없이  
필요한 정보만 선택적으로 전달이 가능합니다

3. 네트워크 최적화 : 큰 객체가 아닌 필요한 정보만 담은 작은 객체를 전송하여  
데이터를 절약할 수 있습니다