07/28 회의

TDD

```
public static ArrayList<Integer> answers = new ArrayList<>();

public static ArrayList<Integer> numbers = new ArrayList<>();
```

```
메모리구조를 알아야하는 이유!!!

-> 단위테스트는 성공. 통합테스트는 실패. 이유는???
```

static 은 클래스 로드 시점에 딱 한번만 초기화 되다는 사실. 결국 해당 클래스가 여러번 만들어져도 answers, numbers 는 초기화 되지 않는다.

TEST Code 함께 리팩토링

```
16  //  - [0] 같은 수가 전혀 없으면 낫싱이다.
17  public class BaseballTest {
18
19      @Test
20      @DisplayName("서로다른 임의의 수 3개 생성")
21  -   public void init_random_number() {
22  -       Baseball game = new Baseball();
23          game.createRandomNumber();
24
        assertThat(Baseball.answers.get(0)).isNotEqualTo(Baseball.answers.get(1)).isNotEqualTo(Baseball.answers.get(2));
25          assertThat(Baseball.answers.get(1)).isNotEqualTo(Baseball.answers.get(2));
```

```
18  //  - [0] 같은 수가 전혀 없으면 낫싱이다.
19  public class BaseballTest {
20
21  +   Baseball game = new Baseball();
22  +
23  +   @AfterEach
24  +   public void init_answer() {
25  +       Baseball.answers = new ArrayList<>();
26  +       Baseball.numbers = new ArrayList<>();
27  +   }
28  +
29  +   @BeforeEach
30  +   public void setting() {
31  +       Baseball.answers.add(1);
32  +       Baseball.answers.add(2);
33  +       Baseball.answers.add(3);
34  +       game.createUserInput("123");
35  +   }
36  +
37      @Test
38      @DisplayName("서로다른 임의의 수 3개 생성")
39  +   public void init_random_number() throws InterruptedException {
40          game.createRandomNumber();
41
        assertThat(Baseball.answers.get(0)).isNotEqualTo(Baseball.answers.get(1)).isNotEqualTo(Baseball.answers.get(2));
42          assertThat(Baseball.answers.get(1)).isNotEqualTo(Baseball.answers.get(2));
```

```
28      @Test
29      @DisplayName("컴퓨터의 숫자와 사용자의 입력값이 몇개가 같은지 확인")
30      public void compare_number_computer_user() {
31  -       Baseball game = new Baseball();
32  -       Baseball.answers.add(1);
33  -       Baseball.answers.add(2);
34  -       Baseball.answers.add(3);
35  -       game.createUserInput("123");
36          int result = game.compare();
37          assertThat(result).isEqualTo(3);
38      }
39
40      @Test
41      @DisplayName("strike 수 구하기")
42      public void getStrike() {
43  -       Baseball game = new Baseball();
44  -       Baseball.answers.add(1);
45  -       Baseball.answers.add(2);
46  -       Baseball.answers.add(3);
47  -       game.createUserInput("123");
48          int result = game.getStrike();
49          assertThat(result).isEqualTo(3);
50      }
51
52      @Test
53      @DisplayName("ball 수 구하기")
54      public void getBall() {
55  -       Baseball game = new Baseball();
56  -       Baseball.answers.add(1);
57  -       Baseball.answers.add(2);
58  -       Baseball.answers.add(3);
59  -       game.createUserInput("123");
60          int result = game.getBall();
61          assertThat(result).isEqualTo(0);
62      }
63
```

```
45      @Test
46      @DisplayName("컴퓨터의 숫자와 사용자의 입력값이 몇개가 같은지 확인")
47      public void compare_number_computer_user() {
48          int result = game.compare();
49          assertThat(result).isEqualTo(3);
50      }
51
52      @Test
53      @DisplayName("strike 수 구하기")
54      public void getStrike() {
55          int result = game.getStrike();
56          assertThat(result).isEqualTo(3);
57      }
58
59      @Test
60      @DisplayName("ball 수 구하기")
61      public void getBall() {
62          int result = game.getBall();
63          assertThat(result).isEqualTo(0);
64      }
65
```

```
64      @Test
65      @DisplayName("결과 값 print")
66  -   public void getResult() {
67  -       Baseball game = new Baseball();
68  -       Baseball.answers.add(1);
69  -       Baseball.answers.add(2);
70  -       Baseball.answers.add(3);
71  -       game.createUserInput("456");
72          game.makeResult();
73      }
74
```

```
66      @Test
67      @DisplayName("결과 값 print")
68  +   public void getResult() throws Exception {
69          game.makeResult();
70      }
71
```

객체지향 사실과 오해

Interface 로 얻을 수 있는 이점

인터페이스가 아닌 직접 클래스를 생성하면 확장,변경용이성 이 매우 떨어진다.
예를 들어 UserRepository 가 있고 우리가 JPA 를 통해 구현 되어 있는걸 Querydsl 로 옮기는 작업을 한다고 했을때, 다른 도메인의 계층해서 주입받아 사용하는 로직들을 모두 변경해야한다.
-> SUWIKI 프로젝트 예시를 통해 이해.

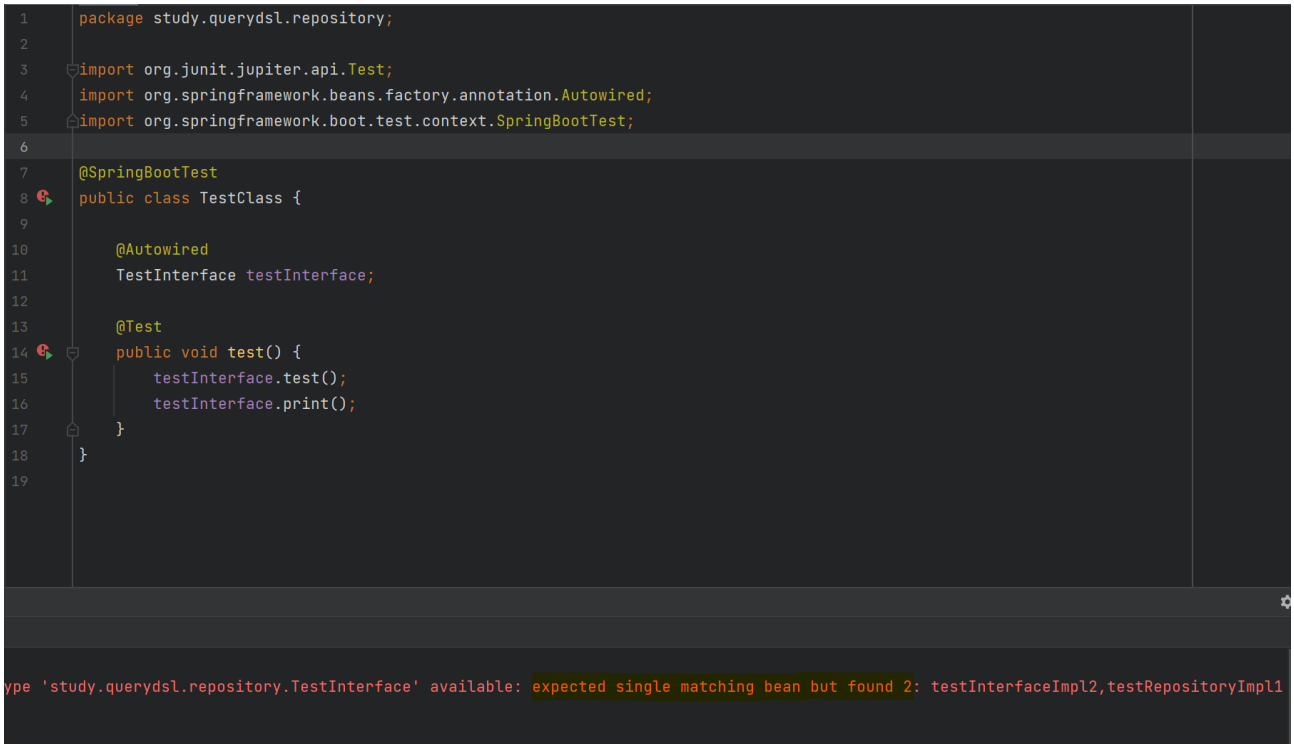
인터페이스를 통해 외부에 공개 되는 내용과 실제 구현되는 내부로직을 철저히 구분하여, 구현체가 바뀌어도 가져다 사용하는 객체는 바꾸지 않게끔 설계 가능하다.

구현체가 2개가 존재한다면 인터페이스는 어떤 구현체를 선택할까?
-> 직접 실습

```java
package study.querydsl.repository;

public interface TestInterface {
    void test();
    void print();
}

```

```java
package study.querydsl.repository;

import org.springframework.stereotype.Repository;

@Repository
public class TestRepositoryImpl1 implements TestInterface {

    @Override
    public void test() {
        System.out.println("test");
    }

    @Override
    public void print() {
        System.out.println("print");
    }
}

```

```java
package study.querydsl.repository;

import org.springframework.stereotype.Repository;

@Repository
public class TestInterfaceImpl2 implements TestInterface {

    @Override
    public void test() {
        System.out.println("구현체 2번에서 test실행");
    }

    @Override
    public void print() {
        System.out.println("구현체 2번에서 print실행");
    }
}

```



```
package study.querydsl.repository;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class TestClass {

    @Autowired
    TestInterface testInterface;

    @Test
    public void test() {
        testInterface.test();
        testInterface.print();
    }
}
```

```
ype 'study.querydsl.repository.TestInterface' available: expected single matching bean but found 2: testInterfaceImpl2,testRepositoryImpl1
```

결론 : 사용하고자 하는 구현체 1개만 빈등록을 시켜주자.