1. 기본 CRUD Test

① User 생성 테스트 코드 작성

- Ctrl+Shift+T : 테스트 코드 생성

```
@Test
@DisplayName("User 입력 테스트")
void userInputTest(){
    Users users = Users.builder()
            .name("김형민")
            .gender(Gender.Male)
            .createdAt(LocalDateTime.now())
            .updatedAt(LocalDateTime.now())
            .email("a@naver.com")
            .likeColor("Red")
            .build();
    usersRepository.save(users);
    System.out.println("---------------" + usersRepository.count());
}
```

② 전체 레코드의 수 가져오고 id = 2 레코드 존재 유무 확인하기

```
@Test
@DisplayName("전체레코드 수와 id = 2L 존재 확인")
void userCountAndExistTest(){
    Long cnt = usersRepository.count();
    System.out.println("전체 레코드 수 = " + cnt);

    Boolean exist = usersRepository.existsById(2L);
    System.out.println("id = 2 가 있는가? ----  " + exist );
}
```

③ 특정 레코드(id=1L) 삭제하기

```
@Test
    @DisplayName("id = 1L 삭제한 후 존재 확인")
```

```
    void userDeleteAndExistTest(){
        usersRepository.deleteById(1L);

        Boolean exist = usersRepository.existsById(1L);
        System.out.println("id = 1 가 있는가? ----  " + exist );
    }
```

④ 레코드 업데이트 하기

```
@Test
    @DisplayName("Update : id = 1L, color = '빨강'")
    void userUpdateTest(){

        if(usersRepository.findById(1L).isPresent()){
            Optional<Users> oldUser = usersRepository.findById(1L);
            System.out.println("Old User = " + oldUser);
            Users newUser = oldUser.get();
            newUser.setLikeColor("빨강");
            usersRepository.save(newUser);
            System.out.println("Old User = " + newUser);
        }
    }
```

Query Method 생성하기

1. 이름을 통해서 User를 가져오는 Method 생성

- 단일 객체 리턴인 경우 & 중복 객체 리턴일 경우(List로 바꿔서 처리...)

```
List<Users> findByName(String name);
```

--> Map, Optional 등 개발스펙에 맞추어 리턴 타입을 결정해 준다.

- Test 및 결과 출력

```
    @Test
    @DisplayName("findByName 메소드 테스트")
    void findByNameTest(){
        String name = "Torrin";
        //메소드 레퍼런스
        //Java 8에서 도입된 메소드 레퍼런스(Method Reference)는 Lambda 표현식을
        // 더 간단하게 표현하는 방법
        usersRepository.findByName(name).forEach(System.out::println);

        //람다식
        usersRepository.findByName("Damon").forEach(damon->
                                                    System.out.println(damon));
    }
```

2. 상위 3개 같은 색상정보 레코드 가져오기

```
List<Users> findTop3ByLikeColor(String color);
```

```
    @Test
    @DisplayName("findTop3ByLikeColor 테스트")
    void findTop3ByLikeColorTest(){
        usersRepository.findTop3ByLikeColor("Pink")
                .forEach(user-> System.out.println(user));
    }
```

3. 한 개 이상의 조건으로 검색하기(And, Or)

① gender And color로 비교

```
//Gender:Female And Color:Red
    List<Users> findByGenderAndLikeColor(Gender gender, String color);
```

```
@Test
    @DisplayName("findByGenderAndLikeColor 테스트")
    void findByGenderAndLikeColorTest(){
        usersRepository.findByGenderAndLikeColor( Gender.Male, "Pink")
                .forEach(user-> System.out.println(user));
    }
```

② gender Or color로 비교

```
//Gender:Female Or Color:Red
    List<Users> findByGenderOrLikeColor(Gender gender, String color);
```

```
@Test
    @DisplayName("findByGenderOrLikeColor 테스트")
    void findByGenderOrLikeColorTest(){
        usersRepository.findByGenderOrLikeColor(Gender.Female, "Pink")
                .forEach(user-> System.out.println(user));
    }
```

4. 범위로 검색하기(After, Before) --- 대부분 날짜와 시간의 경우에 한정지어 사용

--> 어제 이후 자료 검색하기("=" 가 포함되지 않으므로 주의)

```
List<Users> findByCreatedAtAfter(LocalDateTime searchDate);
```

```
    @Test
    @DisplayName("findByCreatdAtAfter 테스트")
    void findByCreatdAtAfterTest(){
        usersRepository.findByCreatedAtAfter(LocalDateTime.now().minusDays(7L))
                .forEach(user-> System.out.println(user));
    }
```

```
//범위로 검색하기(7일 전 자료 가져오기 - 당일 포함)
    List<Users> findByCreatedAtGreaterThanEqual(LocalDateTime searchDate);
```

```
@Test
    @DisplayName("findByCreatedAtGreaterThanEqual 테스트")
    void findByCreatedAtGreaterThanEqualTest(){

usersRepository.findByCreatedAtGreaterThanEqual(LocalDateTime.now().minusDays(15L))
                    .forEach(user-> System.out.println(user));
    }
```

-- Between 으로 자료 검색하기(얘는 등호 포함 된다....)

```
//범위로 검색하기 - Between(등호 포함)
    List<Users> findByCreatedAtBetween(LocalDateTime startDate,LocalDateTime
endDate);
    List<Users> findByIdBetween(Long startId,Long endId);
```

--> 테스트 결과

```
@Test
    @DisplayName("findByCreatedAtBetween 테스트")
    void findByCreatedAtBetween(){
        usersRepository.findByCreatedAtBetween(LocalDateTime.now().minusDays(31L),
LocalDateTime.now())
                    .forEach(user-> System.out.println(user));

        usersRepository.findByIdBetween(1L, 5L)
                    .forEach(user-> System.out.println(user));
    }
```

-- Null / IsNotNull 사용

```
//Null or IsNotNull 검색하기
    List<Users> findByUpdatedAtIsNotNull();
```

```
@Test
    @DisplayName("findByUpdatedAtIsNotNull 테스트")
```

```
void findByUpdatedAtIsNotNull(){
    System.out.println("--- InNotNull Count : " +
            usersRepository.findByUpdatedAtIsNotNull().stream().count());
}
```

-- IN 구문(많이 자주 사용됨)

```
//In 구문
List<Users> findByLikeColorIn(List<String> color);
```

---> 변수명 구성 시 단수/복수 개념을 많이 사용하는 편

```
@Test
@DisplayName("findByLikeColorIn 테스트")
void findByLikeColorInTest(){
    usersRepository.findByLikeColorIn(Lists.newArrayList("Red", "Orange"))
            .forEach(users -> System.out.println(users));
}
```

```
where
    u1_0.name in(?,?)
```

--> subQuery 실행결과를 In구문의 인자로 전달하는 경우가 많음.

5. 문자열 관련 Query Method

- StartingWith , EndingWith, Contains, Like 구문을 사용
- 그러나 Like의 경우 파라미터를 넘겨받아 사용할 경우 % + "변수명" + % 로 사용할 경우가 많아서 준비된 Method를 사용함.

```
//문자열 관련 Query Method
List<Users> findByNameStartingWith(String name);
List<Users> findByNameEndingWith(String name);
List<Users> findByNameContains(String name);
List<Users> findByNameLike(String name);
```

```
@Test
    @DisplayName("StringSearch 테스트")
    void stringSearechTest(){
        usersRepository.findByNameStartingWith("D")
                .forEach(users -> System.out.println("D로 시작 : " + users));

        usersRepository.findByNameEndingWith("s")
                .forEach(users -> System.out.println("s로 끝 : " + users));

        usersRepository.findByNameContains("k")
                .forEach(users -> System.out.println("k 포함 : " + users));

        usersRepository.findByNameLike("%k%")
                .forEach(users -> System.out.println("k 포함 : " + users));
    }
```

6. Sorting 하고 순서대로 데이터 추출하기(OrderByAsc, OrderByDesc)

```
//Sorting하고 순서대로 출력하기
    //id : 1~10, 이름의 내림차순
    List<Users> findByIdBetweenOrderByNameDesc(Long start, Long end);
```

```
@Test
    @DisplayName("findByIdBetweenOrderByNameDesc 테스트")
    void findByIdBetweenOrderByNameDescTest(){
        usersRepository.findByIdBetweenOrderByNameDesc(1L, 5L)
                .forEach(users -> System.out.println(users));
    }
```

-- 한 가지 컬럼 이상의 조건으로 검색해서 레코드 가져올 경우

```
    //Orange 색상 상위 10개 검색해서 Gender 오름차순, Create_At 내림차순
    List<Users> findTop10ByLikeColorOrderByGenderDescCreatedAtAsc(String color);
```

```
    @Test
    @DisplayName("findTop10OrderByNameAscCreatedAtDesc 테스트")
    void findTop10OrderByNameAscCreatedAtDescTest(){

usersRepository.findTop10ByLikeColorOrderByGenderDescCreatedAtAsc("Orange")
                    .forEach(users -> System.out.println(users));
    }
```

-- 정렬 처리 하기

```
    //Orange 색상 검색해서 Gender 오름차순, Create_At 내림차순
    // sort 조건 직접 적용하기
    List<Users> findByLikeColor(String color, Sort sort);
```

```
    @Test
    @DisplayName("findByLikeColor 테스트")
    void findByLikeColorTest(){
        usersRepository.findByLikeColor("Orange"
                            , Sort.by(Sort.Order.desc("gender"),
                                    Sort.Order.asc("createdAt")))
                    .forEach(users -> System.out.println(users));
    }
```

※ 쿼리 Order 구문을 별도로 처리하는 방법

```
    @Test
    @DisplayName("Sort Method 테스트")
    void findSortMethodTest(){
        usersRepository.findByLikeColor("Orange", getSort())
                    .forEach(users -> System.out.println(users));
    }
    //Query Order 구문을 별도 처리하는 방법
    private Sort getSort(){
        return  Sort.by(
          Sort.Order.desc("gender"),
```

```
                Sort.Order.asc("createdAt")
            );
    }
```