1.- Select all the options that are describing TDD.

1. TDD is the approach keeping test one step ahead of your code (T)
2. Is also refereed like Test Driven Design. (T)
3. Write your initial implementation and write the initial test
4. Write your full functional code and write the Junit.
5. The idea of this approach is you are designing and developing your code in conjunction with your test. (T)

2.- What is the best option to define the next sentence?

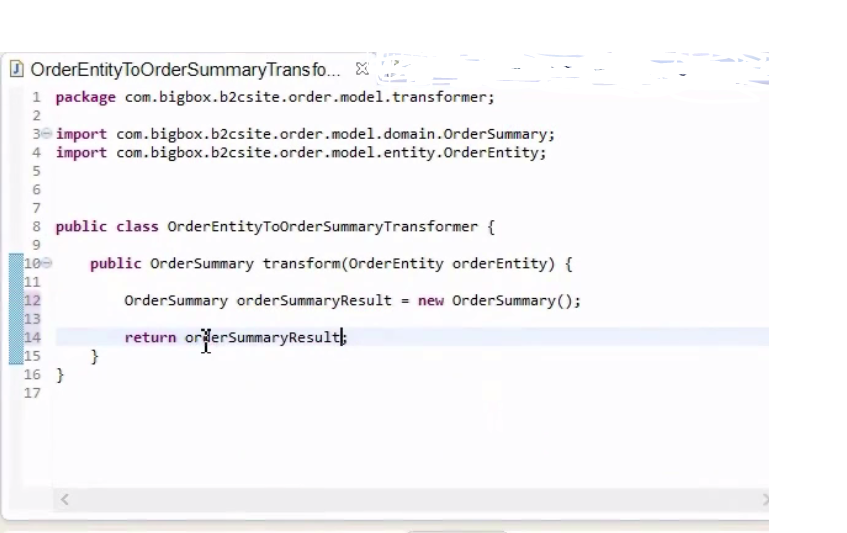
The premise of this approach is to write a statement or a set of statement resulting in a failing test. This would result in a red bar in the unit test runner’s display. Next you implement the functionality. This should result in a green bar displayed in the test runner. Once this is complete, you decide if refactoring is needed.

1. Red - Green-Refactor Pattern
2. Mockito /Easy Mock/PowerMock
3. MVC Pattern
4. Test Case Assertion Refactoring pattern

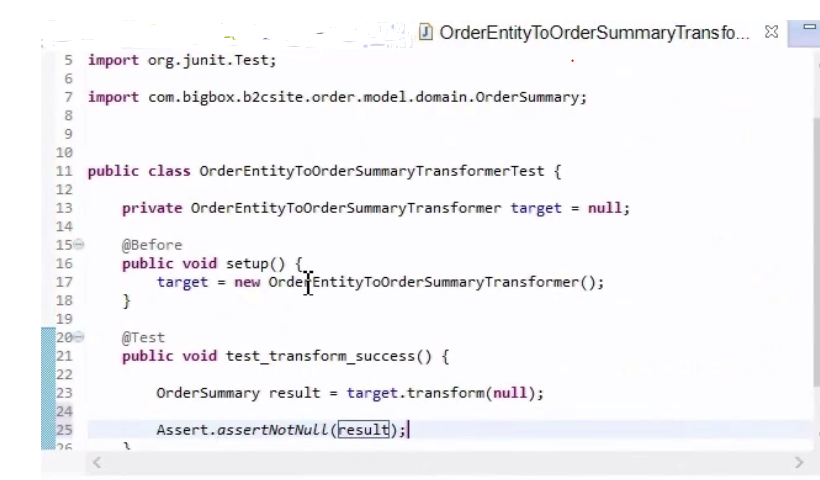
3.- Mention some Framewors in Java Supporting Testing TDD Answer: Mockito, PowerMockito, DBUnit

4.-explain what is not correct in the next process using TTD?

Step 1



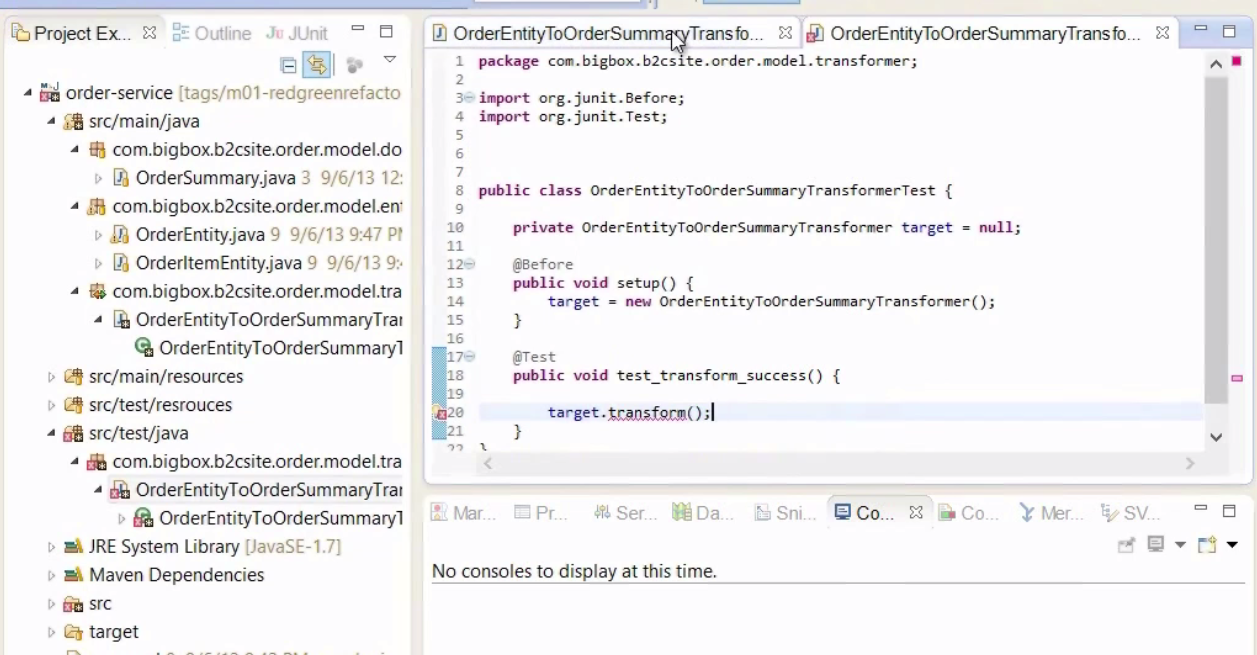
Step 2



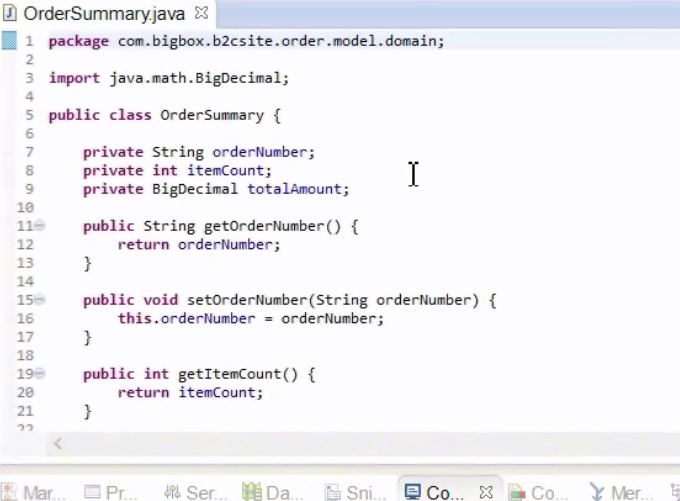
Possible Answer: The Unit test was written after the Implementation and is not following the Red/Green/Refactoring pattern

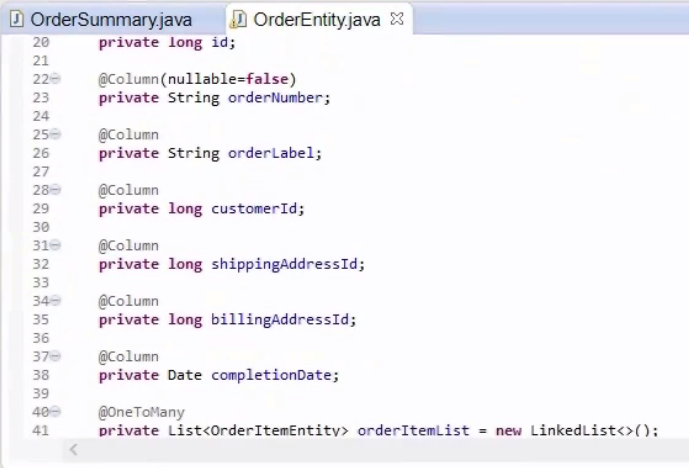
5.- Identify Red/Green/Refactor and explain the TDD in the next screenshots

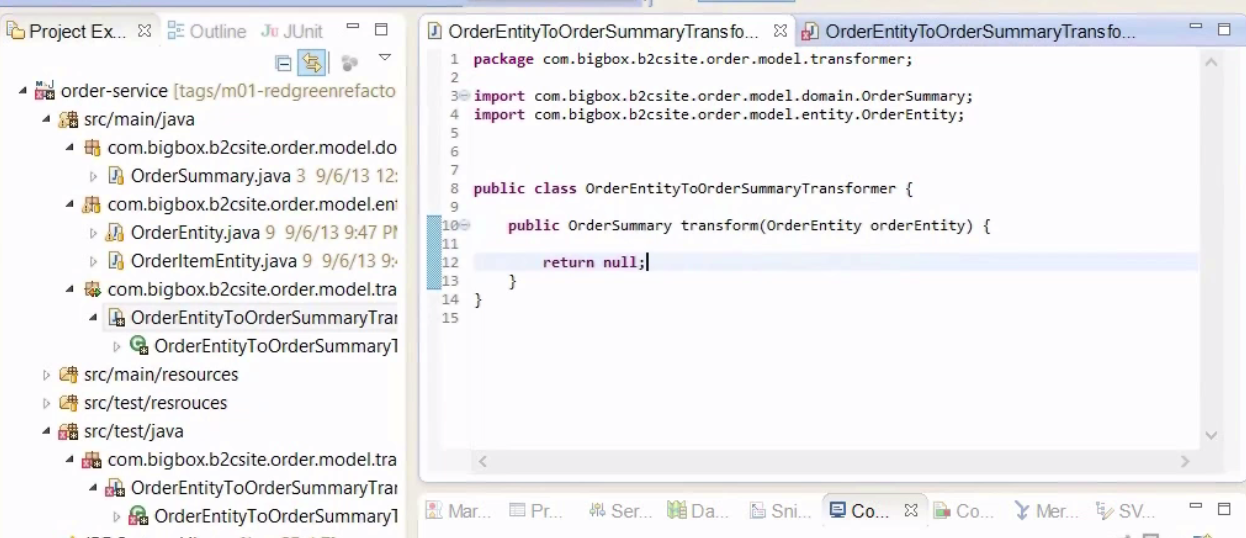
Following the Red/Green/Refactor. The Unit test is created and we got a compilation error that is considered Red (the first Red in our cycle), this is correct using TDD.

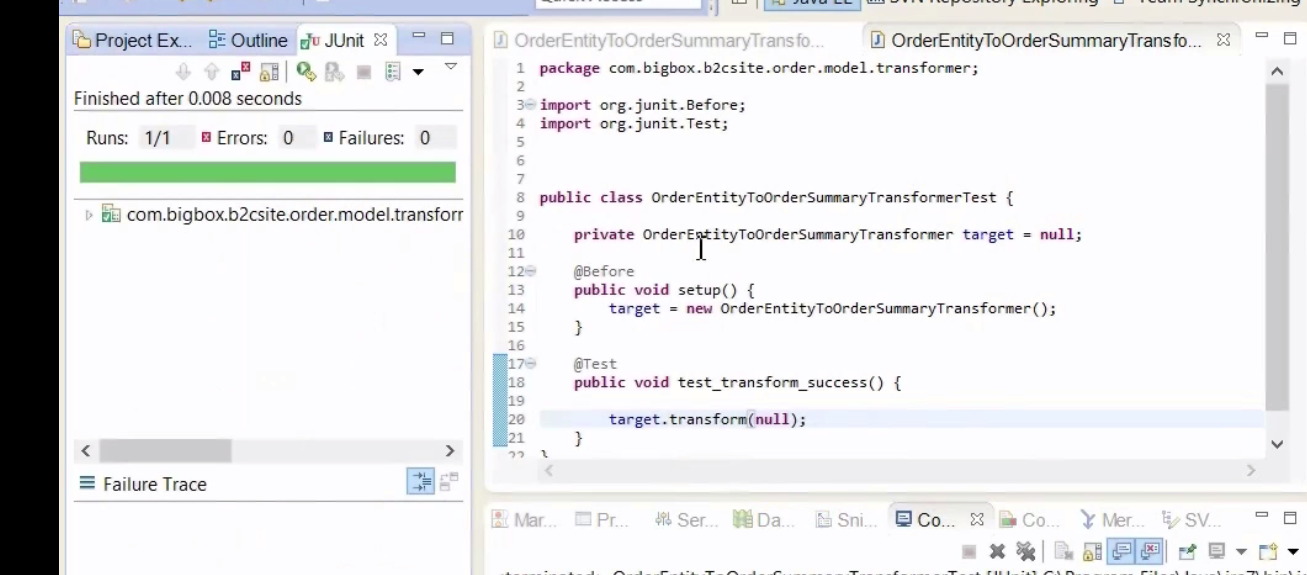


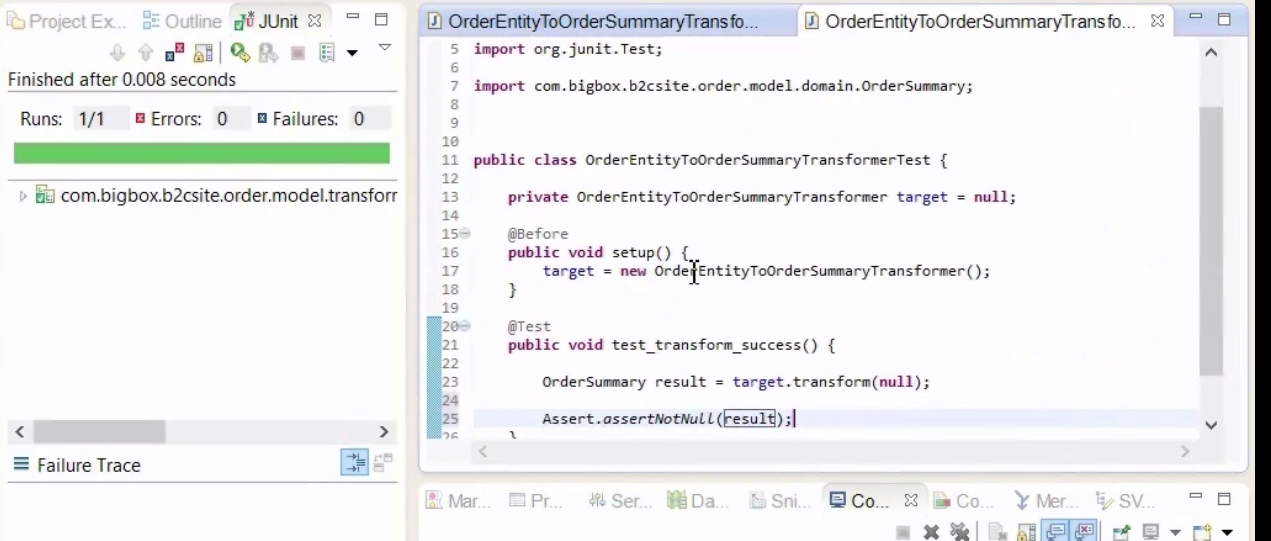
Write the Functional Implementation our real code







Run the junit to get a Green test and Refactor the code if is necessary code 



6.- from the previous screenshots, finish the Junit and implementation code following the TTD methodology (enumerate and explain the steps to follow)

Generate a Random orderNumber in your test\_transform\_success method and validate that transform method is returning the orderNumber generated.

The target in our junit:

{

String randomOrder=generateOrderNumber();

Assert.assertEquals(randomOrder ,result.getOrderNumber());

}

Answer

Step 1, refactor the Junit and run it to get a Red Fail

@Test

public void test\_transform\_success(){

String orderNumber= UUID.randomUUID.toString();

OrderEntity entity=new OrderEntity();

entity.setOrderNumber(orderNumber);

OrderSummary result=target.transform(entity);

Assert.assertEquals randomOrder ,result.getOrderNumber());

}

Step 2

Refactor the transform method in OrderEntityToOrderSummaryTransformer

public OrderSummary transform(OrderEntity orderEntity){

OrderSummary orderSummaryResult=new OrderSummary();

orderSummaryResult.setOrderNumber(entity.getOrderNumber());

return orderSummaryResult;

}

Step 3

Run the test to get a Green result, the Assert should be success

7.- Which of the following annotations marks a method to setup test data before each test method execution in the testing class?

A)@After

B)@BeforeClass

C)@Before

D)@AfterClass

8.- Test-Driven Development is sometimes referred to as Test-Driven Design due to its focus on designing a system in a modular manner that is testable?

True

False

9.- Which of the following is considered an anti-pattern of TDD?

A) Developing a complete user store or use case before writing any unit testing code

B) Following the red/green/refactor approach to developing code

C) Setting up a fresh test fixture for each automated test method

D) All the options.

10) Mention some commons Myths about TDD

Coding effort increases significantly

All tests are written before any code