



It's Your Life

with





Maven 프로젝트로

만들기!



New Project

Search

New Project

- Java
- Kotlin
- Groovy
- Empty Project

Generators

- Maven Archetype
- Jakarta EE
- Spring Boot
- JavaFX
- Quarkus
- Micronaut

Name: jdbc_dao_ex2

Location: D:\git
Project will be created in: D:\git\jdbc_dao_ex2

☐ Create Git repository

Build system: IntelliJ **Maven** Gradle

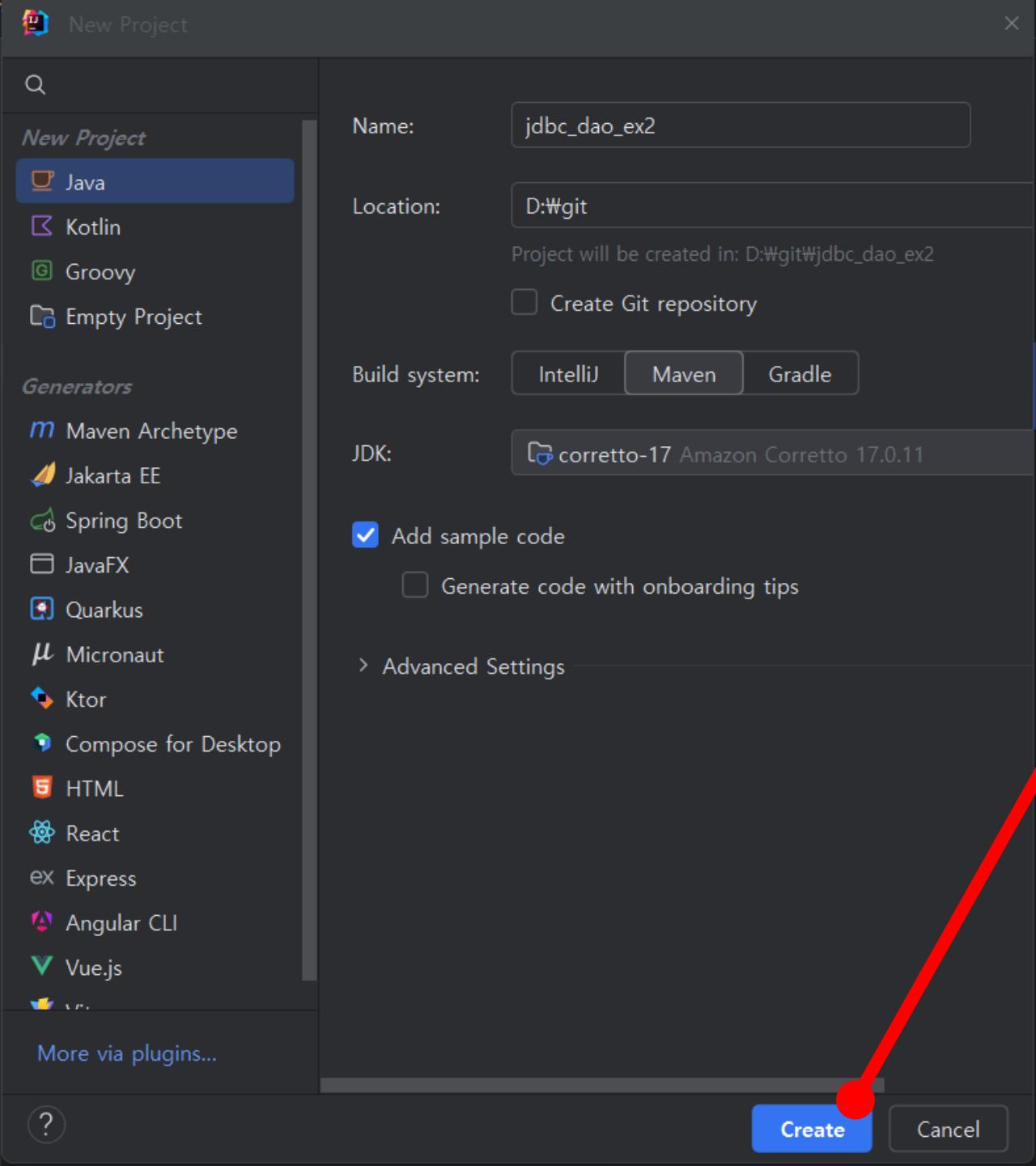
JDK: corretto-17 Amazon Corretto 17.0.11

☒ Add sample code
☐ Generate code with onboarding tips

> Advanced Settings

jdbc_dao_ex2 로 프로젝트 명
정하기

Maven 으로 빌드 시스템 설정
(Gradle 의 옛날 버전)



Create 로 Maven 프로젝트 생성

Project ▾



m pom.xml (jdbc_dao_ex2) ×



▼ jdbc_dao_ex2 D:\WgitW\jdbc_dao_ex2

> .idea

▼ src

▼ main

▼ java

▼ org

▼ example

🔄 Main

resources

> test

🚫 .gitignore

m pom.xml

> External Libraries

> Scratches and Consoles

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema
4         xsi:schemaLocation="http://maven.apache.org
5         <modelVersion>4.0.0</modelVersion>
6
7         <groupId>org.example</groupId>
8         <artifactId>jdbc_dao_ex2</artifactId>
9         <version>1.0-SNAPSHOT</version>
10
11        <properties>
12            <maven.compiler.source>17</maven.compiler.sc
13            <maven.compiler.target>17</maven.compiler.ta
14            <project.build.sourceEncoding>UTF-8</project
15
16        </properties>
17
18    </project>
```

Maven 프로젝트는
pom.xml 이 build.gradle 과 같이
프로젝트 정보를 관리 합니다!



```
10 <> dependencies
11 <> dependencyManagement
12 <> description
13 <> developers
14 <> modelVersion
```

Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards Next Tip

```
17 <de
```

<properties></ properties> 항목 아래
<de 정도를 치면 위와 같이 추천 목록이 뜨는데
<> dependencies 선택 후 엔터

→ 프로젝트 라이브러리 설정입니다!



```
<dependencies>
  <de
</> dependency
```

<dependencies> 를 만든 다음에
<dependencies> 내부에 <de 정도를 치면

다시 추가 항목이 뜨고
<> dependency 선택 후 엔터를 치면 아래와 같이
코드가 자동 완성 됩니다!

```
<dependencies>
  <dependency>
    <groupId></groupId>
    <artifactId></artifactId>
  </dependency>
</dependencies>
```


```
<dependencies>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId></artifactId>
  </dependency>
</dependencies>
```

<groupId> 내부에 com.mysql 을 치면
위와 같이 선택창이 뜨는 선택 후 엔터를 치면
아래와 같이 mysql-connector 라이브러리가
자동으로 추가 됩니다!

```
<dependencies>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.3.0</version>
  </dependency>
</dependencies>
```


m pom.xml (jdbc_dao_ex2) x

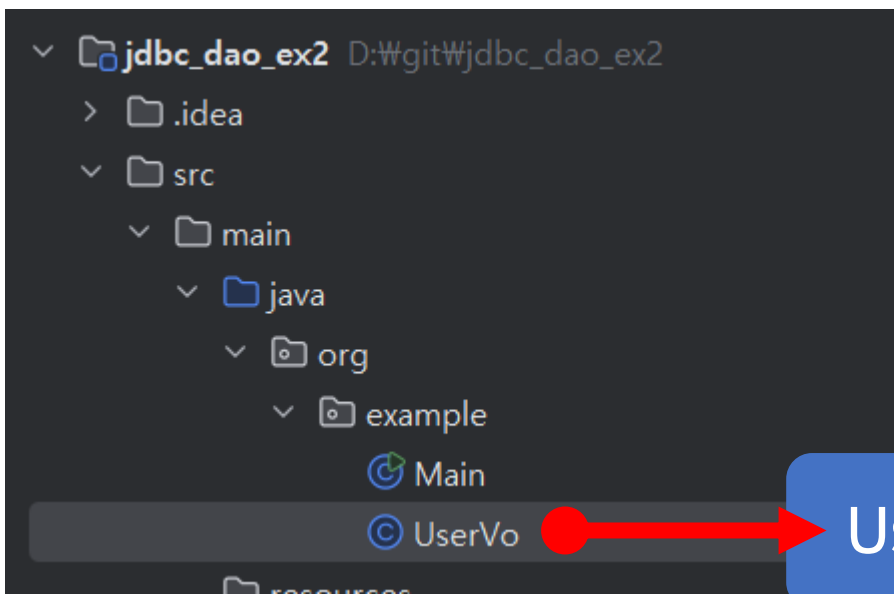
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
5       <modelVersion>4.0.0</modelVersion>
```



우측 상단의 maven sync 를 클릭해서
추가한 mysql 라이브러리를 세팅!



로복 없이 생성자 만들기



UserVo 클래스 추가

```
public class UserVo {  
    int id; no usages  
    String email; no usages  
    String password; no usages  
}
```

Setter 도 만들어야 하므로
final 키워드는 제거



Generate

Constructor

Getter

Setter

Getter and Setter

equals() and hashCode()

toString()

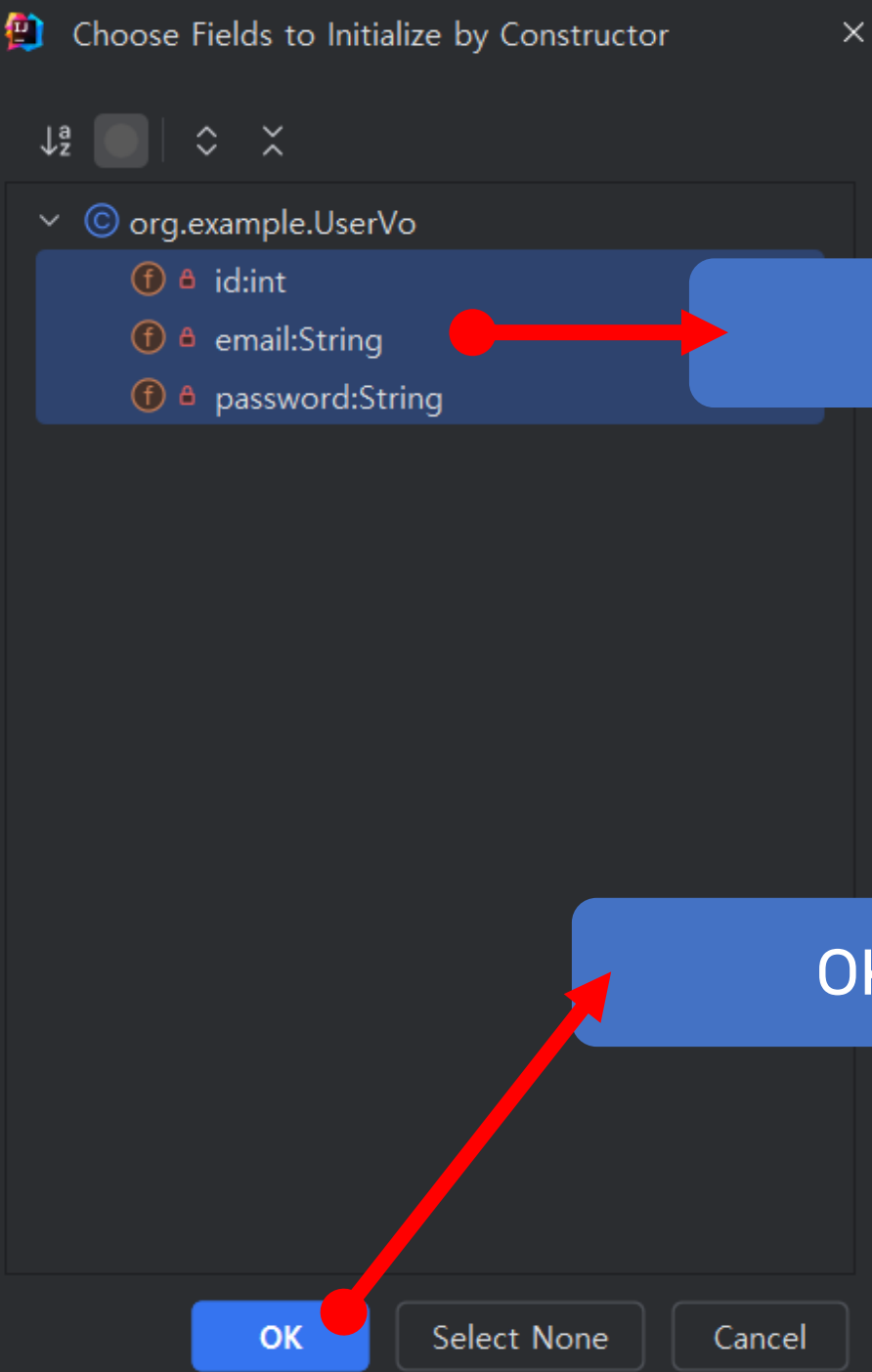
Override Methods... Ctrl+O

Delegate Methods...

Test...

Copyright

Alt + Insert (맥은 cmd + N)
누르고 Constructor 선택



모든 필드를 선택

OK 눌러서 생성자 생성



```
public class UserVo { no usages
    private int id; 1 usage
    private String email; 1 usage
    private String password; 1 usage

    public UserVo(int id, String email, String password) {
        this.id = id;
        this.email = email;
        this.password = password;
    }
}
```

인텔리제이에 의해 자동으로 생성자 생성
→ 롬복의 @AllArgsConstructor 기능



롬복 없이

Getter/Setter 작업



Generate

Constructor

Getter

Setter

Getter and Setter

equals() and hashCode()

toString()

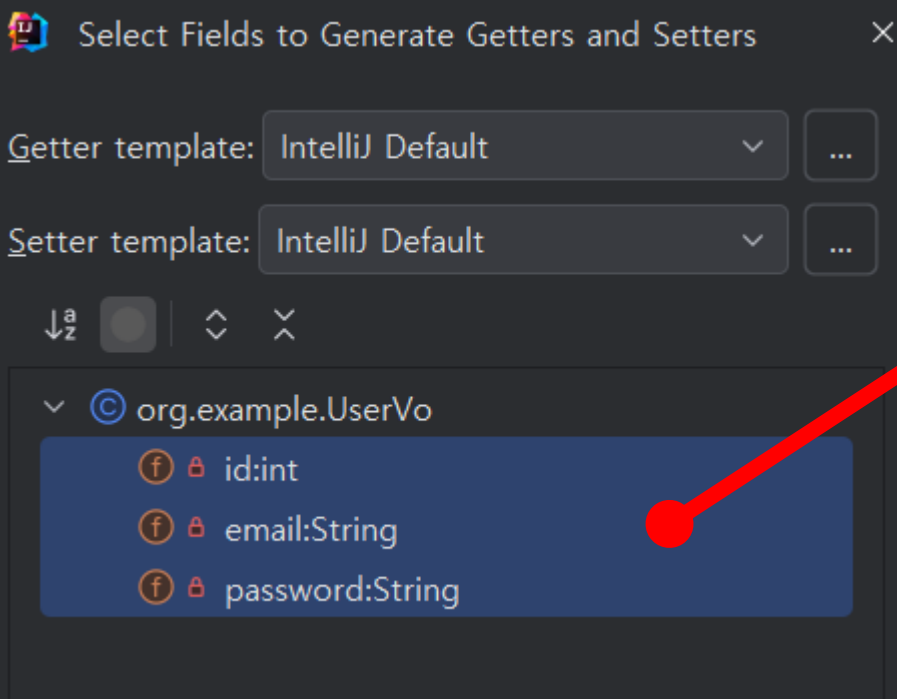
Override Methods... Ctrl+O

Delegate Methods...

Test...

Copyright

Alt + Insert (맥은 cmd + N)
누르고 선택



모든 필드 값 선택 후 OK





```
public int getId() { no usages  
    return id;  
}
```

```
public void setId(int id) { no usages  
    this.id = id;  
}
```

```
public String getEmail() { no usages  
    return email;  
}
```

```
public void setEmail(String email) { no usages  
    this.email = email;  
}
```

```
public String getPassword() { no usages  
    return password;  
}
```

```
public void setPassword(String password) { n  
    this.password = password;  
}
```

인텔리제이가 Getter / Setter
자동 생성!



toString

오버라이딩!



Generate

Constructor

Getter

Setter

Getter and Setter

equals() and hashCode()

toString()

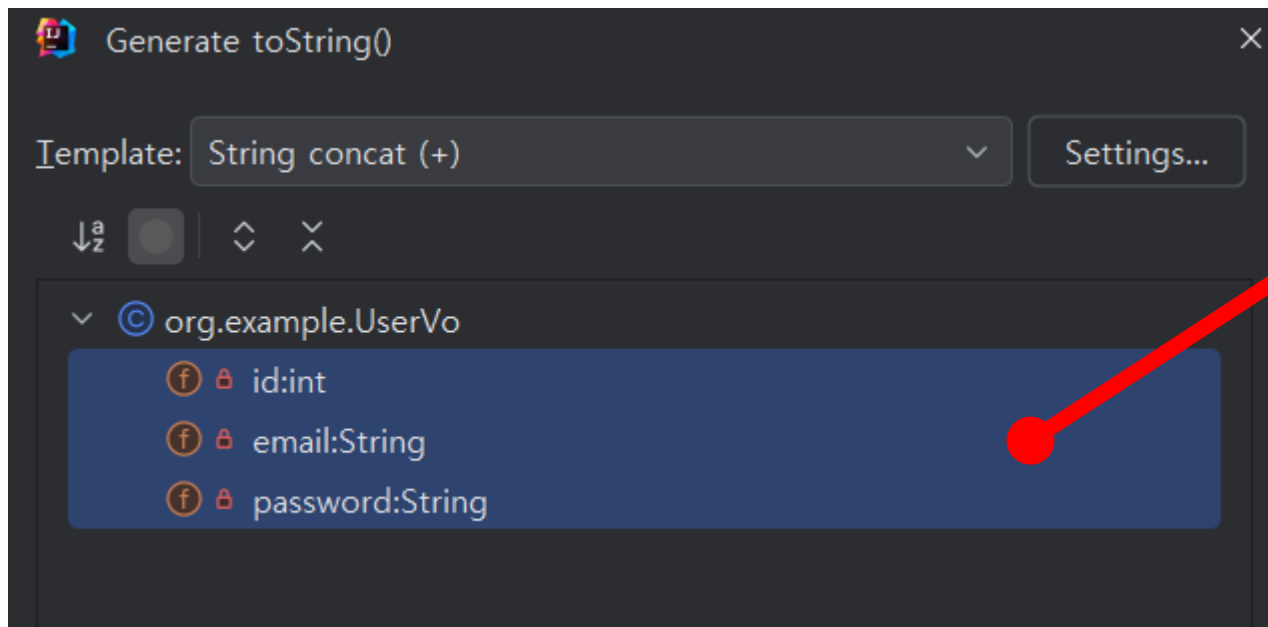
Override Methods... Ctrl+O

Delegate Methods...

Test...

Copyright

Alt + Insert (맥은 cmd + N)
누르고 선택



모든 필드 값 선택 후 OK

```
@Override
public String toString() {
    return "UserVo{" +
        "id=" + id +
        ", email='" + email + '\'' +
        ", password='" + password + '\'' +
        '}';
}
```



인텔리제이가 toString
메서드 자동 오버라이딩!





equals, hashCode

오버라이딩!



Generate

Constructor

Getter

Setter

Getter and Setter

equals() and hashCode()

toString()

Override Methods... Ctrl+O

Delegate Methods...

Test...

Copyright

Alt + Insert (맥은 cmd + N)
누르고 선택



Generate equals() and hashCode()

Template: `java.util.Objects.equ...()` (Java 7 and higher) ▾ ...

For class type comparison in equals() method generate: ?

☐ instanceof expression

☒ getClass() comparison expression

☐ Use getters when available

? **Next** Cancel

Next 클릭

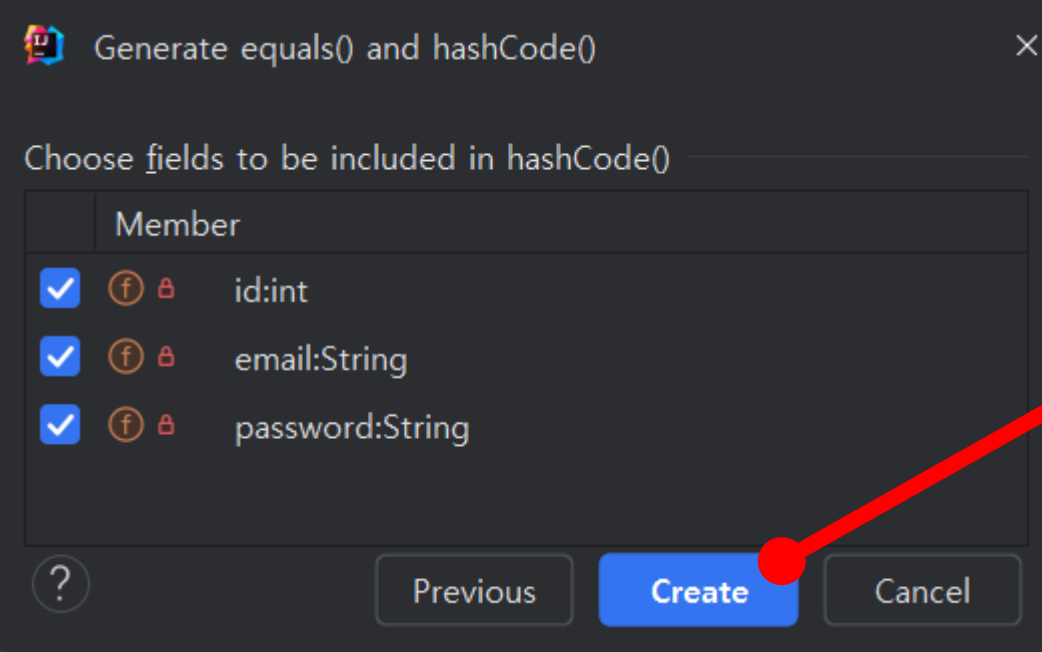
Generate equals() and hashCode()

Choose fields to be included in equals()

	Member
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input type="checkbox"/> id:int
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input type="checkbox"/> email:String
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input type="checkbox"/> password:String

? Previous **Next** Cancel

Next 클릭



```
@Override
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 && Objects.equals(email, userVo.email) && Objects

}

@Override
public int hashCode() {
    return Objects.hash(id, email, password);
}
```

인텔리제이가 equals / hashCode
메서드 자동 오버라이딩!



CRUD 코드

like 전공반 스타일

Create



```
public class UserDao { no usages
    // 회원 추가 메서드
    public void addUser(UserVo newUser) { no usages
        try {
            // 1. Driver 커넥터 설정
            String driver = "com.mysql.cj.jdbc.Driver";
            Class.forName(driver);
            System.out.println("1. 드라이버 설정 OK");

            // 2. DB 연결
            String url = "jdbc:mysql://localhost:3306/user_ex";
            String id = "root";
            String password = "1234";
            Connection conn = DriverManager.getConnection(url, id, password);
            System.out.println("2. DB연결 OK");
```

try-with-resources 문을 쓰지 않고
하나의 try/catch 문으로 처리

각각 메서드마다 driver 클래스를
따로 만들어서 설정

접속도 메서드별로 따로 하는 방식!
Connection conn 이 각각 메서드에
지역 변수로 존재!



// 3. SQL 문 생성

```
String sql = "INSERT INTO users (email, password) VALUES (?, ?)";  
PreparedStatement pstmt = conn.prepareStatement(sql);  
pstmt.setString(parameterIndex: 1, newUser.getEmail());  
pstmt.setString(parameterIndex: 2, newUser.getPassword());  
System.out.println("3. SQL문 생성 OK");
```

이제는 매개변수가 아닌
UserVo 객체를 받으므로
Getter 를 사용해서 데이터 삽입

// 4. SQL 문 전송

```
int affetedRows = pstmt.executeUpdate();  
if (affetedRows > 0) {  
    System.out.println("회원 추가 성공!");  
} else {  
    System.out.println("회원 추가 실패");  
}
```

SQL 문을 생성

SQL 전송 및 실행

// 5. 자원 해제

```
pstmt.close();  
conn.close();
```

```
// 5. 자원 해제
```

```
pstmt.close();
```

```
conn.close();
```

```
} catch (Exception e) {
```

```
    e.printStackTrace();
```

```
}
```

```
}
```

try-with-resources 문을 사용하지 않으므로
자원을 직접 해제

try 구문을 하나만 쓰므로 다양한 예외를
처리하기 위해 Exception 클래스 하나로
예외를 처리



이제 회원 email, password 정보를
매개 변수로 전달하는게 아니라 회원 객체를
만들어서 전달해야 하므로
UserVo 객체를 만들어서 전달

```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
  
        // 회원 추가, 이제 회원 추가는 회원 객체를 만들어서 전달 해야함!  
        UserVo newUser = new UserVo(id: 0, email: "tetz2", password: "1234");  
        userDao.addUser(newUser);  
    }  
}
```

id 값은 자동으로 생성 되므로
아무 값이나 전달해도 문제 없습니다!

새롭게 생성된 회원 객체를
addUser 에 전달!

Read





// 모든 회원 정보를 조회하는 메서드

```
public List<UserVo> getAllUsers() { no usages
    // 리턴할 데이터를 미리 선언! try/catch 문이 종료 되고 리턴이 되어야 하므로
    // Scope 의 개념으로 try/catch 문 밖에 존재해야 함
    List<UserVo> userList = new ArrayList<>();

    try {
        // 1. Driver 커넥터 설정
        String driver = "com.mysql.cj.jdbc.Driver";
        Class.forName(driver);
        System.out.println("1. 드라이버 설정 OK");

        // 2. DB 연결
        String url = "jdbc:mysql://localhost:3306/user_ex";
        String id = "root";
        String password = "1234";
        Connection conn = DriverManager.getConnection(url, id, password);
        System.out.println("2. DB연결 OK");
```

메서드에서 직접 출력하는 것이 아니라
회원 정보 리스트 컬렉션을 리턴하는
형태로 변경

각각 메서드마다 driver 클래스를
따로 만들어서 설정

접속도 메서드별로 따로 하는 방식!
Connection conn 이 각각 메서드에
지역 변수로 존재!



// 3. SQL 문 생성

```
String sql = "SELECT id, email, password FROM users";  
Statement stmt = conn.createStatement();  
System.out.println("3. SQL문 생성 OK");
```

SQL 문을 생성

// 4. SQL 문 전송

```
ResultSet rs = stmt.executeQuery(sql);  
// 결과 데이터를 전부 순회하는 while 문  
while (rs.next()) {  
    int userid = rs.getInt(columnLabel: "id");  
    String email = rs.getString(columnLabel: "email");  
    String userpassword = rs.getString(columnLabel: "password");  
  
    // 결과 데이터를 바탕으로 회원 정보 객체(=UserVo)를 만들고 해당 객체  
    UserVo user = new UserVo(userid, email, userpassword);  
    userList.add(user);  
}
```

SQL 전송 및 실행 & 실행 결과를
ResultSet 데이터로 받기

rs 의 각각 행 데이터를 받아서
UserVo 객체로 만들어서 리스트에 추가



```
// 5. 자원 해제
```

```
rs.close();
```

```
stmt.close();
```

```
conn.close();
```

```
} catch (Exception e) {  
    e.printStackTrace();  
}
```

try-with-resources 문을 사용하지 않으므로
자원을 직접 해제

```
// 6. 결과 리턴
```

```
// 데이터가 전부 추가된 리스트를 리턴!
```

```
// 통신이 잘못되면 try/catch 구문이 정상적으로 실행이 안되므로 빈 리스트가 리턴
```

```
return userList;
```

```
}
```

완성된 리스트를 리턴!

```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
  
        // 전체 회원 조회 메서드  
        List<UserVo> users = userDao.getAllUsers();  
  
        for (UserVo user : users) {  
            System.out.println(user);  
        }  
    }  
}
```

이제 getAllUsers 는 List 컬렉션을 리턴!

회원 정보는 리턴 받은 컬렉션을 이용하여
메인 클래스에서 직접 출력 합니다!

Update





// 회원 정보를 수정하는 메서드

```
public void updateUser(int userid, String newEmail, String newPassword) {  
    try {  
        // 1. Driver 커넥터 설정  
        String driver = "com.mysql.cj.jdbc.Driver";  
        Class.forName(driver);  
        System.out.println("1. 드라이버 설정 OK");  
  
        // 2. DB 연결  
        String url = "jdbc:mysql://localhost:3306/user_ex";  
        String id = "root";  
        String password = "1234";  
        Connection conn = DriverManager.getConnection(url, id, password);  
        System.out.println("2. DB연결 OK");  
    }  
}
```

각각 메서드마다 driver 클래스를
따로 만들어서 설정

접속도 메서드별로 따로 하는 방식!
Connection conn 이 각각 메서드에
지역 변수로 존재!



// 3. SQL 문 생성

```
String sql = "UPDATE users SET email = ?, password = ? WHERE id = ?";  
PreparedStatement pstmt = conn.prepareStatement(sql);  
pstmt.setString(parameterIndex: 1, newEmail);  
pstmt.setString(parameterIndex: 2, newPassword);  
pstmt.setInt(parameterIndex: 3, userid);  
System.out.println("3. SQL문 생성 OK");
```

SQL 문을 생성

// 4. SQL 문 전송

```
int affectedRows = pstmt.executeUpdate();  
if (affectedRows > 0) {  
    System.out.println("회원 정보 수정 성공!");  
} else {  
    System.out.println("회원 정보 수정 실패");  
}
```

SQL 전송 및 실행


```
// 5. 자원해제  
pstmt.close();  
conn.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}
```

try-with-resources 문을 사용하지 않으므로
자원을 직접 해제





```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
  
        // 회원 수정  
        userDao.updateUser( userid: 5, newEmail: "tetz3", newPassword: "1234");  
    }  
}
```



기존과 동일한 스타일로
update 실행

Delete





// 회원 정보를 삭제하는 메서드

```
public void deleteUser(int userid) { no usages
```

```
try {
```

```
    // 1. Driver 커넥터 설정
```

```
    String driver = "com.mysql.cj.jdbc.Driver";
```

```
    Class.forName(driver);
```

```
    System.out.println("1. 드라이버 설정 OK");
```

```
    // 2. DB 연결
```

```
    String url = "jdbc:mysql://localhost:3306/user_ex";
```

```
    String id = "root";
```

```
    String password = "1234";
```

```
    Connection conn = DriverManager.getConnection(url, id, password);
```

```
    System.out.println("2. DB연결 OK");
```

각각 메서드마다 driver 클래스를
따로 만들어서 설정

접속도 메서드별로 따로 하는 방식!
Connection conn 이 각각 메서드에
지역 변수로 존재!



```
// 3. SQL 문 생성
String sql = "DELETE FROM users WHERE id = ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(parameterIndex: 1, userid);
System.out.println("3. SQL문 생성 OK");
```

SQL 문을 생성

```
// 4. SQL 문 전송
int affectedRows = pstmt.executeUpdate();
if (affectedRows > 0) {
    System.out.println("회원 삭제 성공!");
} else {
    System.out.println("회원 삭제 실패");
}
```

SQL 전송 및 실행

```
// 5. 자원해제  
pstmt.close();  
conn.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}
```

try-with-resources 문을 사용하지 않으므로
자원을 직접 해제





```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
  
        // 회원 삭제  
        userDao.deleteUser(userid: 5);
```

기존과 동일한 스타일로
delete 실행

JOIN





```
// 테이블을 합친 뒤, 회원의 이름 정보까지 전부 출력하는 메서드
public void getAllUsersWithName() { no usages
    try {
        // 1. Driver 커넥터 설정
        String driver = "com.mysql.cj.jdbc.Driver";
        Class.forName(driver);
        System.out.println("1. 드라이버 설정 OK");

        // 2. DB 연결
        String url = "jdbc:mysql://localhost:3306/user_ex";
        String id = "root";
        String password = "1234";
        Connection conn = DriverManager.getConnection(url, id, password);
        System.out.println("2. DB연결 OK");
```

각각 메서드마다 driver 클래스를
따로 만들어서 설정

접속도 메서드별로 따로 하는 방식!
Connection conn 이 각각 메서드에
지역 변수로 존재!



// 3. SQL 문 생성

```
String sql = "SELECT users.id, users.email, users.password, user_info.name " +  
            "FROM users " +  
            "JOIN user_info ON users.id = user_info.id";  
Statement stmt = conn.createStatement();  
System.out.println("3. SQL문 생성 OK");
```

SQL 문을 생성

// 4. SQL 문 전송

```
ResultSet rs = stmt.executeQuery(sql);  
// 결과 데이터를 전부 순회하는 while 문  
while (rs.next()) {  
    int userid = rs.getInt(columnLabel: "id");  
    String email = rs.getString(columnLabel: "email");  
    String userpassword = rs.getString(columnLabel: "password");  
    String name = rs.getString(columnLabel: "name");  
  
    System.out.printf("ID: %d, Email: %s, Password: %s, Name: %s\n", userid, ema  
}
```

SQL 전송 및 실행 후
데이터를 ResultSet 로 받아서
데이터를 순회하면서
각각의 데이터 출력

```
// 5. 자원 해제  
rs.close();  
stmt.close();  
conn.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}
```

try-with-resources 문을 사용하지 않으므로
자원을 직접 해제





```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
  
        // 이름이 출력되는 회원 조회 메서드  
        userDao.getAllUsersWithName();
```

기존과 동일한 스타일로
delete 실행

