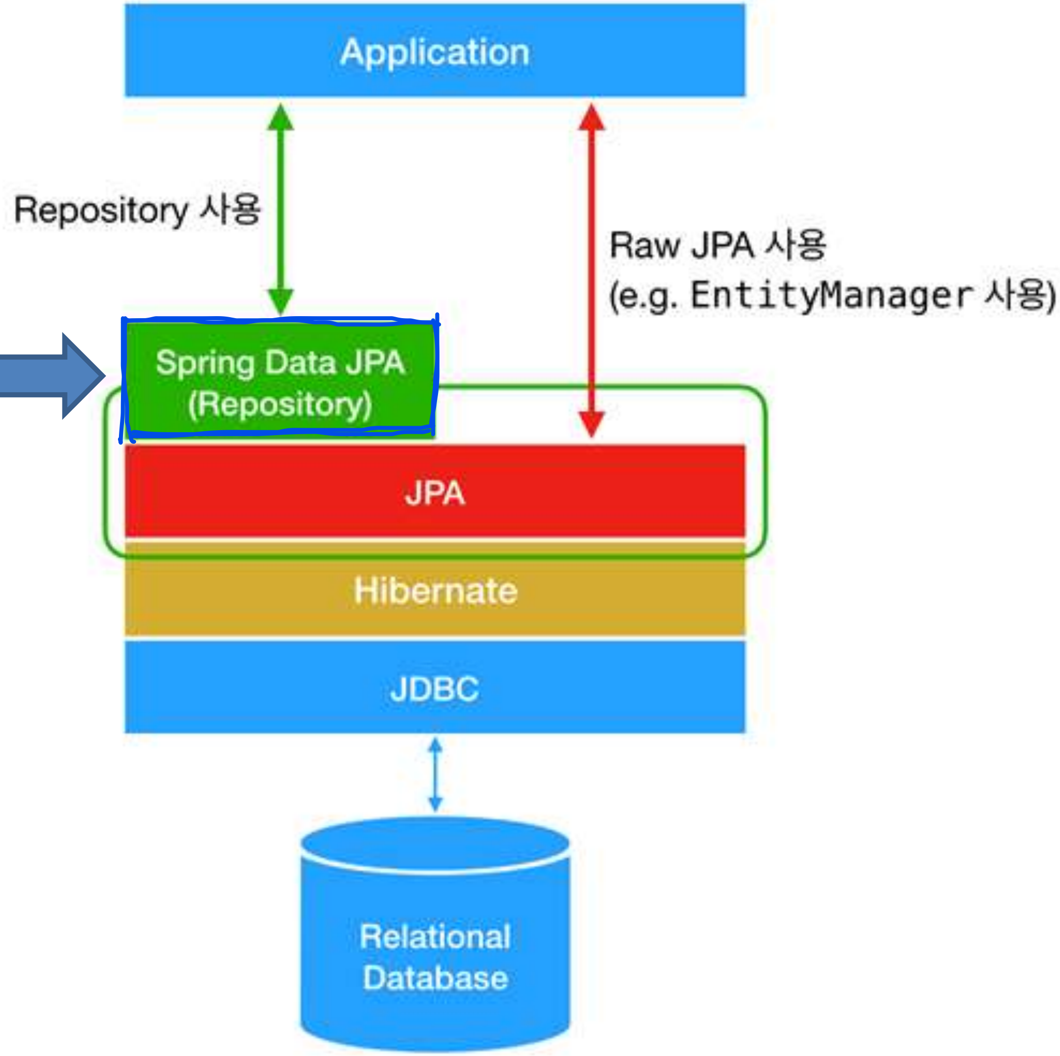
Working with Spring Data JPA

Data Access



Spring Data JPA를 알아볼 것이다

1. Creating DAO

Programming with JPA/Hibernate

```
@Repository
@Transactional
public class OfferDao {

    @PersistenceContext
    private EntityManager entityManager;

    public Product createProduct(Product product) {
        entityManager.persist(product);
        return product;
    }

    public Product findProduct(Long id) {
        return entityManager.find(Product.class, id);
    }

    public Product updateProduct(Product product) {
        return entityManager.merge(product);
    }

    public void deleteProduct(Long id) {
        Product product = findProduct(id);
        if (product != null) {
            entityManager.remove(product);
        }
    }
}
```

C

R

U

D

The Problem

- We saw how to create a DAO for Product

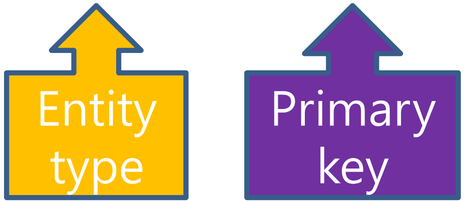
- What if we need to create a DAO for another entity?
  - Customer, Student, Book, …

- Do we have to repeat all of the same code again?

The Problem

- You may have noticed a pattern with creating DAOs

```
public Product findProduct(Long id) {

        return entityManager.find(Product.class, id);

}
```



Entity type

Primary key

Only difference is the entity type and primary key

Approach

- I wish we could tell Spring:

복붙하기 귀찮으니까 날 위해서 Dao를 만들어줘

Create a Dao for me

I'll simply plug in my entity type and primary key

Give me all of the basic CRUD features for free

엔티티 타입과 기본키를 줄테니 알아서 만들어줘

Approach

Entity: Offer

Primary key: Integer

findAll()
findById(…)
save(…)
deleteById(…)
… others…

CRUD methods

Spring Data JPA

- **Spring Data JPA** is the solution

- Create a DAO if you plug in your entity type and primary key

- Spring will give you a CRUD implementation for free
  - Helps to minimize boiler-plate DAO code

2. Spring Data JPA

관계형 DB를 작업할 때 유용한 모듈

- Spring Data JPA is one of the modules for working with relational databases using JPA

- Spring Data JPA provides various repository interfaces, such as CrudRepository, PagingAndSortingRepository, JpaRepository 의 인터페이스를 제공함
  - Provide out-of-the-box support for CRUD operations, as well as pagination and sorting

이 인터페이스들은 CRUD operations 를 제공함

Spring Data JPA

엔티티 타입과 기본키, 그리고
어떤 레포지토리한테서 상속받을지

Entity Type

Primary Key

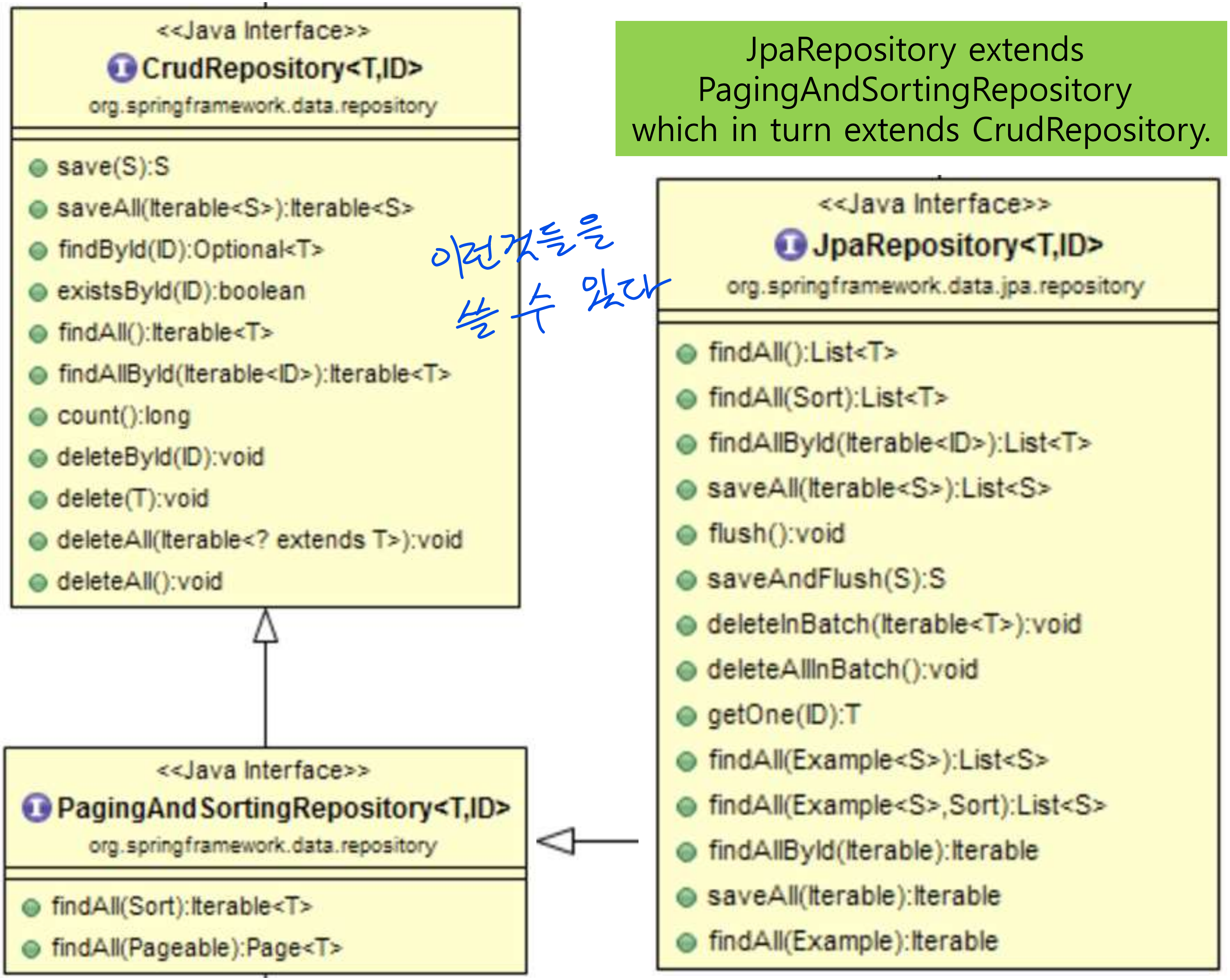
```
public interface ProductRepository extends CrudRepository<Product, Integer> {

}
```

No need for Implementation class

findAll()
findById(…)
save(…)
deleteById(…)

Get these methods for free

JpaRepository extends
PagingAndSortingRepository
which in turn extends CrudRepository.



<<Java Interface>>
CrudRepository<T,ID>
org.springframework.data.repository

- save(S):S
- saveAll(Iterable<S>):Iterable<S>
- findById(ID):Optional<T>
- existsById(ID):boolean
- findAll():Iterable<T>
- findAllById(Iterable<ID>):Iterable<T>
- count():long
- deleteById(ID):void
- delete(T):void
- deleteAll(Iterable<? extends T>):void
- deleteAll():void

<<Java Interface>>
PagingAndSortingRepository<T,ID>
org.springframework.data.repository

- findAll(Sort):Iterable<T>
- findAll(Pageable):Page<T>

<<Java Interface>>
JpaRepository<T,ID>
org.springframework.data.jpa.repository

- findAll():List<T>
- findAll(Sort):List<T>
- findAllById(Iterable<ID>):List<T>
- saveAll(Iterable<S>):List<S>
- flush():void
- saveAndFlush(S):S
- deleteInBatch(Iterable<T>):void
- deleteAllInBatch():void
- getOne(ID):T
- findAll(Example<S>):List<S>
- findAll(Example<S>,Sort):List<S>
- findAllById(Iterable):Iterable
- saveAll(Iterable):Iterable
- findAll(Example):Iterable

이런것들을
쓸 수 있다

Spring Data JPA

- Their main functions are:
  - CrudRepository mainly provides CRUD functions
  - PagingAndSortingRepository provide methods to do pagination and sorting records
  - JpaRepository provides some JPA related method such as flushing the persistence context and delete record in a batch

1) CrudRepository Interface

Modifier and Type	Method and Description
long	count() Returns the number of entities available.
void	delete(T entity) Deletes a given entity.
void	deleteAll() Deletes all entities managed by the repository.
void	deleteAll(Iterable<? extends T> entities) Deletes the given entities.
void	deleteAllById(Iterable<? extends ID> ids) Deletes all instances of the type T with the given IDs.
void	deleteById(ID id) Deletes the entity with the given id.
boolean	existsById(ID id) Returns whether an entity with the given id exists.
Iterable<T>	findAll() Returns all instances of the type.
Iterable<T>	findAllById(Iterable<ID> ids) Returns all instances of the type T with the given IDs.
Optional<T>	findById(ID id) Retrieves an entity by its id.
<S extends T> S	save(S entity) Saves a given entity.
<S extends T> Iterable<S>	saveAll(Iterable<S> entities) Saves all given entities.

2) PagingAndSortingRepository Interface

```
public interface PagingAndSortingRepository<T, ID extends Serializable>
            extends CrudRepository<T, ID> {

            Iterable<T> findAll(Sort sort);
            Page<T> findAll(Pageable pageable);
}
```

PagingAndSortingRepository Interface

```
public interface ProductRepository extends PagingAndSortingRepository<Product, Long> {
}
```

```
@Service
public class ProductService {
    @Autowired
    private ProductRepository productRepository;

    public List<Product> findAllProductsSortedByPrice() {
        Sort sort = Sort.by(Sort.Direction.DESC, "price");
        return productRepository.findAll(sort);
    }

    public Page<Product> findProductsByPage(int page, int size) {
            Pageable pageable =
                        PageRequest.of(page, size, Sort.by(Sort.Direction.ASC, "name"));
            return productRepository.findAll(pageable);
    }
}
```

price 라는 필드를
기준으로 내림차순 ...?

the page number starts at zero

3) JpaRepository Interface

- JpaRepository provides CRUD and pagination operations, along with JPA related methods such as flushing the persistence context and delete records in a batch

- Return type of saveAll() and findAll() method is a List

3. How to Use Spring Data JPA interfaces

Without Spring Data JPA

기존 코딩 방식

```
@Repository
public class ProductRepository {

    @PersistenceContext
    private EntityManager entityManager;

    @Transactional
    public Product save(Product product) {
        if (product.getId() == null) {
            entityManager.persist(product);
            return product;
        } else {
            return entityManager.merge(product);
        }
    }

    public Product findById(Integer id) {
        return entityManager.find(Product.class, id);
    }
```

```
 @Transactional
    public void deleteById(Integer id) {
        Product product = findById(id);
        if (product != null) {
            entityManager.remove(product);
        }
    }

public boolean existsById(Integer id) {
        Product product = findById(id);
        return product != null;
    }

    ...
}
```

With Spring Data JPA

를 쓰면 간단해진다

```
public interface ProductRepository extends JpaRepository<Product, Integer> {

}
```

How to Use Spring Data JPA interfaces

Service layer

```
@Service
@Transactional
public class ProductService {

    @Autowired
    private ProductRepository repo;

    public Product get(long id) { return repo.findById(id).get();    }
    public List<Product> listAll() {  return repo.findAll();    }
    public void save(Product product) { repo.save(product);    }
    public void delete(long id) { repo.deleteById(id);
    }
}
```

id 대신 name으로 검색이 가능할까? 가능하다

4. Query Method

- What if you want to perform a search based on a keyword like "name" instead of searching by "id"?
- Spring Data JPA not only provides CRUD operations out-of-the-box, but it also supports dynamic query generation based on the method names 메소드 이름을 기반으로 동적으로 쿼리를 지원해주기 때문
- For example:
  - By defining a User findByEmail(String email) method, Spring Data will automatically generate the query with a where clause, as in "where email = ?1"
  - By defining a User findByEmailAndPassword(String email, String password) method, Spring Data will automatically generate the query with a where clause, as in "where email = ?1 and password=?2"
- If query method return more than one result, we can return the following types: List<T>, Page<T>

Query Method

```
public interface MemberRepository
                 extends JpaRepository<Member, Long> {

       List<Customer> findByUsername(String username);

}
```

이러면, Username을 보고 알아서
아래 코드를 짜준다

find + By + 변수 이름

In Spring Data JPA, you just declare the method in the repository interface,
and Spring Data JPA automatically creates the necessary query and implementation

```
public List<Member> findByUsername(String username) {

    TypedQuery<Member> query = entityManager.createQuery(
           "SELECT m FROM Member m WHERE m.username = ?1", Member.class);
    query.setParameter(1, username);
    return query.getResultList();
}
```

Supported keywords inside method names

단, 규칙이 있다

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull, Null	findByAge(Is)Null	… where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 한성* (parameter bound with appended %) ?1%
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 *한성 (parameter bound with prepended %) %?1
Containing	findByFirstnameContaining	… where x.firstname like ?1 *한성* (parameter bound wrapped in %) %?1%
…	…	…

예시

Query Method

JPA

```
public List<Member> findByUsernameAndAgeGreaterThan(String username, int age) {

    TypedQuery<Member> query = entityManager.createQuery(
        "SELECT m FROM Member m WHERE m.username = ?1 AND m.age > ?2", Member.class);
    query.setParameter(1, username);
    query.setParameter(2, age);
    return query.getResultList();
}
```

Spring Data JPA

```
public interface MemberRepository extends JpaRepository<Member, Long> {

    List<Member> findByUsernameAndAgeGreaterThan(String username, int age);
}
```

Use Cases

다수개의 조건 기술
게시판에서 title 혹은 content에서 특정 단어가 포함된 글 목록 조회

```
public interface BoardRepository extends JpaRepositoty<Board, Long> {

    List<Board> findByTitleContainingOrContentContaining(String title, String content);
}
```

```
@SpringBootTest
public class QueryMethodTest {

    @Autowired
    private BoardRepository repo;

    @Test
    public void testFindByTitleContainingOrContentContaining() {

       List<Board> boardList= repo.findByTitleContainingOrContentContaining(“17”, “17”);

       for(Board board: boardList) {  System.out.println(“→” + board.toString() ); }
}
```

Pagination
게시판에서 title변수에 “한성대”이라는 검색어가 포함된 게시글 목록 검색
페이지 단위로 조회

```
public interface BoardRepository  extends JpaRepositoty<Board, Long> {

    List<Board> findByTitleContaining(String searchKeyword, Pageable paging);
}
```

(Page number, Page size)

```
@Test
public void testFindByTitleContaining() {

      Pageable paging = PageRequest.of(0, 5);
      List<Board> boardList= repo.findByTitleContaining(“한성대”, paging);

      for(Board board: boardList) {
            System.out.println(“→” + board.toString() );
    }
}
```

Pagination and Sort
게시판에서 title변수에 "한성대"이라는 검색어가 포함된 게시글 목록 검색
"seq" 변수에 따라 내림차순으로 정렬

```
public interface BoardRepository extends JpaRepositoty<Board, Long> {

    List<Board> findByTitleContaining(String searchKeyword, Pageable paging);
}


@Test
public void testFindByTitleContaining( ) {

     Pageable paging = PageRequest.of(0,5, Sort.Direction.DESC, "seq");
     List<Board> boardList= repo.findByTitleContaining("한성대", paging);

     for(Board board: boardList) {
          System.out.println("→" + board.toString() );
     }
```

Return Type: List<T> -> Page<T>
Page<T> 객체는 페이징 처리할 때 다양한 정보를 추가로 제공

```
public interface BoardRepository  extends CrudRepositoty<Board, Long> {

   Page<Board> findByTitleContaining(String searchKeyword, Pageable paging);
}


@Test
public void testFindByTitleContaining() {

      Pageable paging = PageRequest.of(0,5, Sort.Direction.DESC, “seq”);
      Page<Board> pageInfo= repo.findByTitleContaining(“한성대”, paging);

      System.out.println(“Page size: ”    + pageInfo.getSize() );
      System.out.println(“Total Pages: ” + pageInfo.getTotalPages() );
      System.out.println(“Total Count: ” + pageInfo.getTotalElements() );

      List<Board> boardList=pageInfo.getContent();

      for(Board board: boardList) {
            System.out.println(“→” + board.toString() );
      }
}
```

```
5
40
200
```

5. @Query Annotation

- Sometimes you may not be able to express your criteria using method names
- Spring Data provides flexibility to configure the query explicitly using the @Query annotation

이렇게 쓰면, 메서드를 만들어 준다

```
@Query("select u from User u where u.email=?1 and u.password=?2)
User findByEmailAndPassword(String email, String password);
```

복잡한 경우, 내가 쿼리문을 직접 줘야할 것 같은 경우에는 이 @Query를 써서 명시할 수 있다

참고로 SQL 아니고 JPQL 이다

@Query: JPQL

Positional parameter binding

```
public interface BoardRepository extends JpaRepositoty<Board, Long> {

    @Query(“SELECT b From Board b ”
                + “WHERE b.title like %?1% ORDER BY b.seq DESC”)

    List<Board> queryAnnotationTest1(String searchKeyword);
}
```

Note: JPQL uses Entity and Field name
Case Sensitive

```
@Test
Public void QueryAnnotationTest() {

     List<Board> boardList=   repo.queryAnnotationTest1(“한성대”);

      for(Board board: boardList) {
             System.out.println(“→” + board.toString() );
      }
```

@Query: JPQL

Named Parameter binding

```
public interface BoardRepository extends CrudRepositoty<Board, Long> {

    @Query(“SELECT b From Board b WHERE b.title like %:searchKeyWord% “
        + “ORDER BY b.seq DESC”)

    List<Board>
            queryAnnotationTest1(@Param(“searchKeyword”)String searchKeyword);
}
```

@Query: native

- We can use also native SQL to define our query SQL을 꼭 써야겠으면 true로

```
public interface BoardRepository extends JpaRepositoty<Board, Long> {

    @Query(value="select seq, title from board "
                + "where title like '%' || ?1 '%' "
                + "order by seq dec", nativeQuery=true);

    List<Object[]> queryAnnotationTest2(String searchKeyword);
}
```