62일차 강의

* 개발자 테스트 ( TDD ) – Test Driven Development 방법론
* 개발 방법론 중의 하나
  + 애자일 방법론

: 테스트1 -> 구현1 -> 테스트2 -> 구현2 -> 완료 …

( 이전 방식의 흐름)

구현 -> 테스트 -> 완료 -> 재분석…

* JUnit

: Java에 독립된 단위테스트(Unit Test)를 지원해주는 프레임워크

소스 코드의 특정 모듈이 의도된 대로 정확히 작동하는지 검증하는 절차

(진짜 전체가 다 그렇게 돌아가는지 검증, 메모리 검증, 수행속도에 대한 검증)

즉, **모든** 함수와 메서드에 대한 테스트 케이스(Test Case)를 작성하는 절차를 말함

* JUnit을 쓸려면 pom.xml 에 추가를 해야한다

: 인터넷에서 maven Repository 사이트로 가서

JUnit 4.2버전을 클릭하여 복사한후 pom.xml의 <dependencies> 사이에 붙여넣기 해준다

* Junit 테스트를 이용해서 ( EduDao)

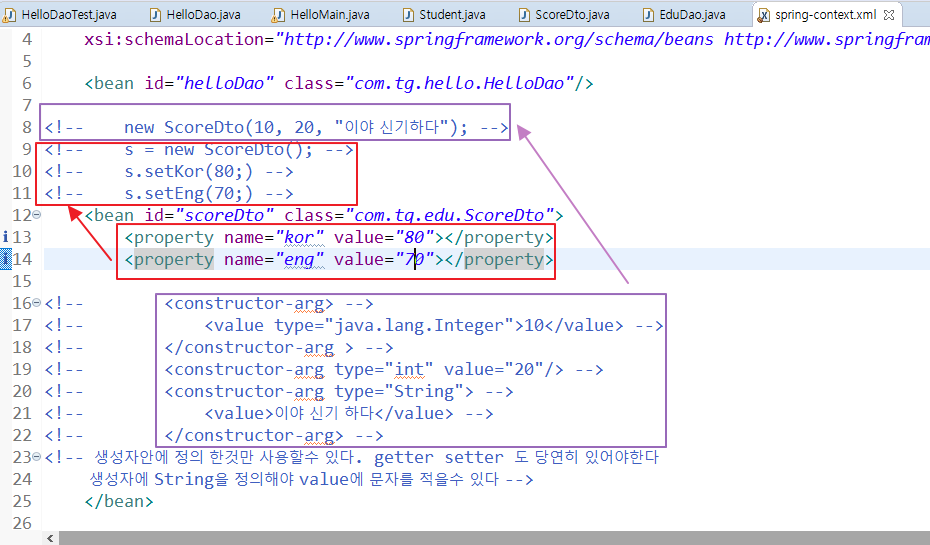
메서드를 수행하면, 성과 이름이 일치해야하고

성적이 들어가야한다

* eduDao에 무슨짓을 해서 점수에 대한 총합을 테스트할수 있는 것을 만들어라

scoreDto에서 객체를 받든

eduDaoTest에서 set 을 하든

* 강사님 방식 ( 깃허브에서 받을 것
* 
* TDD 수행 순서

1. 무조건 main 에서 구현할 기능을 만든다

최소한의 수행만 가능하도록(미구현)

2. test에서 동일한 형태의 구조로 테스트 클래스를 만든다

테스트를 수행할 수 있도록 내용을 구성한다

1. main에서 test의 결과가 잘 반영될 수 있도록 내용을 구현한다

**@ Test**

: 선언된 메서드는 테스트를 수행하는 메서드가 됨

JUnit은 각각의 테스트가 서로 영향을 주지 않고 독립적으로 실행됨을 원칙으로 하므로 @Test 마다 객체를 생성함

**@ Ignore**

: 선언된 메서드는 테스트를 실행하지 않게 함

**@ Before**

: 선언된 메서드는 @ Test 메서드가 실행되기 전에 반드시 실 행함.

@ Test 메서드에서 공통으로 사용하는 코드를

@ Before 메서드에 선언하여 사용한다.

**@ After**

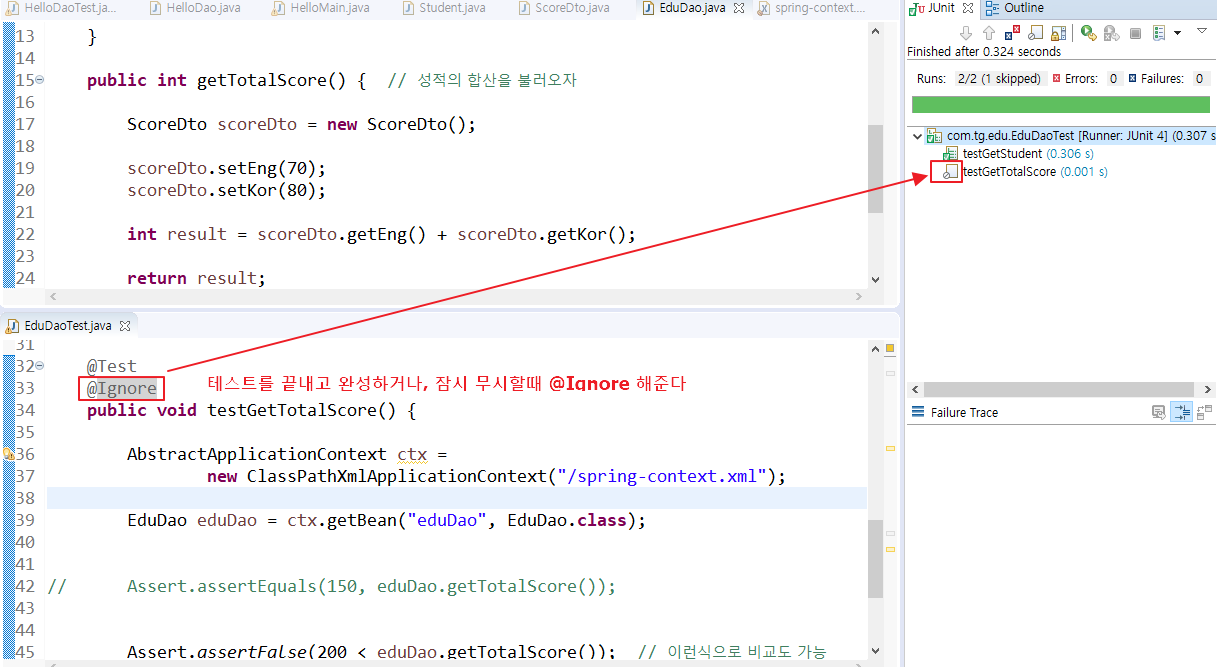
: 선언된 메서드는 @Test 메서드가 실행된 후 실행 됨.

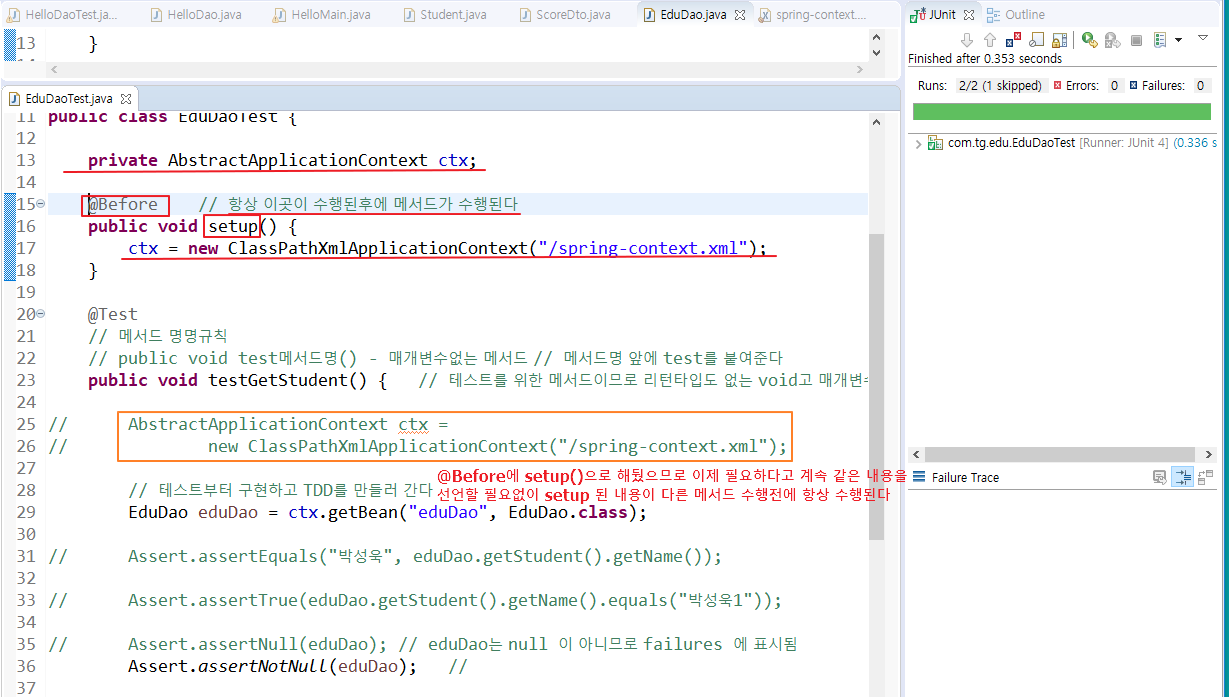
**@ BeforeCalss**

: @ Test 메서드 보다 먼저 한번만 수행되어야 할 경우에 사용한다.

**@ AfterClass**

: @ Test 메서드 보다 나중에 한번만 수행되어야 할 경우에 사용한다.





* 강사님이 만드신 SpringHome 프로젝트 가져오고, 설정확인하기
* Web.xml 은 웹에 대한 설정이므로

무조건 실행시키면 가장 먼저 실행된다!!!!!!

* 구성의 요약지라고 생각하면 된다
* Views 에서 사용자에게 보여지는 파일들을 관리한다
* Spring – 스프링에 대한 설정들
* Doc – document 폴더,
* Webapp 에서 화면에 관여된 것들을 관리한다

<http://localhost:8090/springHome/home.do>

* **MVC (Model – View – Controller )**

: 모델 – 뷰 – 컨트롤러 목적

Business Logic과 Presentation Logic을 분리하기 위함

* Business logic : 업무에 관련된 내용 (사용자에게 보여주지 않아야 될 부분을 분리하여 숨기려 함)
* Presentation logic : 비즈니스 로직에 대한 내용을 사용자에게 보여주는 영역
  + 좋은 점

: 사용자 인터페이스로부터 비즈니스 로직을 분리하여 애플리케에션의 시각적 요소나 그 이면에서 실행되는 비즈니스 로직을 서로 영향 없이 쉽게 고칠 수 있는 애플리케이션을 만들 수 있다.

* + Model : 애플리케이션의 정보(데이터, Business Logic포함)
  + View : 사용자에게 제공할 화면(Presentation Logic)
  + Controller : Model과 View 사이의 상호 작용을 관리

(무슨 모델이, 무슨 뷰가 필요한지 컨트롤러에서 관리한다)

* **MVC 컴포넌트의 역할**
  + **모델 컴포넌트**
    - 데이터 저장소(ex: 데이터베이스 등)와 연동하여 사용자가 입력하나 데이터나 사용자에게 출력할 데이터를 다루는 일을 함.
    - 여러 개의 데이터 변경 작업(추가, 변경, 삭제)을 하나의 작업으로 묶는 트랜잭션을 다루는 일도 함

( 추가,변경,삭제 – 가 하나의 업무로서 처리되도록 관리)

- 따로 따로가 아닌 하나의 업무로서 처리되도록 하고 commit 되어야 함. 그래야 rollback해도 한번에 다 처리되니깐)

* + - Dao(Data Access Object)클래스, Service 클래스에 해당
  + **뷰 컴포넌트**
    - 모델이 처리한 데이터나 그 작업 결과를 가지고 사용자에게 출력할 화면을 만드는 일을 함
    - 생성된 화면은 웹 브라우저가 출력하고, 뷰 컴포넌트는 HTML과 CSS, Javascript를 사용하여 웹 브라우저가 출력할 UI를 만듦
    - Html과 JSP(Java Server, Servlet Page)를 사용하여 작성 가능함
  + **컨트롤러 컴포넌트 (여기가 중요함!!! 우리가 코드짤곳!)**
    - 클라이언트의 요청을 받았을 때 그 요청에 대해 실제 업무를 수행하는 모델 컴포넌트를 호출하는 일을 함
    - 클라이언트가 보낸 데이터가 있다면, 모델을 호출할 때 전달하기 쉽게 데이터를 적절히 가공하는 일을 함
    - 모델이 업무 수행을 완료하면, 그 결과를 가지고 화면을 생성하도록 뷰에게 전달(클라이언트 요청에 대해 모델과 뷰를 결정하여 전달)
    - Servlet과 JSP를 사용하여 작성 할 수 있음
* Only servlet, 스파게티 패턴, mvc 패턴, 중앙집중형(mvc 모델2)
* 페이지 프론트 콘트롤러 방식

: 명령을 어떻게 전달할까?

* + 중앙집권형
  + 중앙분권형