CS544

LESSON 14
TESTING

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
November 28 **Lesson 1** Introduction Spring framework Dependency injection	November 29 **Lesson 2** Spring Boot AOP	November 30 **Lesson 3** JDBC JPA	December 1 **Lesson 4** JPA mapping 1	December 2 **Lesson 5** JPA mapping 2	December 3 **Lesson 6** JPA queries	December 4
December 5 **Lesson 7** Transactions	December 6 **Lesson 8** MongoDB	December 7 **Midterm Review**	December 8 **Midterm exam**	December 9 **Lesson 9** REST webservices	December 10 **Lesson 10** SOAP webservices	December 11
December 12 **Lesson 11** Messaging	December 13 **Lesson 12** Scheduling Events Configuration	December 14 **Lesson 13** Monitoring	December 15 **Lesson 14** Testing your application	December 16 **Final review**	December 17 **Final exam**	December 18
December 19 **Project**	December 20 **Project**	December 21 **Project**	December 22 **Presentations**			

UNIT TESTING BEST PRACTICES

Good unit tests: FIRST

- Fast
- Isolated
- Repeatable
- Self-validating
- Timely

Fast

- It should be comfortable to run all unit tests often
- Isolate slow tests from fast tests
    - Separate unit and integration tests

Isolated

- Only two possible results: PASS or FAIL
- No partially successful tests.
  - If a test can break for more than one reason, consider splitting it into separate tests
- Isolation of tests:
  - Different execution order must yield same results.
  - Test B should not depend on outcome of Test A

Repeatable

- A test should produce the same results each time you run it.
- Watch out for
  - Dates, times
  - Random numbers
  - Data from a datastore
- Use mock objects to give consistent data

Self-validating

- Your tests should be able to run anywhere at any time
- They should not depend on
  - Manual interaction
  - External setup

Timely

- Do not defer writing unit tests
  - For every method you write, write the corresponding unit tests at the same time
- Use test rules in your project
  - Review process
  - Test coverage tools

Unit test best practices

- Write tests for every found bug
- Fix failing tests immediately
- Make unit tests simple to run
  - Test suites can be run by a single command or a one button click.
- An incomplete set of unit tests is better than no unit tests at all.
- Don’t repeat production logic
- Reuse test code (setup, manipulate, assert)
- Don’t run a test from another test

Single Responsibility

- One test should be responsible for one scenario only.

- Test behavior, not methods:
  - One method, multiple behaviors → Multiple tests
  - One behavior, multiple methods → One test

Single Responsibility

```java
@Test
public void testMethod() {
    assertTrue(behaviour1);
    assertTrue(behaviour2);
    assertTrue(behaviour3);
}
```

```java
@Test
public void testMethodCheckBehaviour1() {
    assertTrue(behaviour1);
}

@Test
public void testMethodCheckBehaviour2() {
 assertTrue(behaviour2);
}

@Test
public void testMethodCheckBehaviour3() {
  assertTrue(behaviour3);
}
```

Self Descriptive

- Unit test must be easy to read and understand
  - Variable Names
  - Method Names
  - Class Names

    Self descriptive
  - No conditional logic
  - No loops
- Name tests to represent PASS conditions:
  - canMakeReservation()
  - totalBillEqualsSumOfMenuItemPrices()

No conditional logic

- Test should have no uncertainty:
  - All inputs should be known
  - Method behavior should be predictable
  - Expected output should be strictly defined
  - Split in to two tests rather than using “If” or “Case”
- Tests should not contain conditional logic.
  - If test logic has to be repeated, it probably means the test is too complicated.

No conditional logic

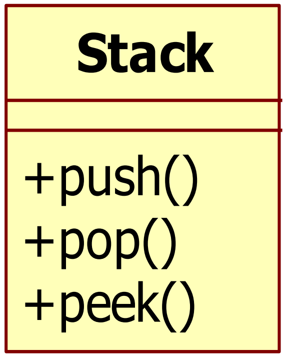
```
@Test
public void testMethod() {
  if (before)
    assertTrue(behaviour1);
  else if (after)
    assertTrue(behaviour2);
  else
    assertTrue(behaviour3);
}
```

```
@Test
public void testBefore() {
  boolean before = true;
  assertTrue(behaviour1);
}

@Test
public void testAfter() {
  boolean after= true;
  assertTrue(behaviour2);
}

@Test
public void testNow() {
   boolean before = false;
   boolean after= false;
   assertTrue(behaviour3);
}
```

Test only the public interface

- Every method has a side effect
  - Test this side effect
  - Test behavior, not methods
- What if this side effect is not visible (private attributes and methods)?
  - Do not sacrifice good design just for testing
  - Test behavior, not state

Test behavior, not methods/state



There is no use to test these methods in isolation

- Unit tests:
  - Pop of an empty stack should return null
  - Peek of an empty stack should return null
  - Push first x on the stack, then a peek should return x
  - Push first x on the stack, then a pop should remove x from the stack
  - Push first x, then y. A pop should return y and another pop should return x.

Summary

- Fast
- Isolated
- Repeatable
- Self-validating
- Timely
- Single responsibility
- No conditional logic
- Test behavior, not methods
  - Test the public interface

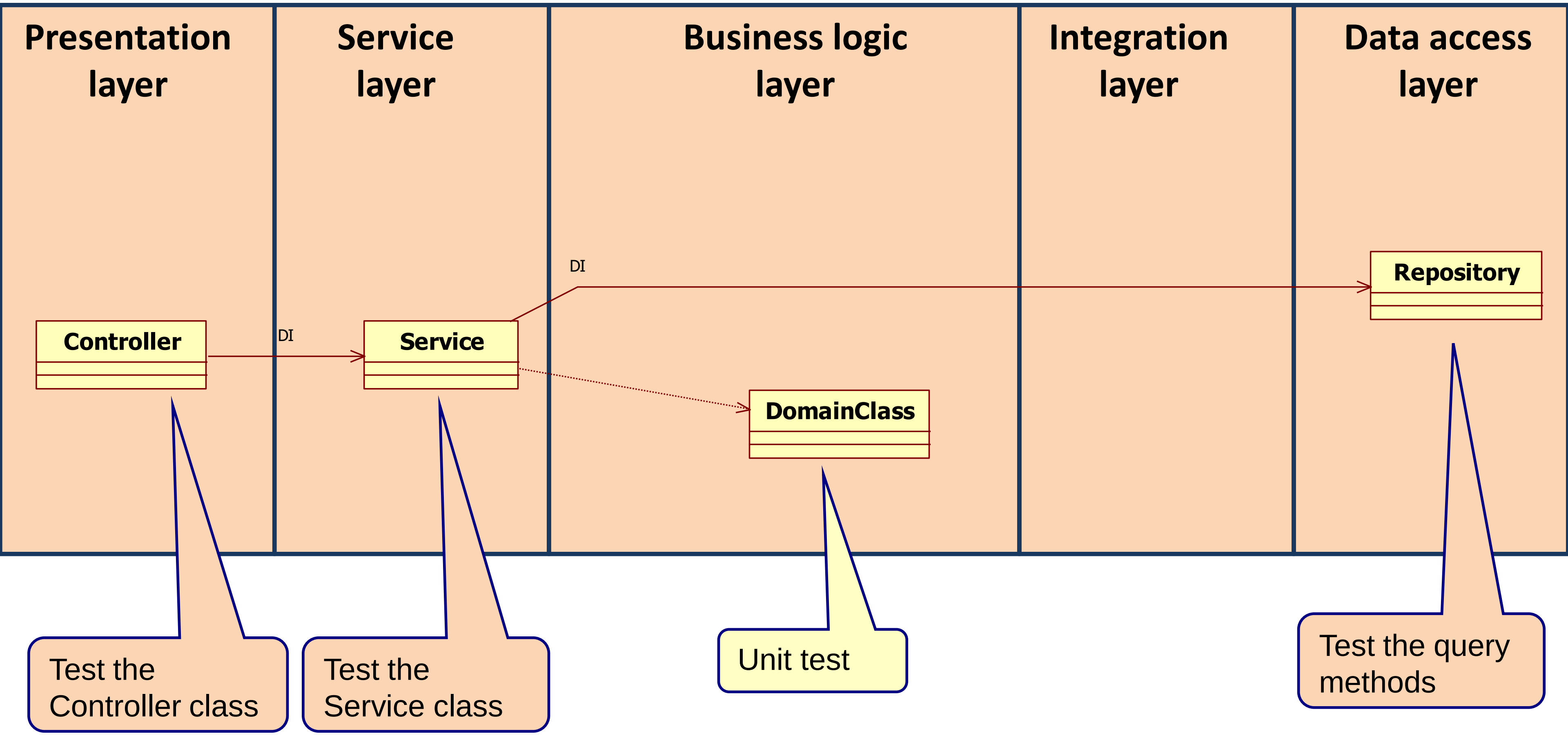
Treat test code as production code

Keep your tests

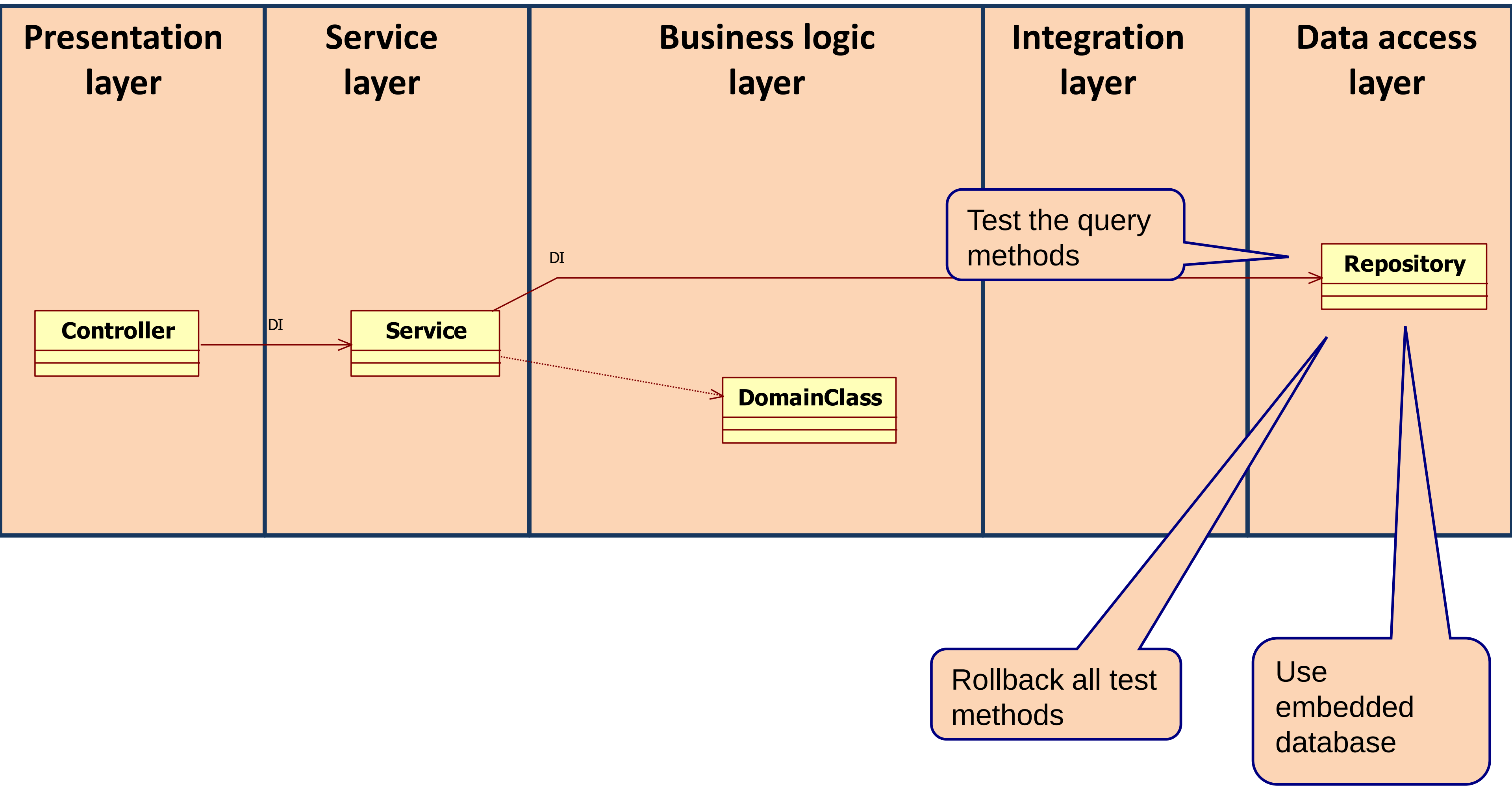
- Simple
- Short
- Understandable
- Loosely coupled

SPRING TESTING

Spring testing



Test the repository



Testing the repository

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {

  Customer findByName(String name);
}
```

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class CustomersRepositoryTests {
    @Autowired
    private TestEntityManager entityManager;
    @Autowired
    private CustomerRepository customerRepository;

    @Test
    public void whenFindByName_thenReturnEmployee() {
        // given
        Customer frank = new Customer(123L,"Frank Brown","fbrown@gmail.com");
        entityManager.persist(frank);
        entityManager.flush();
        // when
        Customer found = customerRepository.findByName(frank.getName());
        // then
        assertThat(found.getName())
          .isEqualTo(frank.getName());
    }
}
```

Auto configure JPA
- Scan entities
- Setup database and datasource
- Create entityManager
- Create repository

Data JPA tests are transactional and rolled back at the end of each test

Use the entityManager to persist a Customer

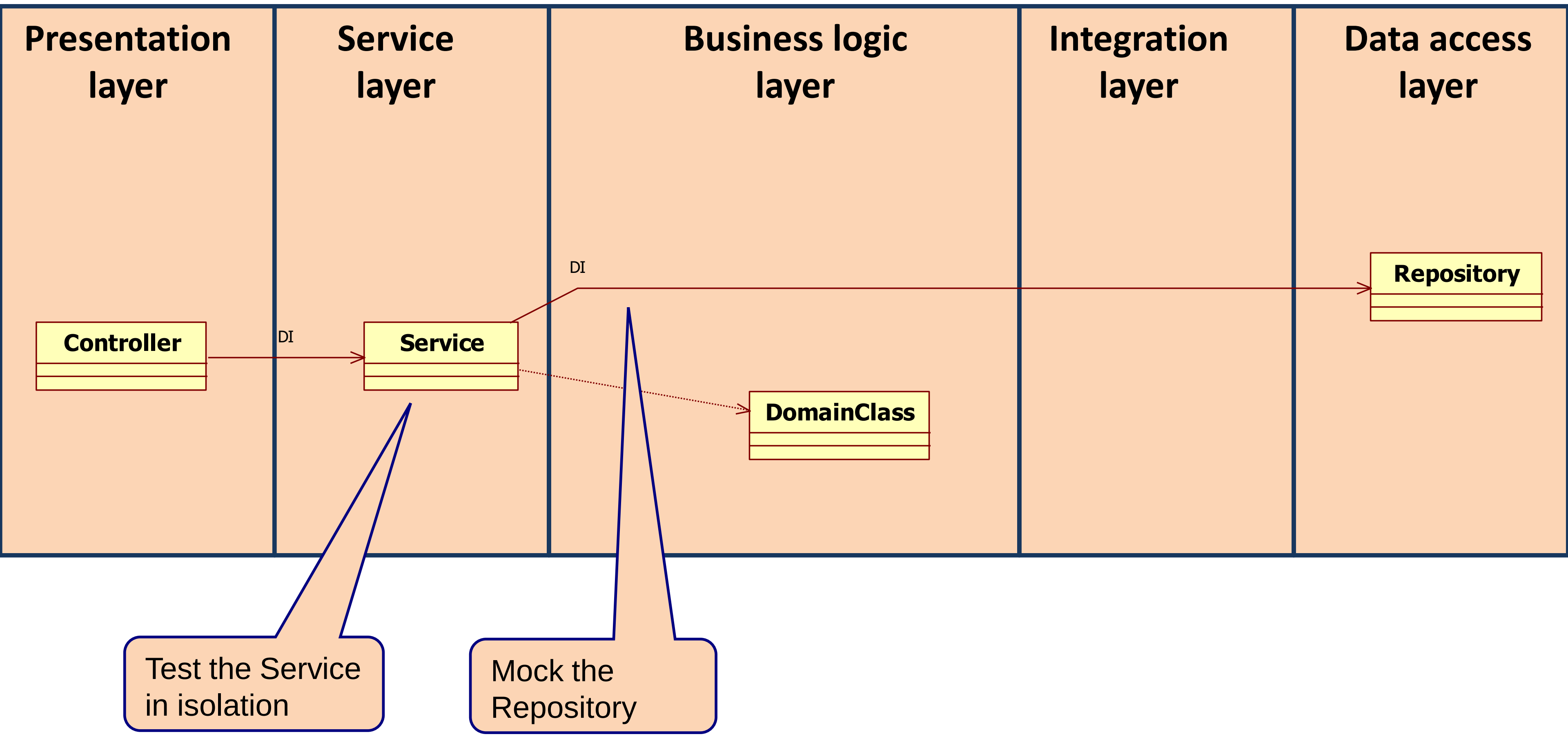
Call the method on the repository

Using an embedded database

```xml
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>test</scope>
  <version>1.4.194</version>
</dependency>
```

```
Replacing 'dataSource' DataSource bean with embedded versionStarting embedded database:
url='jdbc:h2:mem:cda533b4-a53f-4fb6-8f00-8a608a533537;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=false',
username='sa'
Hibernate: drop table customer if exists
Hibernate: create table customer (customer_number bigint not null, email varchar(255), name
varchar(255), primary key (customer_number))
Started CustomersRepositoryTests in 3.128 seconds (JVM running for 3.832)
Began transaction (1) for test context
Hibernate: insert into customer (email, name, customer_number) values (?, ?, ?)
Hibernate: select customer0_.customer_number as customer1_0_, customer0_.email as email2_0_,
customer0_.name as name3_0_ from customer customer0_ where customer0_.name=?
Rolled back transaction for test:
Closing JPA EntityManagerFactory for persistence unit 'default'
Hibernate: drop table customer if exists
```

Test the Service



Testing the service

```
public class CustomerService {
  @Autowired
  CustomerRepository customerRepository;

  public Customer findCustomer(String customerNumber) {
    Optional<Customer> customerOptional =
      customerRepository.findById(Long.valueOf(customerNumber));
    return customerOptional.get();
  }

...
}
```

We need to mock the customerRepository

Testing the service (1/2)

```java
@RunWith(SpringRunner.class)
public class CustomerServiceTests {

    @TestConfiguration
    static class CustomerServiceImplTestContextConfiguration {

        @Bean
        public CustomerService customerService() {
            return new CustomerService();
        }
    }

    @Autowired
    private CustomerService customerService;

    @MockBean
    private CustomerRepository customerRepository;
```

Create an ApplicationContext with only a CustomerService

Get the customerService from the context

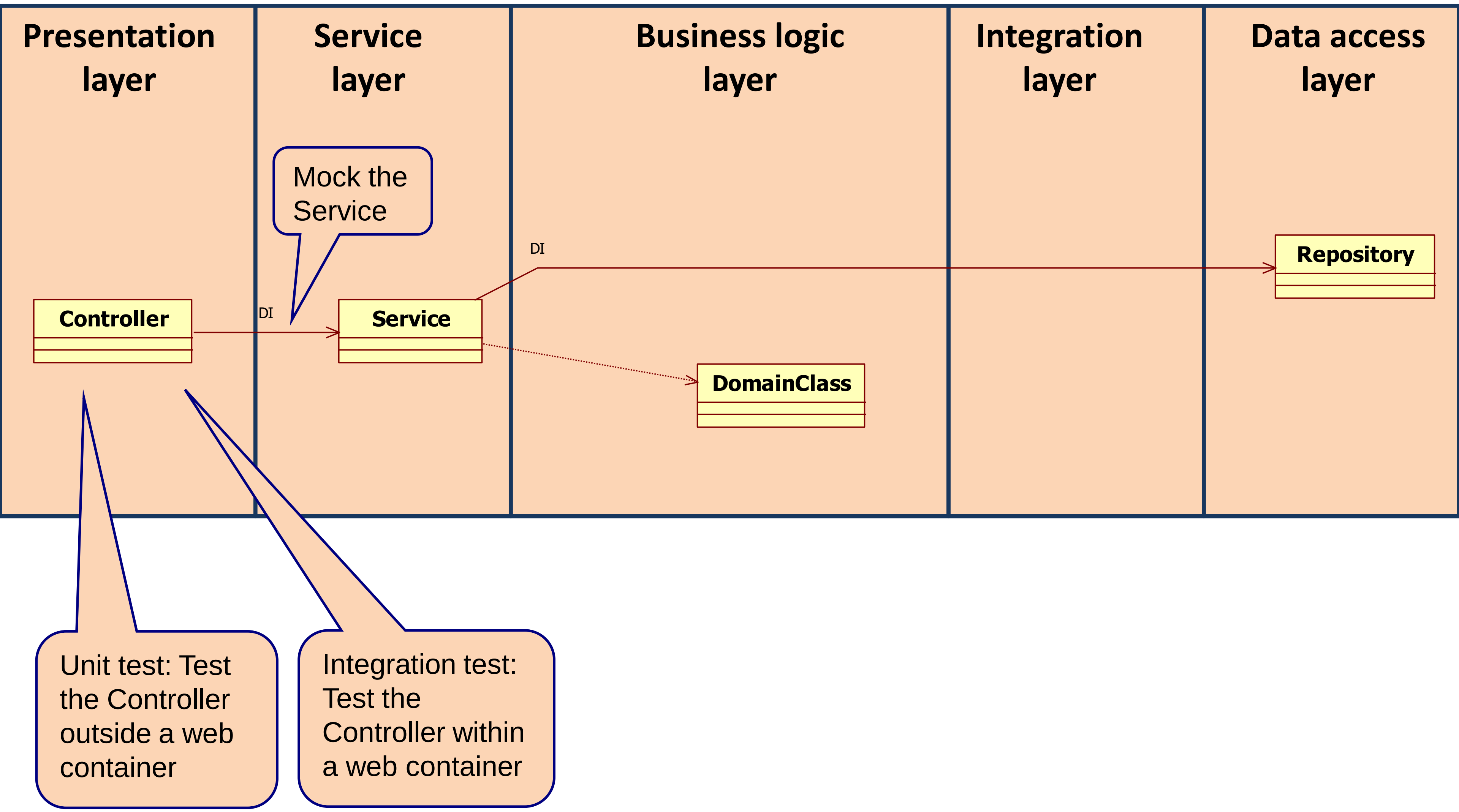
Create a mock of type CustomerRepository

Testing the service (2/2)

```
    @Before
    public void setUp() {
      Long customerNumber = 123L;
      Customer frank = new Customer(customerNumber,"Frank Brown","fbrown@gmail.com");
      Optional<Customer> frankOptional = Optional.of(frank);

        Mockito.when(customerRepository.findById(customerNumber))
           .thenReturn(frankOptional);
    }

    @Test
    public void whenValidCustomerNumberThenCustomerShouldBeFound() {
      Long customerNumber = 123L;
      Customer found = customerService.findCustomer(customerNumber+"");

       assertThat(found.getCustomerNumber())
          .isEqualTo(customerNumber);
     }
}
```

Tell the mock what to do

Test the Controller



Testing the controller

```
@RestController
public class CustomerController {
  @Autowired
  CustomerService customerService;

  @GetMapping("/customer/{customerNumber}")
  public Customer getCustomer(@PathVariable String customerNumber) {
    return customerService.findCustomer(customerNumber);
  }
  @DeleteMapping("/customer/{customerNumber}")
  @ResponseStatus(HttpStatus.OK)
  public void deleteCustomer(@PathVariable String customerNumber) {
    customerService.removeCustomer(customerNumber);
  }
  @PostMapping("/customer")
  @ResponseStatus(HttpStatus.OK)
  public void addCustomer(@RequestBody Customer customer) {
    customerService.addCustomer(customer);
  }
  @PutMapping("/customer")
  @ResponseStatus(HttpStatus.OK)
  public void updateCustomer(@RequestBody Customer customer) {
    customerService.updateCustomer(customer);
  }
  @GetMapping("/customers")
  public Customers getAllCustomers() {
    return customerService.getAllCustomers();
  }
}
```

We need to mock the customerService

Testing the controller outside the container

```
@RunWith(SpringRunner.class)
@WebMvcTest(CustomerController.class)
public class CustomerControllerTest {

  @Autowired
  MockMvc mock;

  @MockBean
  CustomerService customerService;

  @Test
  public void testGetCustomerByCustomerNumber() throws Exception {
    Mockito.when(customerService.findCustomer("1")).thenReturn(new Customer(1L, "Frank
                                Brown", "fbrown@gmail.com"));
    mock.perform(get("/customer/1"))
      .andExpect(status().isOk())
      .andExpect(MockMvcResultMatchers.jsonPath("$.customerNumber").value(1L))
      .andExpect(MockMvcResultMatchers.jsonPath("$.name").value("Frank Brown"))
      .andExpect(MockMvcResultMatchers.jsonPath("$.email").value("fbrown@gmail.com"));

  }
```

Apply only configuration relevant to Mvc tests

This mock calls the controller class in the same way as you do with HTTP, but now without a server

Create a mock of type CustomerService

Tell the customerService mock how to behave

Testing the controller: delete

```java
@DeleteMapping("/customer/{customerNumber}")
@ResponseStatus(HttpStatus.OK)
public void deleteCustomer(@PathVariable String customerNumber) {
   customerService.removeCustomer(customerNumber);
}
```

```java
@Test
public void testDeleteCustomerByCustomerNumber() throws Exception {
  mock.perform(MockMvcRequestBuilders.delete("/customer/{id}",1))
    .andExpect(status().isOk());

  verify(customerService, times(1)).removeCustomer("1");
}
```

Testing the controller: post

```
@PostMapping("/customer")
@ResponseStatus(HttpStatus.OK)
public void addCustomer(@RequestBody Customer customer) {
  customerService.addCustomer(customer);
}
```

```
@Test
public void testAddCustomer() throws Exception {
  Customer customer = new Customer(1L, "Frank Brown", "fbrown@gmail.com");
  mock.perform(MockMvcRequestBuilders.post("/customer")
      .content(asJsonString(customer))
      .contentType(MediaType.APPLICATION_JSON))
          .andExpect(status().isOk());

  verify(customerService, times(1)).addCustomer(customer);
}

public static String asJsonString(final Object obj) {
  try {
    return new ObjectMapper().writeValueAsString(obj);
  } catch (Exception e) {
    throw new RuntimeException(e);
  }
}
```

Testing the controller: put

```java
@PutMapping("/customer")
@ResponseStatus(HttpStatus.OK)
public void updateCustomer(@RequestBody Customer customer) {
  customerService.updateCustomer(customer);
}
```

```java
@Test
public void testUpdateCustomer() throws Exception {
  Customer customer = new Customer(1L, "Frank Brown", "fbrown@gmail.com");
  mock.perform(MockMvcRequestBuilders.put("/customer")
      .content(asJsonString(customer))
      .contentType(MediaType.APPLICATION_JSON))
          .andExpect(status().isOk());

  verify(customerService, times(1)).updateCustomer(customer);
}
```

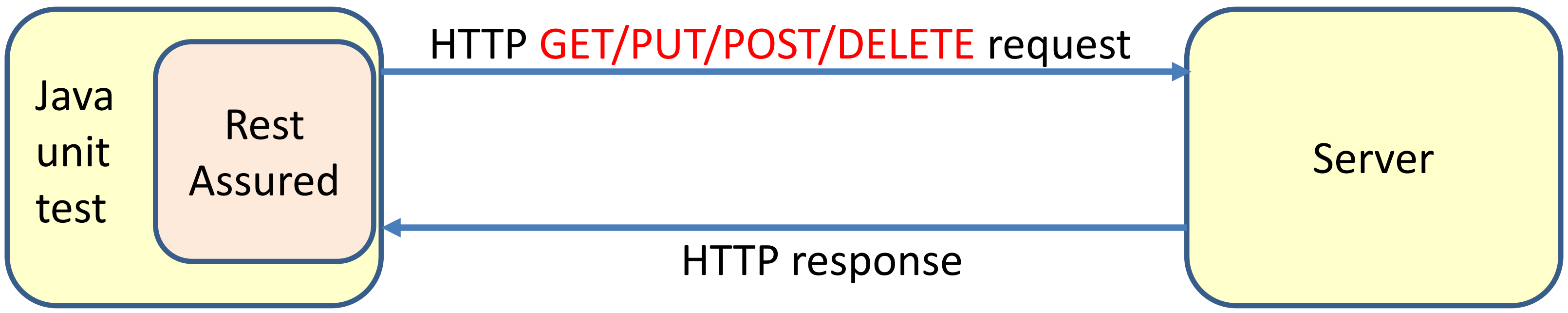
Testing the controller: get all customers

```java
@GetMapping("/customers")
public Customers getAllCustomers() {
  return customerService.getAllCustomers();
}
```

```java
@Test
public void testGetallCustomers() throws Exception {
  Customers customers= new Customers();
  customers.addCustomer(new Customer(1L, "Frank Brown", "fbrown@gmail.com"));
  customers.addCustomer(new Customer(2L, "John Doe", "jdoe@gmail.com"));
  Mockito.when(customerService.getAllCustomers()).thenReturn(customers);

  mock.perform(MockMvcRequestBuilders.get("/customers"))
    .andExpect(status().isOk())
          .andExpect(MockMvcResultMatchers.jsonPath("$.customers").isArray())
          .andExpect(MockMvcResultMatchers.jsonPath("$.customers", hasSize(2)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.customers[0].customerNumber").value(1L))
    .andExpect(MockMvcResultMatchers.jsonPath("$.customers[0].name").value("Frank Brown"))
    .andExpect(MockMvcResultMatchers.jsonPath("$.customers[0].email").value("fbrown@gmail.com"));

  verify(customerService, times(1)).getAllCustomers();
}
```

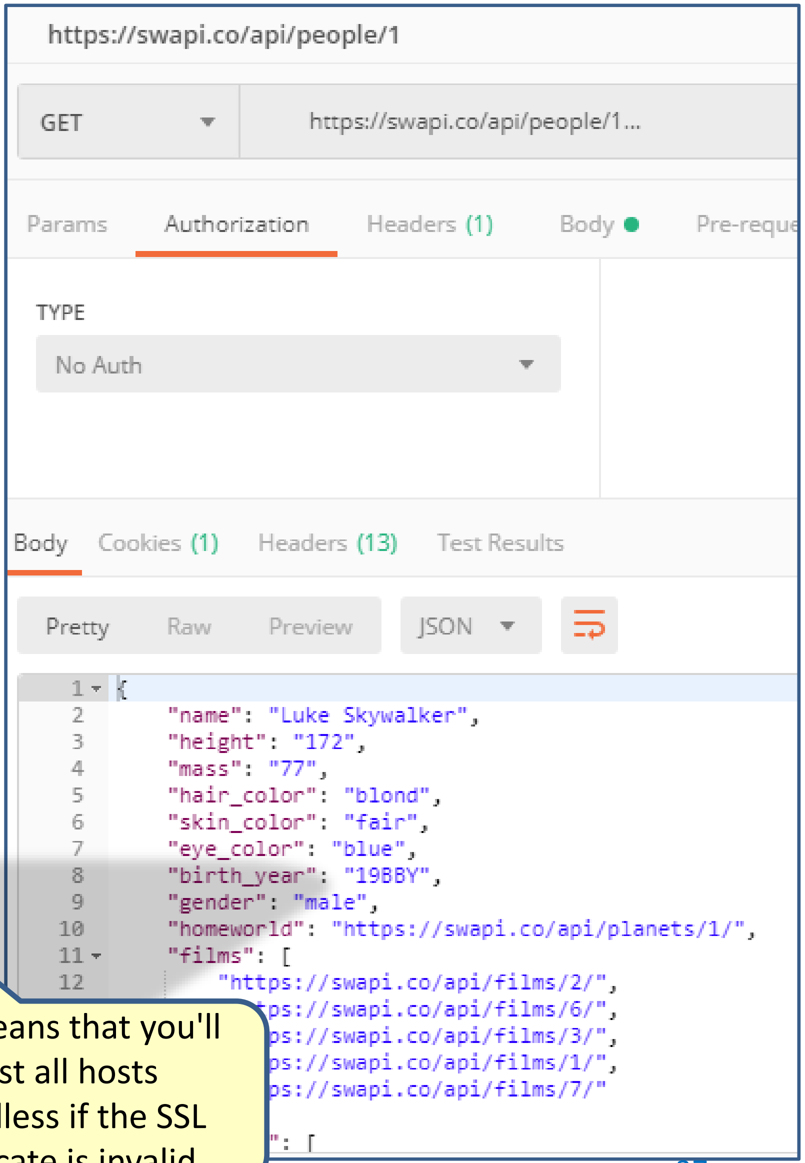
RESTASSURED

REST client


Java unit test
Rest Assured
HTTP GET/PUT/POST/DELETE request
HTTP response
Server

RestAssured example

```java
import org.junit.BeforeClass;
import org.junit.Test;
import io.restassured.RestAssured;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.equalTo;

public class RestTest {

  @BeforeClass
  public static void setup() {
        RestAssured.port = Integer.valueOf(8080);
        RestAssured.baseURI = "http://swapi.co";
        RestAssured.basePath = "/api/people/";
  }

  @Test
  public void test() {
    given()
      .relaxedHTTPSValidation("TLSv1.2")
      .when()
      .get("1")
      .then()
      .body("name",equalTo("Luke Skywalker"));
  }
}
```


https://swapi.co/api/people/1

GET https://swapi.co/api/people/1...

Params | Authorization | Headers (1) | Body ● | Pre-reque

TYPE: No Auth

Body | Cookies (1) | Headers (13) | Test Results

Pretty | Raw | Preview | JSON

```
{
    "name": "Luke Skywalker",
    "height": "172",
    "mass": "77",
    "hair_color": "blond",
    "skin_color": "fair",
    "eye_color": "blue",
    "birth_year": "19BBY",
    "gender": "male",
    "homeworld": "https://swapi.co/api/planets/1/",
    "films": [
        "https://swapi.co/api/films/2/",
        tps://swapi.co/api/films/6/",
        ps://swapi.co/api/films/3/",
        ps://swapi.co/api/films/1/",
        ps://swapi.co/api/films/7/"

        ": [
```

This means that you'll trust all hosts regardless if the SSL certificate is invalid.

statusCode

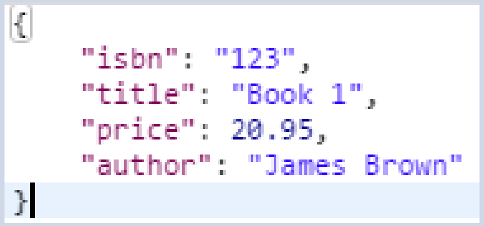
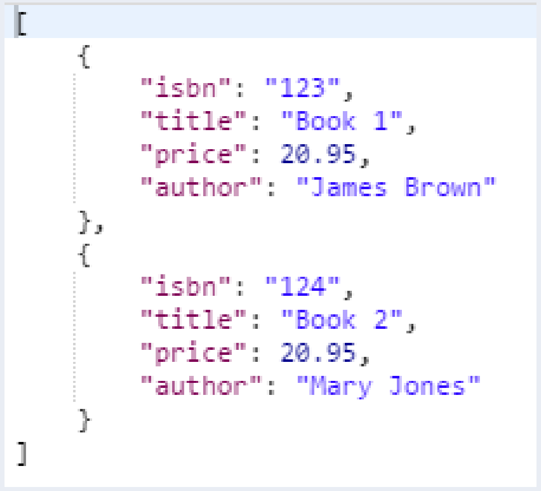
```
@Test
public void testStatusLuke() {
  given()
    .relaxedHTTPSValidation("TLSv1.2")
    .when()
    .get("1")
    .then()
    .statusCode(200)
    .body("name",equalTo("Luke Skywalker"));
}
```

```
@Test
public void testStatusLuke() {
  given()
    .relaxedHTTPSValidation("TLSv1.2")
    .when()
    .get("123")
    .then()
    .statusCode(404);
}
```

contentType

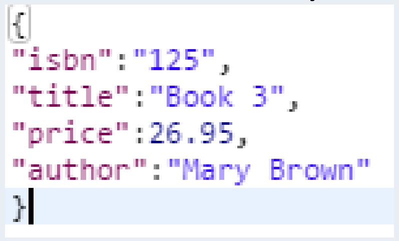
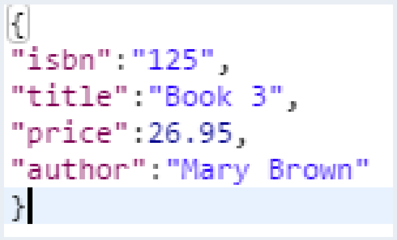
```
@Test
public void test() {
  given().relaxedHTTPSValidation("TLSv1.2")
   .when()
   .get("1")
   .then()
   .contentType(ContentType.JSON)
   .and()
   .body("name",equalTo("Luke Skywalker"));
}
```

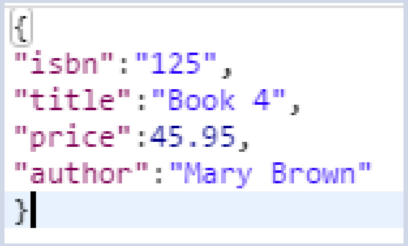
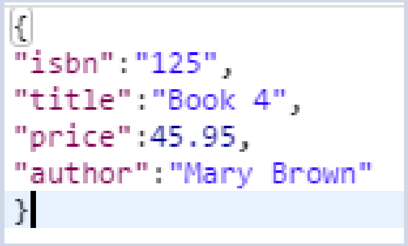
Example REST Bookservice

Request	Response
GET /book/{isbn} Get localhost:8081/book/123	Return book with this isbn  `{ "isbn": "123", "title": "Book 1", "price": 20.95, "author": "James Brown" }`
GET /books Get localhost:8081/books	Return all books  `[ { "isbn": "123", "title": "Book 1", "price": 20.95, "author": "James Brown" }, { "isbn": "124", "title": "Book 2", "price": 20.95, "author": "Mary Jones" } ]`

Example REST Bookservice

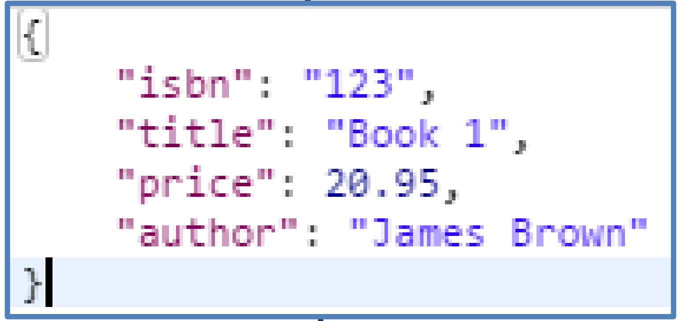
Request	Response
DELETE /book/{isbn} *DELETE localhost:8081/book/123*	Delete book with this isbn

POST /book
 	
POST localhost:8081/book

{
"isbn":"125",
"title":"Book 3",
"price":26.95,
"author":"Mary Brown"
}
	Add new book
	
	{
	"isbn":"125",
	"title":"Book 3",
	"price":26.95,
	"author":"Mary Brown"
	}

PUT /book
 	
PUT localhost:8081/book

{
"isbn":"125",
"title":"Book 4",
"price":45.95,
"author":"Mary Brown"
}
	Update existing book
	
	{
	"isbn":"125",
	"title":"Book 4",
	"price":45.95,
	"author":"Mary Brown"
	}

Get one book

```java
public class BookTest {

  @BeforeClass
  public static void setup() {
    RestAssured.port = Integer.valueOf(8081);
    RestAssured.baseURI = "http://localhost/";
    RestAssured.basePath = "";
  }

  @Test
  public void testGetOneBook() {
    given()
    .when()
    .get("book/123")
    .then()
    .contentType(ContentType.JSON)
    .and()
    .body("isbn",equalTo("123"))
    .body("title",equalTo("Book 1"))
    .body("price",equalTo(20.95f))
    .body("author",equalTo("James Brown"));
  }
}
```

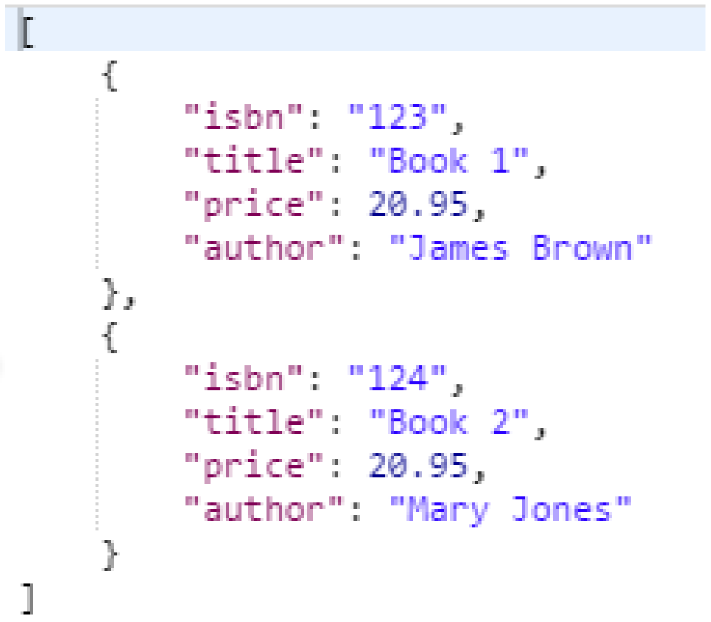


```json
{
    "isbn": "123",
    "title": "Book 1",
    "price": 20.95,
    "author": "James Brown"
}
```

Use **f** for real numbers

Get all books: test isbn

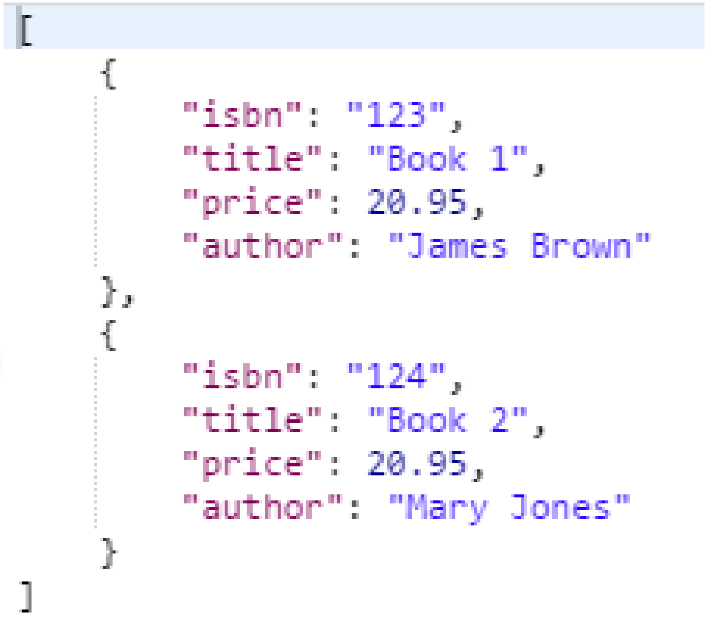
```java
@Test
public void testIsbnAllBooks() {
   given()
    .when()
    .get("books")
    .then()
    .contentType(ContentType.JSON)
    .body("isbn", hasItems("123", "124"));
}
```



```json
[
    {
        "isbn": "123",
        "title": "Book 1",
        "price": 20.95,
        "author": "James Brown"
    },
    {
        "isbn": "124",
        "title": "Book 2",
        "price": 20.95,
        "author": "Mary Jones"
    }
]
```

Get all books: test number of books

```java
@Test
public void testNumberOfAllBooks() {
   given()
     .when()
     .get("books")
     .then()
     .contentType(ContentType.JSON)
     .body("isbn", hasSize(2));
}
```



```json
[
    {
        "isbn": "123",
        "title": "Book 1",
        "price": 20.95,
        "author": "James Brown"
    },
    {
        "isbn": "124",
        "title": "Book 2",
        "price": 20.95,
        "author": "Mary Jones"
    }
]
```

Delete

```
@Test
public void testDelete() {
   // add the to be deleted book
   Book book = new Book("123", "Book 1", 20.95, "James Brown");
   given()
      .contentType("application/json")
      .body(book)
      .when().post("/book").then()
      .statusCode(200);

   given()
     .when()
     .delete("book/123");

   given()
     .when()
     .get("books")
     .then()
     .body("isbn", hasSize(1));

}
```

Add book with isbn “123”

Delete book with isbn “123”

Test the number of books

Post

```java
@Test
public void testPost() {
  Book book = new Book("234", "Book 3", 34.75, "Jack Johnson");

  given()
    .contentType("application/json")
    .body(book)
    .when().post("/book").then()
    .statusCode(200);

  given()
    .when()
    .get("books")
    .then()
    .contentType(ContentType.JSON)
    .body("isbn", hasItems("123", "124", "234"));

  //delete the book again
  given()
    .when()
    .delete("book/234");
}
```

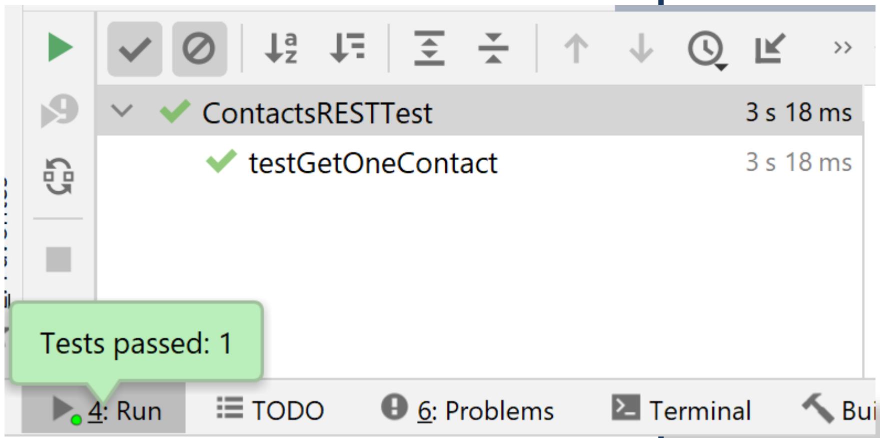
Add book with isbn “234”

Test if the books is added

Delete book with isbn “234”

GET contact

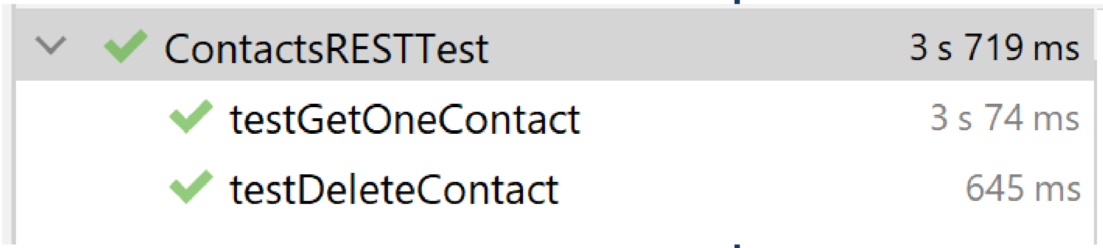
```
public class ContactsRESTTest {
  @BeforeClass
  public static void setup() {
    RestAssured.port = Integer.valueOf(8080);
    RestAssured.baseURI = "http://localhost";
    RestAssured.basePath = "";
  }
@Test
public void testGetOneContact() {
  // add the contact to be fetched
  Contact contact = new Contact("Mary", "Jones", "mjones@acme.com", "2341674376");
  given()
        .contentType("application/json")
        .body(contact)
        .when().post("/contacts").then()
        .statusCode(200);
  // test getting the contact
  given()
        .when()
        .get("contacts/Mary")
        .then()
        .contentType(ContentType.JSON)
        .and()
        .body("firstName",equalTo("Mary"))
        .body("lastName",equalTo("Jones"))
        .body("email",equalTo("mjones@acme.com"))
        .body("phone",equalTo("2341674376"));
  //cleanup
  given()
        .when()
        .delete("contacts/Mary");
}
}
```



```
ContactsRESTTest        3 s 18 ms
   testGetOneContact    3 s 18 ms

Tests passed: 1
```

DELETE contact

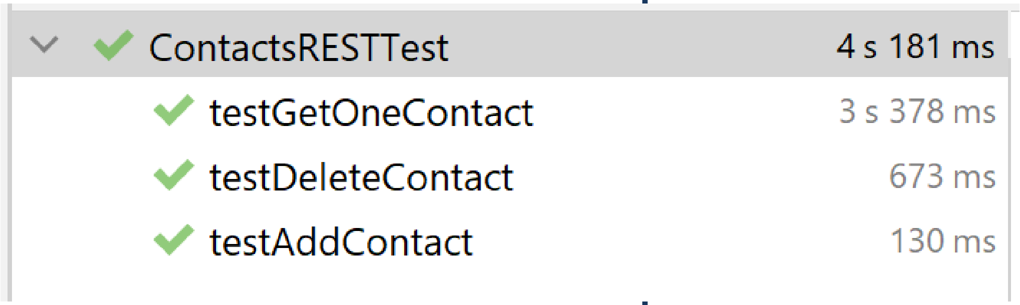
```
@Test
public void testDeleteContact() {
   // add the contact to be deleted book
   Contact contact = new Contact("Bob", "Smith", "bobby@hotmail.com", "76528765498");
   given()
         .contentType("application/json")
         .body(contact)
         .when().post("/contacts").then()
         .statusCode(200);

   given()
         .when()
         .delete("contacts/Bob");

   given()
         .when()
         .get("contacts/Bob")
         .then()
         .statusCode(404)
         .and()
         .body("errorMessage",equalTo("Contact with firstname= Bob is not available"));
}
```



```
✓ ContactsRESTTest          3 s 719 ms
    ✓ testGetOneContact       3 s 74 ms
    ✓ testDeleteContact         645 ms
```

POST contact

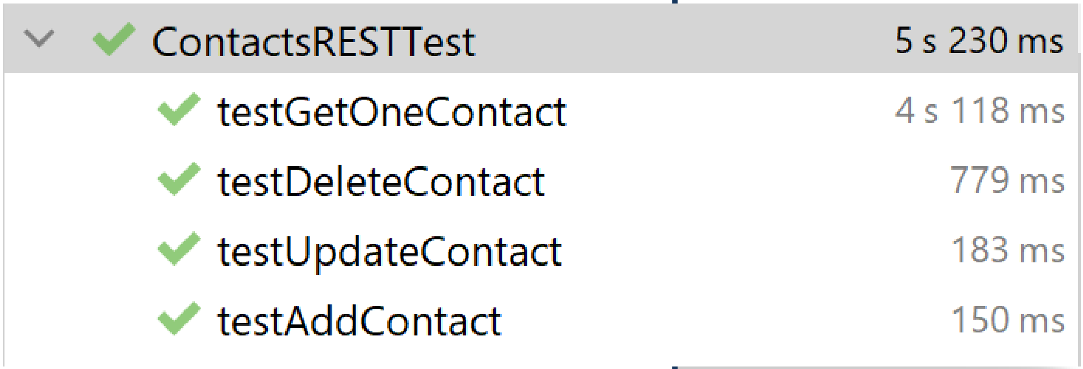
```
@Test
public void testAddContact() {
   // add the contact
   Contact contact = new Contact("Bob", "Smith", "bobby@hotmail.com", "76528765498");
   given()
         .contentType("application/json")
         .body(contact)
         .when().post("/contacts").then()
         .statusCode(200);
   // get the contact and verify
   given()
         .when()
         .get("contacts/Bob")
         .then()
         .statusCode(200)
         .and()
         .body("firstName",equalTo("Bob"))
         .body("lastName",equalTo("Smith"))
         .body("email",equalTo("bobby@hotmail.com"))
         .body("phone",equalTo("76528765498"));
   //cleanup
   given()
         .when()
         .delete("contacts/Bob");
}
```



```
✓ ContactsRESTTest        4 s 181 ms
   ✓ testGetOneContact    3 s 378 ms
   ✓ testDeleteContact       673 ms
   ✓ testAddContact          130 ms
```

PUT contact

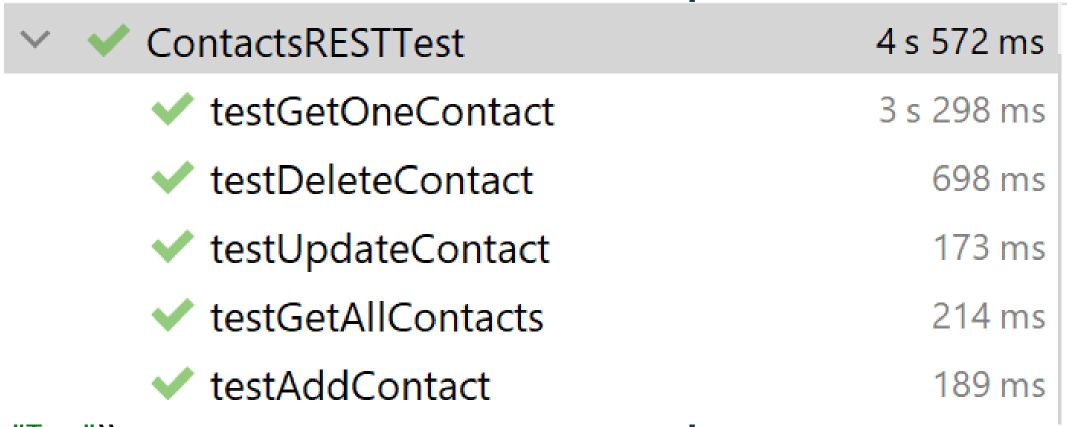
```
@Test
public void testUpdateContact() {
    // add the contact
    Contact contact = new Contact("Bob", "Smith", "bobby@hotmail.com", "76528765498");
    Contact updateContact = new Contact("Bob", "Johnson", "bobby@gmail.com", "89765123");
    given()
            .contentType("application/json")
            .body(contact)
            .when().post("/contacts").then()
            .statusCode(200);
    //update contact
    given()
            .contentType("application/json")
            .body(updateContact)
            .when().put("/contacts/"+updateContact.getFirstName()).then()
            .statusCode(200);
    // get the contact and verify
    given()
            .when()
            .get("contacts/Bob")
            .then()
            .statusCode(200)
            .and()
            .body("firstName",equalTo("Bob"))
            .body("lastName",equalTo("Johnson"))
            .body("email",equalTo("bobby@gmail.com"))
            .body("phone",equalTo("89765123"));
    //cleanup
    given()
            .when()
            .delete("contacts/Bob");
}
```



```
✓ ContactsRESTTest        5 s 230 ms
    ✓ testGetOneContact   4 s 118 ms
    ✓ testDeleteContact       779 ms
    ✓ testUpdateContact       183 ms
    ✓ testAddContact          150 ms
```

Get all contacts

```java
@Test
public void testGetAllContacts() {
    // add the contacts
    Contact contact = new Contact("Bob", "Smith", "bobby@hotmail.com", "76528765498");
    Contact contact2 = new Contact("Tom", "Johnson", "tomjohnson@gmail.com", "543256789");
    given()
        .contentType("application/json")
        .body(contact)
        .when().post("/contacts").then()
        .statusCode(200);
    given()
        .contentType("application/json")
        .body(contact2)
        .when().post("/contacts").then()
        .statusCode(200);
    // get all contacts and verify
    given()
        .when()
        .get("contacts")
        .then()
        .statusCode(200)
        .and()
        .body("contacts.firstName", hasItems("Bob", "Tom"))
        .body("contacts.lastName",hasItems("Smith", "Johnson"))
        .body("contacts.email",hasItems("bobby@hotmail.com", "tomjohnson@gmail.com"))
        .body("contacts.phone",hasItems("76528765498", "543256789"));
    //cleanup
    given()
        .when()
        .delete("contacts/Bob");
    given()
        .when()
        .delete("contacts/Tom");
```



```
✓ ContactsRESTTest            4 s 572 ms
    ✓ testGetOneContact       3 s 298 ms
    ✓ testDeleteContact          698 ms
    ✓ testUpdateContact          173 ms
    ✓ testGetAllContacts         214 ms
    ✓ testAddContact             189 ms
```