



It's Your Life

with





MongoDB

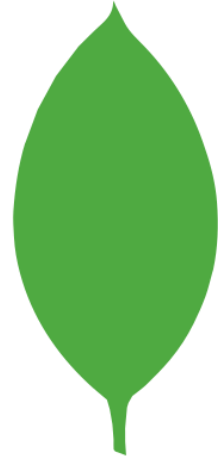
JDBC



해 20년으로 지경고 이별은 저쳤다> 종에서



Java™



mongoDB

가끔 우리가 안맞는다고 생각이 들 수도 있어



JS

내 소원을 들어줘
꼭 들어줘야 해



mongoDB

왜 이렇게 그애를
만나는 게 어려운 걸까?

유 퀴즈 200

본 방송

200회 특집
김연아의 은퇴 후 첫 토크쇼

MBC HD

무슨 생각하면서 (스트레칭을) 하세요?
무슨 생각을 해.... 그냥 하는거지

시키면 해야지 뭐 어떡해



일단 세팅 &

연결 부터!



```
dependencies {  
    testImplementation platform('org.junit:junit-bom:5.10.0')  
    testImplementation 'org.junit.jupiter:junit-jupiter'  
    // MySQL 접속용 라이브러리  
    implementation 'com.mysql:mysql-connector-j:8.3.0'  
    // 롬복 관련 라이브러리  
    compileOnly 'org.projectlombok:lombok:1.18.30'  
    annotationProcessor 'org.projectlombok:lombok:1.18.30'  
    testCompileOnly 'org.projectlombok:lombok:1.18.30'  
    testAnnotationProcessor 'org.projectlombok:lombok:1.18.30'  
    // MongoDB 라이브러리  
    implementation 'org.mongodb:mongodb-driver-sync:5.0.0'  
    implementation 'ch.qos.logback:logback-classic:1.2.11'  
}
```

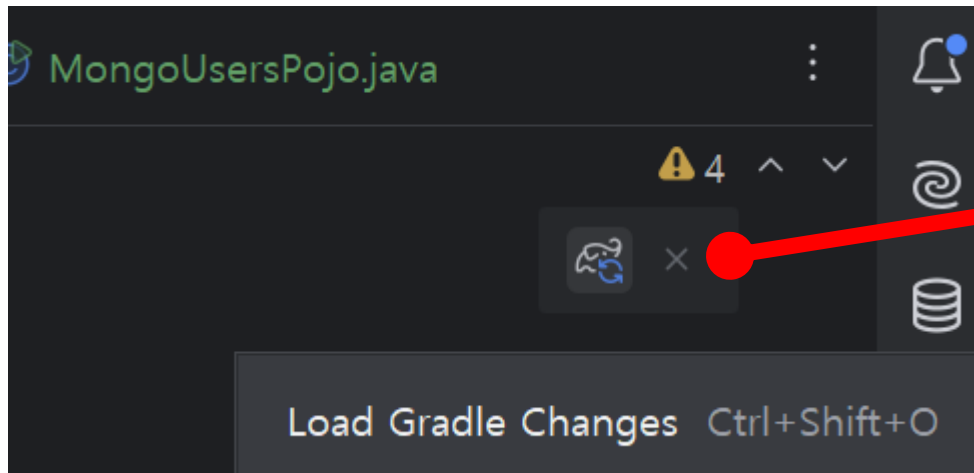
Gradle 에
MongoDB 관련 라이브러리 추가

요걸 복붙 하세요
이상의 VT 가 끼는데 삭제하고
엔터!

```
implementation 'org.mongodb:mongodb-driver-sync:5.0.0'  
implementation 'ch.qos.logback:logback-classic:1.2.11'
```

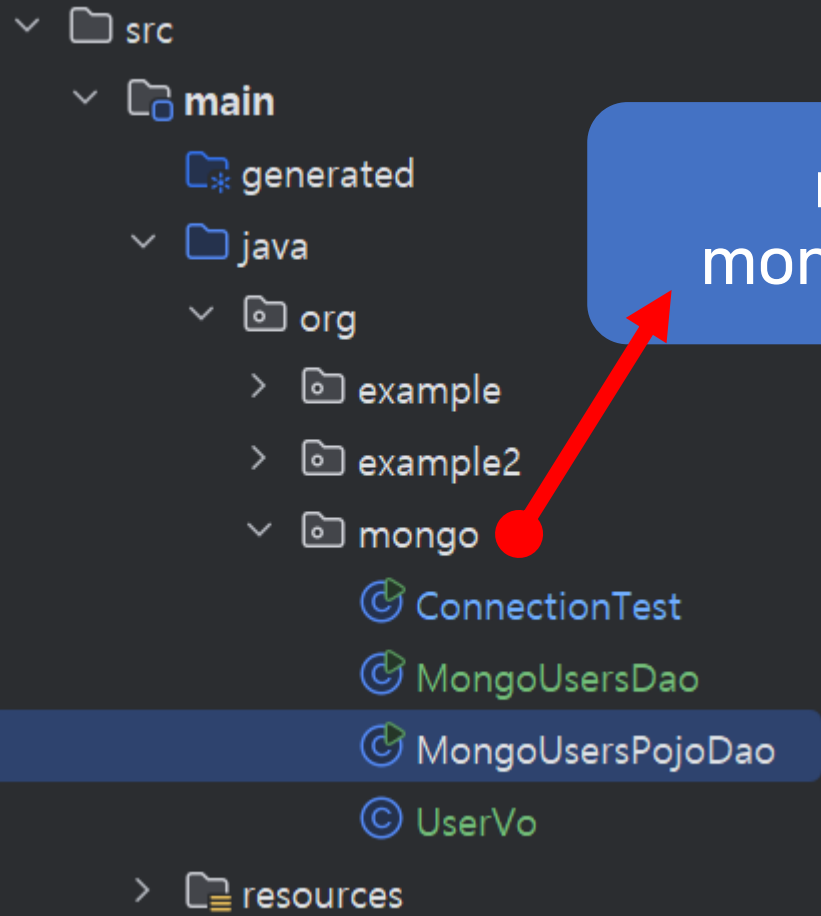


Gradle 싱크





mongoDB 를 위한
mongo 패키지를 만듭시다!



```
public class ConnectionTest {  👤 kdtTetz *  
    public static void main(String[] args) {  👤 kdtTetz *  
        String uri = "mongodb://127.0.0.1:27017";  
        String db = "test2";  
        try (MongoClient client = MongoClient.create(uri)) {  
            MongoDBDatabase database = client.getDatabase(db);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

MongoDB 서버 접속 주소!

DB 이름!
그런데 아무거나
넣어도 됩니다!

없어도 일단 있는 척 접속하고
실제 데이터가 삽입되면
그때 만듭니다!



황현진실

도대체 왜
이러는
걸까요





구조가 없으니까
그때그때 막 만들어도
문제가 없습니다!!




```
365 [main] INFO org.mongodb.driver.client - MongoClient with metadata {"driver": {"name": "mongo-java-driver|sync", "version":  
  "5.0.0"}, "os": {"type": "Windows", "name": "Windows 10", "architecture": "amd64", "version": "10.0"}, "platform":  
  "Java/Amazon.com Inc./17.0.11+9-LTS"} created with settings MongoClientSettings{readPreference=primary,  
writeConcern=WriteConcern{w=null, wTimeout=null ms, journal=null}, retryWrites=true, retryReads=true,  
readConcern=ReadConcern{level=null}, credential=null, transportSettings=null, commandListeners=[],  
codecRegistry=ProvidersCodecRegistry{codecProviders=[ValueCodecProvider{}, BsonValueCodecProvider{}, DBRefCodecProvider{},  
DBObjectCodecProvider{}, DocumentCodecProvider{}, CollectionCodecProvider{}, IterableCodecProvider{}, MapCodecProvider{},  
GeoJsonCodecProvider{}, GridFSFileCodecProvider{}, JsrEnumCodecProvider{}, com.mongodb.client.model.mql.Expr.  
Jep395RecordCodecProvider@38afe297, com.mongodb.Kotli  
loggerSettings=LoggerSettings{maxDocumentLength=1000},  
mode=SINGLE, requiredClusterType=UNKNOWN, requiredRepl  
serverSelectionTimeout='30000 ms', localThreshold='15  
readTimeoutMS=0, receiveBufferSize=0, proxySettings=Pr  
heartbeatSocketSettings=SocketSettings{connectTimeoutM  
proxySettings=ProxySettings{host=null, port=null, user  
connectionPoolSettings=ConnectionPoolSettings{maxSize=
```





그건 난 모르겠고



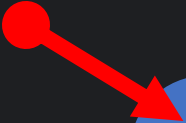


MongoDB 용

DAO 만들기

- CREATE

```
public class MongoUsersDao {  
    static MongoClient client; 3 usages  
    static MongoDBDatabase database; 3 usages  
    static {  
        ConnectionString connectionString = new ConnectionString("mongodb://127.0.0.1:27017")  
        client = MongoClient.create(connectionString);  
        database = client.getDatabase(s: "test2");  
    }  
}
```



static 을 이용하여
Java 프로그램이 시작될 때
자동으로 접속을 시도하고
접속 정보를 client 에 저장!

접속 정보를 종료하는 메서드

```
public static void close() { client.close(); }

public static MongoDBDatabase getDatabase() { return database; }
public static MongoCollection<Document> getCollection(String colName) {
    MongoCollection<Document> collection = database.getCollection(colName);
    return collection;
}
```

매개 변수로 collection
이름을 전달하고
해당 컬렉션에 접속한 정보를
가져오는 메서드

내부에 운영 클래스 만들기!

```
public static void main(String[] args) {  
    MongoClient mongoClient = new MongoClient(mongoUri);  
    MongoClientCollection<Document> users = mongoClient.getCollection(colName: "users");  
    System.out.println("===== 메인 메서드 시작 =====");  
  
    // CREATE  
    Document newUser = new Document();  
    newUser.append("name", "이효석");  
    newUser.append("age", 40);  
    users.insertOne(newUser);  
}
```

users 컬렉션에 접속 정보 가져오기!

역시나 컬렉션이 없어도 문제가 없습니다!
데이터가 삽입되면 그때 만듭니다!



```
public static void main(String[] args) { new *
    MongoClient
```

MongoDB 는 구조가 있나요!?

→ 따라서 구조가 있는
자바 객체가 아닌 Bson 이라고 하는
MongoDB를 위한 독특한 형태의
JSON 객체를 사용 합니다!

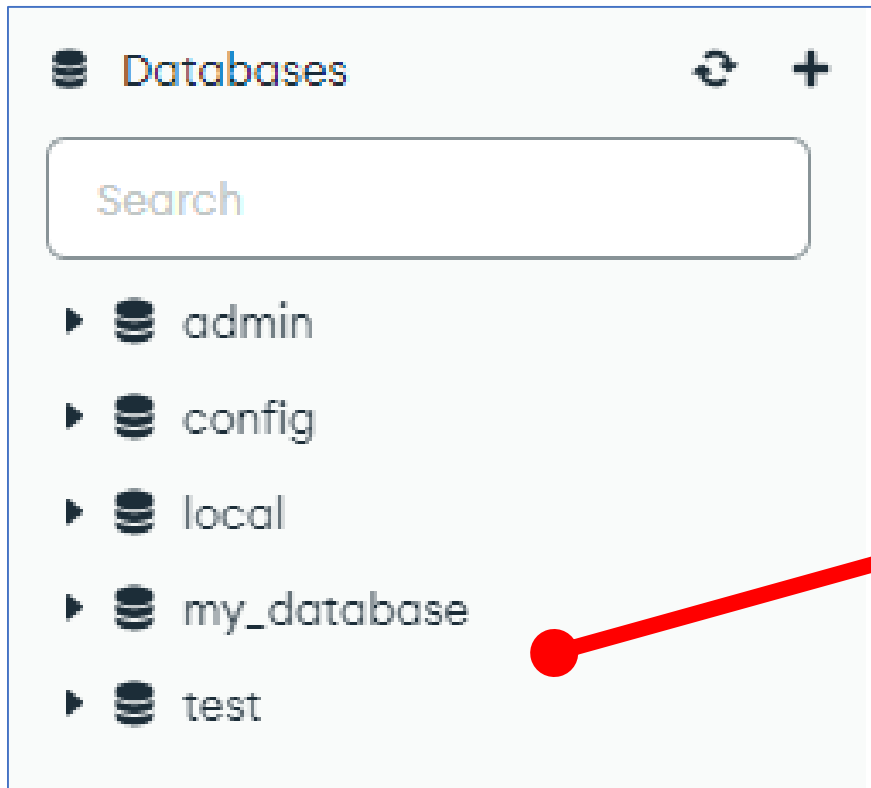
→ JSON 의 구조를 가지면서
데이터 전송이 빠르고
프로퍼티 추가 삭제가 편리



```
public static void main(String[] args) { new *
    MongoClient
```

Document 에
key 가 name, value 가 이효석인
프로퍼티와 나이 프로퍼티 추가!

MongoDB 의 users 컬렉션에
새롭게 만든 Document 를 추가!



아직 test2 데이터 베이스는
존재하지 않습니다!



```
42 ▶ public static void main(String[] args) { new *
43     MongoClient<Document> users = getCollection(colName: "users");
44     System.out.println("===== 메인 메서드 시작 =====");
45
46
47     // CREATE
48     Document newUser = new Document();
49     newUser.append("name", "이효석");
50     newUser.append("age", 40);
51     users.insertOne(newUser);
```





```
roundTripTimeNanos=20388900}  
===== 메인 메서드 시작 =====
```

```
Process finished with exit code 0
```

에러가 없다!?

Databases

Search

- admin
- config
- local
- my_database
- test
- test2
 - users

Type a query: { field: 'value' } or [Generate](#)

[ADD DATA](#) [EXPORT DATA](#) [UPDATE](#)

```
{  
  "_id": ObjectId('669f15f4ea3b27423a66e022'),  
  "name": "이효석",  
  "age": 40  
}
```

도큐먼트도
무사히 추가!

test2 데이터베이스와
users 컬렉션이 자동으로 생성!



긱이에요 긱~



READ



```
// READ
FindIterable<Document> doc = users.find();
Iterator itr = doc.iterator();
while (itr.hasNext()) {
    System.out.println("==> findResultRow : " + itr.next());
}
```

MongoDB 검색 기능인
find() 결과를 저장하기 위해
자주 사용하는 FindIterable 컬렉션

Like MySQL 의 ResultSet

결과 데이터 순회를 위해서 Iterator 타입으로 변경 후
itr.hasNext() 를 사용하여 다음 데이터가 있는지
확인(= rs.next())하고 순회!

itr.next() 는 데이터를 가르키기 때문에 각각의 Document 가 출력!



```
===== 메인 메서드 시작 =====
```

```
==> findResultRow : Document({_id=669f15f4ea3b27423a66e022, name=이효석, age=40})
```

```
==> findResultRow : Document({_id=669f18e68a1d6820ffe2d411, name=이효석, age=40})
```



검색 결과를 순회하면서 각각의
Document 데이터를 출력!



```
public static void printAllUsers() { no usages new *  
    MongoCollection<Document> users = getCollection(colName: "users");  
    FindIterable<Document> doc = users.find();  
    Iterator itr = doc.iterator();  
    while (itr.hasNext()) {  
        System.out.println("==> findResultRow : " + itr.next());  
    }  
}
```

전체 도큐먼트 출력 작업이 반복 되므로
printAllUsers 라는 메서드로 빼기!

UPDATE





```
// UPDATE
System.out.println("=== UPDATE 시작 ===");
// 업데이트 연습을 위한 더미 데이터 삽입
Document user = new Document();
user.append("name", "이효석2");
user.append("age", 20);
users.insertOne(user);

printAllUsers();
```

수정을 위한 더미데이터 삽입 및
전체 도큐먼트 출력

===== 메인 메서드 시작 =====

=== UPDATE 시작 ===

```
==> findResultRow : Document({_id=669f15f4ea3b27423a66e022, name=이효석, age=40}}
==> findResultRow : Document({_id=669f18e68a1d6820ffe2d411, name=이효석, age=40}}
==> findResultRow : Document({_id=669f1b10ca153a4a88a0b0b9, name=이효석2, age=20}}}
```



```
import org.bson.conversions.Bson;  
import static com.mongodb.client.model.Filters.*;  
import static com.mongodb.client.model.Updates.*;
```

// MongoDB 의 쿼리는 결국 객체 형태로 전달 되므로 Bson 으로 만듭니다!

```
Bson updateFilter = eq(fieldName: "name", value: "이효석2");
```

수정 조건 쿼리 만들기

```
Bson updateOperation = set("age", 40);
```

업데이트할 내용 쿼리

// 업데이트 실행

```
users.updateOne(updateFilter, updateOperation);
```

업데이트 수행

```
System.out.println("사기꾼 검거");
```

```
printAllUsers();
```

전체 도큐먼트 확인하기

사기꾼 검거

```
==> findResultRow : Document({_id=669f15f4ea3b27423a66e022, name=이효석, age=40})  
==> findResultRow : Document({_id=669f18e68a1d6820ffe2d411, name=이효석, age=40})  
==> findResultRow : Document({_id=669f1b10ca153a4a88a0b0b9, name=이효석2, age=40})
```



수정 완료!



DELETE



```
// DELETE
System.out.println("=== DELETE 시작 ===");
Bson deleteFilter = eq( fieldName: "name", value: "이효석2");
users.deleteOne(deleteFilter);
System.out.println("사기꾼 Out");

printAllUsers();
```

삭제 조건 쿼리 만들기

삭제 수행

전체 도큐먼트 확인하기

===== 메인 메서드 시작 =====

=== DELETE 시작 ===

사기꾼 Out

==> findResultRow : Document({_id=669f18e68a1d6820ffe2d411, name=이효석, age=40})

==> findResultRow : Document({_id=669f1c55de46b6323c52bc6c, name=이효석, age=40})



삭제 완료!



POJO

(Plain Old Java Object)

적용

```
public class MongoUsersPojoDao { new *
    static MongoClient client; 3 usages
    static MongoDBDatabase database; 4 usages
    static {
        CodecProvider.pojoCodecProvider = PojoCodecProvider.builder().automatic(true).build();
        CodecRegistry.pojoCodecRegistry = fromRegistries(getDefaultCodecRegistry(), fromProviders(pojoCodecProvider));

        ConnectionString connectionString = new ConnectionString("mongodb://127.0.0.1:27017");
        client = MongoClient.create(connectionString);
        database = client.getDatabase(s: "users").withCodecRegistry(pojoCodecRegistry);
    }
}
```




이해보다 외우게 빠른 코드입니다!

자바의 객체를 MongoDB 의 문서로
변환시켜주는 작업을 하는
Codec 설정 코드 입니다!

```
public class MongoUsersPojoDao { new *
    static MongoClient client; 3 usages
    static MongoDBDatabase database; 4 usages
    static {
        CodecProvider.pojoCodecProvider = PojoCodecProvider.builder().automatic(true).build();
        CodecRegistry.pojoCodecRegistry = fromRegistries(getDefaultCodecRegistry(), fromProviders(pojoCodecProvider));

        ConnectionString connectionString = new ConnectionString("mongodb://127.0.0.1:27017");
        client = MongoClient.create(connectionString);
        database = client.getDatabase(s: "test3").withCodecRegistry(pojoCodecRegistry);
    }
}
```



사용 데이터베이스는
test3 로 변경



컬렉션 접속 정보를 가져올 때
위에서 만든 Codec 을 추가하여
해당 정보 교환 시 Codec 이 작동하도록 합니다!



```
import com.mongodb.ConnectionString;
import com.mongodb.client.*;
import com.mongodb.client.result.InsertManyResult;
import com.mongodb.client.result.InsertOneResult;
import org.bson.Document;
import org.bson.codecs.configuration.CodecProvider;
import org.bson.codecs.configuration.CodecRegistry;
import org.bson.codecs.pojo.PojoCodecProvider;
import org.bson.conversions.Bson;

import static com.mongodb.MongoClientSettings.getDefaultCodecRegistry;
import static com.mongodb.client.model.Filters.*;
import static com.mongodb.client.model.Updates.*;
import static org.bson.codecs.configuration.CodecRegistries.fromProviders;
import static org.bson.codecs.configuration.CodecRegistries.fromRegistries;

import java.util.ArrayList;
import java.util.List;
```

인텔리제이 입장에선 잘 안쓰는 애들이라
자동 추가가 잘 안되오니 일단 전달 드립니다!
임포트 자동으로 안되는 친구만 복붙해서 쓰세요!



```
public static void close() { client.close(); }

public static MongoDBDatabase getDatabase() { return database; }

public static MongoClient<Document> getCollection(String colName) { no usages new
    MongoClient<Document> collection = database.getCollection(colName);
    return collection;
}

public static <T> MongoClient<T> getCollection(String colName, Class<T> clazz) {
    MongoClient<T> collection = database.getCollection(colName, clazz);
    return collection;
}
```

이제는 자바 클래스(Like a Vo)를 사용해서
도큐먼트를 관리할 것이기 때문에 해당
도큐먼트의 타입을 제네릭으로 전달하여 +
코덱을 적용하여 collection 과 통신하는
getCollection 메서드!



```
1 package org.mongo;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import org.bson.types.ObjectId;
7
8 @Data 11 usages new *
9 @AllArgsConstructor
10 @NoArgsConstructor
11 public class UserPojo {
12     private ObjectId id;
13     private String name;
14     private int age;
15 }
16
```



UserPojo 생성!
롬복 사용!

UserPojo 를
사용할 것이므로
해당 클래스의 정보 전달

```
public static void main(String[] args) { new *
    MongoClient<UserPojo> collection = getCollection(colName: "users", UserPojo.class);

    System.out.println("===== 결과 시작 =====");

    // insertOne
    UserPojo newUser = new UserPojo(id: null, name: "이효석", age: 40);
    InsertOneResult result = collection.insertOne(newUser);
    System.out.println("처리 결과 : " + result.wasAcknowledged());
    System.out.println("삽입 된 문서의 _id : " + result.getInsertedId());
```



POJO

CREATE

```
public static void main(String[] args) { new *
    MongoClient<UserPojo> collection = getCollection(colName: "users", UserPojo.class);

    System.out.println("===== 결과 시작 =====");

    // insertOne
    UserPojo newUser = new UserPojo(id: null, name: "이효석", age: 40);
    InsertOneResult result = collection.insertOne(newUser);
    System.out.println("처리 결과 : " + result.wasAcknowledged());
    System.out.println("삽입 된 문서의 _id : " + result.getInsertedId());
```



UserPojo 타입으로
새로운 회원을 생성

```
public static void main(String[] args) { new *
    MongoClient<UserPojo> collection = getCollection(colName: "users", UserPojo.class);

    System.out.println("===== 결과 시작 =====");

    // insertOne
    UserPojo newUser = new UserPojo(id: null, name: "이효석", age: 40);
    InsertOneResult result = collection.insertOne(newUser);
    System.out.println("처리 결과 : " + result.wasAcknowledged());
    System.out.println("삽입 된 문서의 _id : " + result.getInsertedId());
```




전달 받은 컬렉션에 insertOne 메서드로
UserPojo 로 만든 객체를 전달!

```
public static void main(String[] args) { new *
    MongoClient<UserPojo> collection = getCollection(colName: "users", UserPojo.class);

    System.out.println("===== 결과 시작 =====");

    // insertOne
    UserPojo newUser = new UserPojo(id: null, name: "이효석", age: 40);
    InsertOneResult result = collection.insertOne(newUser);
    System.out.println("처리 결과 : " + result.wasAcknowledged());
    System.out.println("삽입 된 문서의 _id : " + result.getInsertedId());
```



insertOne 의 결과는 해당 결과에 맞는 객체를 리턴
따라서, InsertOneResult 라는 타입으로 결과를 받아야 합니다!



===== 결과 시작 =====

처리 결과 : true

삽입 된 문서의 _id : BsonObjectId{value=669f247274a3e9455e6c4b15}

+ ADD DATA ▼

EXPORT DATA ▼

UPDATE

DELETE

_id: ObjectId('669f247274a3e9455e6c4b15')

age : 40

name : "이효석"



POJO

여러 개 CREATE


```
// insertMany
```

```
List<UserPojo> newTodos = Arrays.asList(
```

```
    new UserPojo(id: null, name: "이효석2", age: 40),
```

```
    new UserPojo(id: null, name: "이효석3", age: 40),
```

```
    new UserPojo(id: null, name: "이효석4", age: 40)
```

```
);
```

```
InsertManyResult manyResult = collection.insertMany(newTodos);
```

```
System.out.println("처리 결과 : " + manyResult.wasAcknowledged());
```

```
System.out.println("삽입 된 도큐먼트의 _id : " + manyResult.getInsertedIds());
```

UserPojo 타입으로
리스트를 생성

insertMany 메서드에
리스트를 전달!

insertMany 의 결과는 해당 결과에 맞는 객체를 리턴
따라서, insertManyResult 라는 타입으로 결과를 받아야 합니다!



===== 결과 시작 =====

처리 결과 : true

삽입 된 문서의 _id : {0=BsonObjectId{value=669f251cfa5839022647f779}, 1=BsonObjectId{value=669f251cfa5839022647f77a}, 2=BsonObjectId{value=669f251cfa5839022647f77b}}

+ ADD DATA ▾

📄 EXPORT DATA ▾

✎ UPDATE

🗑 DELETE

_id: ObjectId('669f247274a3e9455e6c4b15')
age : 40
name : "이효석"

_id: ObjectId('669f251cfa5839022647f779')
age : 40
name : "이효석2"

_id: ObjectId('669f251cfa5839022647f77a')
age : 40
name : "이효석3"

_id: ObjectId('669f251cfa5839022647f77b')
age : 40
name : "이효석4"



POJO

READ

```
// find()
List<UserPojo> users = new ArrayList<>();
collection.find().into(users);
for(UserPojo user : users) {
    System.out.println(user);
}
```

UserPojo 객체를 가지는
리스트로 검색 결과를 저장!

검색 결과 저장

검색 결과 리스트를 순회하면서 출력!



===== 결과 시작 =====

UserPojo(id=669f247274a3e9455e6c4b15, name=이효석, age=40)

UserPojo(id=669f251cfa5839022647f779, name=이효석2, age=40)

UserPojo(id=669f251cfa5839022647f77a, name=이효석3, age=40)

UserPojo(id=669f251cfa5839022647f77b, name=이효석4, age=40)



POJO UPDATE

```
// updateOne
```

```
Bson updateQuery = eq(fieldName: "name", value: "이효석");
```

```
Bson updateOperation = set("age", 41);
```

```
UpdateResult updateResult = collection.updateOne(updateQuery, updateOperation);
```

```
System.out.println("수정된 문서의 수 : " + updateResult.getModifiedCount());
```

업데이트는 이전과 동일하게
Bson 으로 쿼리 만들어서 전달!

update 의 결과는 해당 결과에 맞는 객체를 리턴
따라서, updateResult 라는 타입으로 결과를 받아야 합니다!



===== 결과 시작 =====

수정된 도큐먼트의 수 : 1

`_id: ObjectId('669f247274a3e9455e6c4b15')`

`age : 41`

`name : "이효석"`



POJO

DELETE

삭제도 이전과 동일하게
Bson 으로 쿼리 만들어서 전달!

```
// deleteOne  
Bson deleteQuery = eq(fieldName: "name", value: "이효석");  
DeleteResult deleteResult = collection.deleteOne(deleteQuery);  
System.out.println("삭제된 문서의 수 : " + deleteResult.getDeletedCount());
```

delete 의 결과는 해당 결과에 맞는 객체를 리턴
따라서, updateResult 라는 타입으로 결과를 받아야 합니다!



===== 결과 시작 =====

삭제된 문서의 수 : 1



ADD DATA



EXPORT DATA



UPDATE



```
_id: ObjectId('669f251cfa5839022647f779')
age : 40
name : "이호석2"
```

```
_id: ObjectId('669f251cfa5839022647f77a')
age : 40
name : "이호석3"
```

```
_id: ObjectId('669f251cfa5839022647f77b')
age : 40
name : "이호석4"
```