**JUnit 5 Changes**



**Pom.xml changes**

Spring Boot projects with versions >= 2.2.0 use JUnit 5 by default. You should see something like this in your pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

<dependency>

**Example: Writing Unit Test for a business service**

**package** com.mrcoder.customer\_creation\_service.services;  
  
**public class** CustomerCreationService {  
  
 **private** CustomerDao **customerDao**;  
  
 **public int** calculateSum(**int**[] data){  
 **int** sum=0;  
 **for**(**int** value : data){  
 sum+=value;  
 }  
 **return** sum;  
 }

}

**package** com.mrcoder.customer\_creation\_service.service;  
  
**import** com.mrcoder.customer\_creation\_service.services.CustomerCreationService;  
**import** org.junit.jupiter.api.Assertions;  
**import** org.junit.jupiter.api.Test;  
  
**public class** CustomerCreationTest {  
  
 @Test  
 **public void** calculateSum(){  
 CustomerCreationService service = **new** CustomerCreationService();  
 **int** acutalValue=service.calculateSum(**new int**[]{1,2,3});  
 Assertions.*assertEquals*(6,acutalValue);  
 }  
  
 @Test  
 **public void** calculateSum\_Empty(){  
 CustomerCreationService service = **new** CustomerCreationService();  
 **int** acutalValue=service.calculateSum(**new int**[]{});  
 Assertions.*assertEquals*(0,acutalValue);  
 }  
  
 @Test  
 **public void** calculateSum\_OneValue(){  
 CustomerCreationService service = **new** CustomerCreationService();  
 **int** acutalValue=service.calculateSum(**new int**[]{5});  
 Assertions.*assertEquals*(5,acutalValue);  
 }  
}

**Example : Writing first unit test with stub**

When class’s method is having other class’s method then how we can get data. Lets see using multiple approaches and its advantage and disadvantage.

1. Using implementation class
2. Using mock()
3. Using @InjectMocks and @Mock

Note:

If you are using JUnit 5, in the next lecture use

@ExtendWith(MockitoExtension.class)

instead of @RunWith(MockitoJUnitRunner.class)

**public interface** CustomerDao {  
 **int** findCustomerDetails();  
}

**public class** CustomerCreationService {  
  
 **private** CustomerDao **customerDao**;  
  
 **public void** setCustomerDao(CustomerDao customerDao) {  
 **this**.**customerDao** = customerDao;  
 }  
  
 **public int** calculateSumUsingStub(){  
 **return customerDao**.findCustomerDetails();  
 }

}

**Write Test Cases Using Stub : Using Implementation Class**

* Advantage
* We are able to get expected data when dependent method is called without passing parameters
* Disadvantage
* We have interface so whenever we add more methods then it harms implemeted class

**class** CustomerCreationStubDao **implements** CustomerDao {  
 @Override  
 **public int**[] findCustomerDetails() {  
 **return new int**[]{1,2,3};  
 }  
}  
  
**class** CustomerCreationEmptyStubDao **implements** CustomerDao {  
 @Override  
 **public int**[] findCustomerDetails() {  
 **return new int**[]{};  
 }  
}  
  
**class** CustomerCreationOneValueStubDao **implements** CustomerDao {  
 @Override  
 **public int**[] findCustomerDetails() {  
 **return new int**[]{5};  
 }  
}  
  
  
**public class** CustomerCreationStubTest {  
  
 @Test  
 **public void** calculateSumUsingStub(){  
 CustomerCreationService service = **new** CustomerCreationService();  
 service.setCustomerDao(**new** CustomerCreationStubDao());  
 **int** acutalValue=service.calculateSumUsingStub();  
 Assertions.*assertEquals*(6,acutalValue);  
 }  
  
 @Test  
 **public void** calculateSumUsingStub\_Empty(){  
 CustomerCreationService service = **new** CustomerCreationService();  
 service.setCustomerDao(**new** CustomerCreationEmptyStubDao());  
 **int** acutalValue=service.calculateSumUsingStub();  
 Assertions.*assertEquals*(0,acutalValue);  
 }  
  
 @Test  
 **public void** calculateSumUsingStub\_OneValue(){  
 CustomerCreationService service = **new** CustomerCreationService();  
 service.setCustomerDao(**new** CustomerCreationOneValueStubDao());  
 **int** acutalValue=service.calculateSumUsingStub();  
 Assertions.*assertEquals*(5,acutalValue);  
 }  
}

**Write Test Cases Using Stub : Using Mock**

**package** com.mrcoder.customer\_creation\_service.service;  
  
**import** com.mrcoder.customer\_creation\_service.dao.CustomerDao;  
**import** com.mrcoder.customer\_creation\_service.services.CustomerCreationService;  
**import** org.junit.jupiter.api.Assertions;  
**import** org.junit.jupiter.api.BeforeEach;  
**import** org.junit.jupiter.api.Test;  
**import static** org.mockito.Mockito.*mock*;  
**import static** org.mockito.Mockito.*when*;  
  
**public class** CustomerCreationMockTest {  
  
 **private** CustomerCreationService **service** = **new** CustomerCreationService();  
 **private** CustomerDao **customerDao** = *mock*(CustomerDao.**class**);  
  
 @BeforeEach  
 **public void** beforeEach(){  
 **service**.setCustomerDao(**customerDao**);  
 }  
  
 @Test  
 **public void** calculateSumUsingMock(){  
 *when*(**customerDao**.findCustomerDetails()).thenReturn(**new int**[]{1,2,3});  
 Assertions.*assertEquals*(6,**service**.calculateSumUsingStub());  
 }  
  
 @Test  
 **public void** calculateSumUsingMock\_Empty(){  
 *when*(**customerDao**.findCustomerDetails()).thenReturn(**new int**[]{0});  
 Assertions.*assertEquals*(0,**service**.calculateSumUsingStub());  
 }  
  
 @Test  
 **public void** calculateSumUsingMock\_OneValue(){  
 *when*(**customerDao**.findCustomerDetails()).thenReturn(**new int**[]{5});  
 Assertions.*assertEquals*(5,**service**.calculateSumUsingStub());  
 }  
}

**Write Test Cases Using Stub : Using @InjectMocks and @Mock**

**package** com.mrcoder.customer\_creation\_service.service;  
  
**import** com.mrcoder.customer\_creation\_service.dao.CustomerDao;  
**import** com.mrcoder.customer\_creation\_service.services.CustomerCreationService;  
**import** org.junit.jupiter.api.Assertions;  
**import** org.junit.jupiter.api.BeforeEach;  
**import** org.junit.jupiter.api.Test;  
**import** org.junit.jupiter.api.extension.ExtendWith;  
**import** org.mockito.InjectMocks;  
**import** org.mockito.Mock;  
**import** org.mockito.junit.jupiter.MockitoExtension;  
  
**import static** org.mockito.Mockito.*mock*;  
**import static** org.mockito.Mockito.*when*;  
  
**@ExtendWith(MockitoExtension.class)  
public class** CustomerCreationMockTest {  
  
 **@InjectMocks  
 private CustomerCreationService service;  
  
 @Mock  
 private CustomerDao customerDao;**  
  
 @Test  
 **public void** calculateSumUsingMock(){  
 *when*(**customerDao**.findCustomerDetails()).thenReturn(**new int**[]{1,2,3});  
 Assertions.*assertEquals*(6,**service**.calculateSumUsingStub());  
 }  
  
 @Test  
 **public void** calculateSumUsingMock\_Empty(){  
 *when*(**customerDao**.findCustomerDetails()).thenReturn(**new int**[]{0});  
 Assertions.*assertEquals*(0,**service**.calculateSumUsingStub());  
 }  
  
 @Test  
 **public void** calculateSumUsingMock\_OneValue(){  
 *when*(**customerDao**.findCustomerDetails()).thenReturn(**new int**[]{5});  
 Assertions.*assertEquals*(5,**service**.calculateSumUsingStub());  
 }  
}

**Mockito Tips 1 : Multiple Return Values and Specific Arguments**

**package** com.mrcoder.customer\_creation\_service.service;  
  
*//import org.mockito.Mock;***import** org.junit.jupiter.api.Assertions;  
**import** org.junit.jupiter.api.Test;  
  
**import static** org.mockito.Mockito.\*;  
  
**import** java.util.List;  
  
**public class** ListMockTest {  
  
 List **listMock** = *mock*(List.**class**);  
  
 @Test  
 **public void** size\_basic(){  
 *when*(**listMock**.size()).thenReturn(5);  
 Assertions.*assertEquals*(5,**listMock**.size());  
 }  
  
 @Test  
 **public void** returnDifferentValue(){  
 *when*(**listMock**.size()).**thenReturn(5).thenReturn(10);** Assertions.*assertEquals*(5,**listMock**.size());  
 Assertions.*assertEquals*(10,**listMock**.size());  
 }  
  
 @Test  
 **public void** returnWithParameters(){  
 ***when*(listMock.get(0)).thenReturn("Arun");**  
 Assertions.*assertEquals*(**"Arun"**,**listMock**.get(0));  
 Assertions.*assertEquals*(**null**,**listMock**.get(1));  
 }  
}

**Mockito Tips 2 : Generic Paramaters and Argument Matcher**

**package** com.mrcoder.customer\_creation\_service.service;  
  
*//import org.mockito.Mock;***import** org.junit.jupiter.api.Assertions;  
**import** org.junit.jupiter.api.Test;  
  
**import static** org.mockito.Mockito.\*;  
  
**import** java.util.List;  
  
**public class** ListMockTest {  
  
 List **listMock** = *mock*(List.**class**);  
  
 @Test  
 **public void** returnWithGenericParameters(){  
 *when*(**listMock**.get(***anyInt*()**)).thenReturn(**"Arun"**);  
 Assertions.*assertEquals*(**"Arun"**,**listMock**.get(0));  
 Assertions.*assertEquals*(**"Arun"**,**listMock**.get(1));  
 }

}

**Mockito Tips 2 : Verify method calls**

**Mockito Tips 3 : Argument Matcher**

**Mockito Tips 4 : Argument Matcher on Multiple Calls**

**Introduction to Spy**

* You can mock classes the same way you mock interfaces.
* The mock doesn’t retain behaviour (code) of original classes.
* The spy, by default retain behaviour (code) of original classes.

**import** org.junit.jupiter.api.Assertions;  
**import** org.junit.jupiter.api.Test;  
  
**import static** org.mockito.Mockito.\*;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** ListMockTest {

@Test  
**public void** mocking(){  
 ArrayList arrayListMock = *mock*(ArrayList.**class**);  
 System.***out***.println(arrayListMock.get(0)); *// null* System.***out***.println(arrayListMock.size()); *// 0* arrayListMock.add(**"Test1"**);  
 arrayListMock.add(**"Test2"**);  
 System.***out***.println(arrayListMock.size()); *// 0  
 when*(arrayListMock.size()).thenReturn(5);  
 System.***out***.println(arrayListMock.size()); *// 5*}  
  
@Test  
**public void** spying(){  
 ArrayList arrayListSpy = *spy*(ArrayList.**class**);  
 arrayListSpy.add(**"Test0"**);  
 System.***out***.println(arrayListSpy.get(0)); *// Test0* System.***out***.println(arrayListSpy.size()); *// 1* arrayListSpy.add(**"Test1"**);  
 arrayListSpy.add(**"Test2"**);  
 System.***out***.println(arrayListSpy.size()); *// 3  
 when*(arrayListSpy.size()).thenReturn(5);  
 System.***out***.println(arrayListSpy.size()); *// 5* arrayListSpy.add(**"Test4"**);  
 System.***out***.println(arrayListSpy.size()); *// 5*}

}

**Mockito FAQ**

* <https://github.com/dart-lang/mockito/blob/master/FAQ.md>
* <https://github.com/mockito/mockito/wiki/FAQ>

**Section-2 : Unit Testing with Spring Boot and Mockito**

Note:

(If you are using JUnit 5) In the next lecture, You do NOT need to add

@RunWith(SpringRunner.class) on top of @WebMvcTest(HelloWorldController.class)

JUnit 4 Code

* @RunWith(SpringRunner.class)
* @WebMvcTest(HelloWorldController.class)

JUnit 5 Code

* @WebMvcTest(HelloWorldController.class)

**Lesson – 01 Using Mock Mvc to test Hello World Controller**

HelloWorldController.class

**import** org.springframework.web.bind.annotation.GetMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
@RestController  
**public class** HelloWorldController {  
  
 @GetMapping(**"/hello-world"**)  
 **public** String helloWorld(){  
 **return "Hello World"**;  
 }  
}

HelloWorldControllerTest.class

**import** com.mrcoder.customer\_creation\_service.controllers.HelloWorldController;  
**import** org.junit.jupiter.api.Assertions;  
**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;  
**import** org.springframework.test.web.servlet.MockMvc;  
**import** org.springframework.test.web.servlet.MvcResult;  
**import** org.springframework.test.web.servlet.RequestBuilder;  
**import** org.springframework.test.web.servlet.request.MockMvcRequestBuilders;  
  
@WebMvcTest(value = HelloWorldController.**class**)  
**public class** HelloWorldControllerTest {  
  
 @Autowired  
 **private** MockMvc **mockMvc**;  
  
 @Test  
 **public void** helloWorld\_basis() **throws** Exception {  
 *// call GET 'hello-world' application/json* RequestBuilder requestBuilder = MockMvcRequestBuilders  
 .*get*(**"/hello-world"**)  
 .accept(**"application/json"**);  
 MvcResult mvcResult = **mockMvc**.perform(requestBuilder).andReturn();  
 *// verify Hello World* Assertions.*assertEquals*(**"Hello World"**,mvcResult.getResponse().getContentAsString());  
 }  
}

**Lesson – 02 Using response matchers to check status and content**

@Test  
**public void** helloWorld\_response\_matchers() **throws** Exception {  
 *// call GET 'hello-world' application/json* RequestBuilder requestBuilder = MockMvcRequestBuilders  
 .*get*(**"/hello-world"**)  
 .accept(**"application/json"**);  
 MvcResult mvcResult = **mockMvc**.perform(requestBuilder)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().string(**"Hello World"**))  
 .andReturn();  
 *// verify Hello World  
 //Assertions.assertEquals("Hello World",mvcResult.getResponse().getContentAsString());*}

**Lesson – 03 Using Test Item Controller and Basic Json Assert**

**import** com.mrcoder.customer\_creation\_service.controllers.ItemController;  
**import** org.junit.jupiter.api.Assertions;  
**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;  
**import** org.springframework.test.web.servlet.MockMvc;  
**import** org.springframework.test.web.servlet.MvcResult;  
**import** org.springframework.test.web.servlet.RequestBuilder;  
**import** org.springframework.test.web.servlet.request.MockMvcRequestBuilders;  
  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.*content*;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*;  
  
@WebMvcTest(value = ItemController.**class**)  
**public class** ItemControllerTest {  
  
 @Autowired  
 **private** MockMvc **mockMvc**;  
  
 @Test  
 **public void** dummyItem\_basis() **throws** Exception {  
 *// call GET 'hello-world' application/json* RequestBuilder requestBuilder = MockMvcRequestBuilders  
 .*get*(**"/dummy-item"**)  
 .accept(**"application/json"**);  
 MvcResult mvcResult = **mockMvc**.perform(requestBuilder)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().json(**"{\"id\":1,\"name\":\"Tea\"}"**))  
 .andReturn();  
 }  
}

**Lesson – 04 Digging deeper into JSON Assert**

**package** com.mrcoder.junit.mockito.spike;  
  
**import** org.json.JSONException;  
**import** org.junit.jupiter.api.Test;  
**import** org.skyscreamer.jsonassert.JSONAssert;  
  
**public class** JsonAssertTest {  
  
 String **actualResponse** =**"{\"id\":1,\"name\":\"Tea\"}"**;  
  
 @Test  
 **public void** jsonAssert\_strictTrue\_ExactMatchExceptSpaces() **throws** JSONException {  
 String expectedResponse =**"{\"id\":1,\"name\":\"Tea\"}"**;  
 JSONAssert.*assertEquals*(expectedResponse,**actualResponse**,**true**);  
 *// test pass if exact json in expect and actual* }  
  
 @Test  
 **public void** jsonAssert\_strictFalse\_ExactMatchExceptSpaces() **throws** JSONException {  
 String expectedResponse =**"{\"id\":1}"**;  
 JSONAssert.*assertEquals*(expectedResponse,**actualResponse**,**false**);  
 *// test pass if json format is correct* }  
  
 @Test  
 **public void** jsonAssert\_WithoutEscapeCharacter() **throws** JSONException {  
 String expectedResponse =**"{id:1,name:Tea}"**;  
 JSONAssert.*assertEquals*(expectedResponse,**actualResponse**,**false**);  
 }  
}

**Lesson – 05 Writing unit test for REST service mocking business layer**

Controller

@Autowired  
**private** ItemBusinessService **itemBusinessService**;

@GetMapping(**"/item-from-business-service"**)  
**public** Item itemFromBusinessService(){  
 **return itemBusinessService**.readHardCodedItem();  
  
}

Service

**public class** ItemBusinessService {  
 **public** Item readHardCodedItem(){  
 **return new** Item(1,**"Tea"**);  
 }  
}

Unit Test

**package** com.mrcoder.junit.mockito.controller;  
  
**import** com.mrcoder.junit.mockito.controllers.ItemController;  
**import** com.mrcoder.junit.mockito.models.Item;  
**import** com.mrcoder.junit.mockito.services.ItemBusinessService;  
**import** org.junit.jupiter.api.Test;  
**import** org.mockito.Mockito;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;  
**import** org.springframework.boot.test.mock.mockito.MockBean;  
**import** org.springframework.test.web.servlet.MockMvc;  
**import** org.springframework.test.web.servlet.MvcResult;  
**import** org.springframework.test.web.servlet.RequestBuilder;  
**import** org.springframework.test.web.servlet.request.MockMvcRequestBuilders;  
  
**import static** org.mockito.Mockito.*when*;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.*content*;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*;  
  
@WebMvcTest(value = ItemController.**class**)  
**public class** ItemControllerTest {  
  
 @Autowired  
 **private** MockMvc **mockMvc**;  
  
 @MockBean  
 **private** ItemBusinessService **itemBusinessService**;  
  
 @Test  
 **public void** itemFromBusinessService\_basis() **throws** Exception {  
 *when*(**itemBusinessService**.readHardCodedItem()).thenReturn(**new** Item(1,**"Car"**));  
 RequestBuilder requestBuilder = MockMvcRequestBuilders  
 .*get*(**"/item-from-business-service"**)  
 .accept(**"application/json"**);  
 MvcResult mvcResult = **mockMvc**.perform(requestBuilder)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().json(**"{id:1,name:Car}"**))  
 .andReturn();  
 }

}

**Lesson – 06 Introducing H2 Memory Database**

Use this configuration in  application.properties

1. spring.datasource.url=jdbc:h2:mem:testdb;NON\_KEYWORDS=USER
2. spring.h2.console.enabled=true
3. spring.data.jpa.repositories.bootstrap-mode=default
4. spring.jpa.defer-datasource-initialization=true

Dependencies

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.h2database</**groupId**>  
 <**artifactId**>h2</**artifactId**>  
</**dependency**>

**Lesson – 07 Implementation with Controller, Service and Repository**

**Controller**

@Autowired  
**private** ItemBusinessService **itemBusinessService**;

@GetMapping(**"/all-item-from-database"**)  
**public** List<Item> allItemFromDatabase(){  
 **return itemBusinessService**.retrieveAllItems();  
}

**Service**

@Autowired  
**private** ItemDao **itemDao**;

**public** List<Item> retrieveAllItems(){  
 List<Item> itemList = **itemDao**.findAll();  
 **for**(Item item:itemList){  
 item.setValue(item.getPrice()\* item.getPrice());  
 }  
 **return** itemList;  
}

Repository

**public interface** ItemDao **extends** JpaRepository<Item,Integer> {  
}

**Controller Test Case**

@WebMvcTest(value = ItemController.**class**)  
**public class** ItemControllerTest {

@Autowired  
**private** MockMvc **mockMvc**;  
  
@MockBean  
**private** ItemBusinessService **itemBusinessService**;

@Test  
**public void** retrieveAllItems\_basic() **throws** Exception {  
 *when*(**itemBusinessService**.retrieveAllItems()).thenReturn(  
 Arrays.*asList*(**new** Item(1,**"Item1"**,10,20),  
 **new** Item(2,**"Item2"**,11,21)  
 ));  
 RequestBuilder requestBuilder = MockMvcRequestBuilders  
 .*get*(**"/all-item-from-database"**)  
 .accept(**"application/json"**);  
 MvcResult mvcResult = **mockMvc**.perform(requestBuilder)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().json(**"[{id:1,name:Item1,price:10},{id:2,name:Item2,price:11}]"**))  
 .andReturn();  
}

}

**Service Test Case**

@ExtendWith(MockitoExtension.**class**)  
**public class** ItemBusinessServiceTest {  
  
 @InjectMocks  
 **private** ItemBusinessService **service**;  
  
 @Mock  
 **private** ItemDao **itemDao**;  
  
 @Test  
 **public void** retrieveAllItems\_Basic(){  
  
 *when*(**itemDao**.findAll()).thenReturn(  
 Arrays.*asList*(**new** Item(1,**"Item1"**,10,20),  
 **new** Item(2,**"Item2"**,11,21)  
 )  
 );  
 List<Item> itemList = **service**.retrieveAllItems();  
 Assertions.*assertEquals*(100,itemList.get(0).getValue());  
 Assertions.*assertEquals*(121,itemList.get(1).getValue());  
 }  
}

**Repository Test Case**

@ExtendWith(MockitoExtension.**class**)  
**@DataJpaTest  
public class** ItemDaoTest {  
  
 @Autowired  
 **private** ItemDao **itemDao**;  
  
 @Test  
 **public void** testFindAll(){  
 List<Item> items = **itemDao**.findAll();  
 Assertions.*assertEquals*(4,items.size());  
 }  
}

**JUnit 5 Alert**

(If you are using JUnit 5) In the next lecture, You do NOT need to add

@RunWith(SpringRunner.class) on top of

@SpringBootTest(webEnvironment=WebEnvironment.RANDOM\_PORT)

JUnit 4 Code

@RunWith(SpringRunner.class)

@SpringBootTest(webEnvironment=WebEnvironment.RANDOM\_PORT)

JUnit 5 Code

@SpringBootTest(webEnvironment=WebEnvironment.RANDOM\_PORT)

**Lesson – 08 Writing an integration test using @SpringBootTest**

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.***RANDOM\_PORT***)  
**public class** ItemControllerTestIT {  
  
 @Autowired  
 **private** TestRestTemplate **restTemplate**;  
  
 @Test  
 **public void** contextTest() **throws** JSONException {  
 ResponseEntity<String> entity = **restTemplate**.getForEntity(**"/all-item-from-database"**, String.**class**);  
 String body = entity.getBody();  
 JSONAssert.*assertEquals*(**"[{id:1},{id:2},{id:3},{id:4}]"**,body,**false**);  
 }  
}

**Lesson – 09 Using @MockBean to mock out dependencies you want to talk**

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.***RANDOM\_PORT***)  
**public class** ItemControllerTestIT {  
  
 @Autowired  
 **private** TestRestTemplate **restTemplate**;  
  
 **@MockBean**  
 **private** ItemBusinessService **businessService**;  
  
 @Test  
 **public void** contextTest() **throws** JSONException {  
 *when*(**businessService**.retrieveAllItems()).thenReturn(  
 Arrays.*asList*(**new** Item(1,**"Item1"**,10,20),  
 **new** Item(2,**"Item2"**,11,21)  
 ));  
 ResponseEntity<String> entity = **restTemplate**.getForEntity(**"/all-item-from-database"**, String.**class**);  
 String body = entity.getBody();  
 JSONAssert.*assertEquals*(**"[{id:1},{id:2}]"**,body,**false**);  
 }  
}