**This project is simply an extension of project 'SpringHibernateExample' with additional functionality of testing. So you are requested to go through 'SpringHibernateExample' project very thoroughly before moving ahead.**

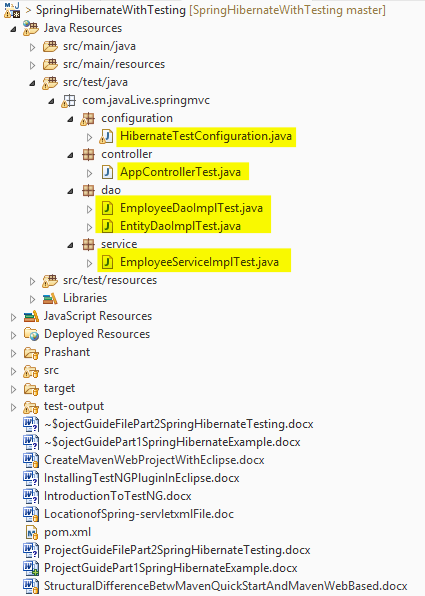
In this project we will learn how to add unit and integration test in SpringMVC and hibernate based maven project using TestNG, mockito, spring-test, DBUnit & H2 database. To know more about testing with TestNG in general, please refer files ' IntroductionToTestNG.docx' and 'InstallingTestNGPluginInEclipse.docx' before going ahead.

* Technologies used in the project:
* Spring 4.0.6.RELEASE
* Hibernate Core 4.3.6.Final
* validation-api 1.1.0.Final
* hibernate-validator 5.1.3.Final
* MySQL Server 5.6
* Maven 3
* JDK 1.7
* Tomcat 8.0.21
* Eclipse JUNO Service Release 2
* TestNG 6.9.4
* Mockito 1.10.19
* DBUnit 2.2
* H2 Database 1.4.187
* Let’s begin

Step 1: Adapt directory structure

It’s same directory structure as in project 'SpringHibernateExample'. What’s new is that we have added src/test/java & src/test/resources folders and testing artifacts. We will maintain the same package structure in test as in src folder.

Following will be the final project structure:[Note that src/main part is hidden in order to show everything in one image]



Step 2: Review pom.xml

This is same pom.xml as declared in project 'SpringHibernateExample'.

|  |
| --- |
| <?xml version="1.0"?>  <project xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>"      xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>">        <modelVersion>4.0.0</modelVersion>      <groupId>com.javalive.springmvc</groupId>      <artifactId>SpringHibernateWithTesting</artifactId>      <packaging>war</packaging>      <version>1.0.0</version>      <name>SpringHibernateWithTesting</name>        <properties>          <springframework.version>4.0.6.RELEASE</springframework.version>          <hibernate.version>4.3.6.Final</hibernate.version>          <mysql.connector.version>5.1.31</mysql.connector.version>          <joda-time.version>2.3</joda-time.version>          <testng.version>6.9.4</testng.version>          <mockito.version>1.10.19</mockito.version>          <h2.version>1.4.187</h2.version>          <dbunit.version>2.2</dbunit.version>      </properties>        <dependencies>          <!-- Spring -->          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-core</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-web</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-webmvc</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-tx</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-orm</artifactId>              <version>${springframework.version}</version>          </dependency>            <!-- Hibernate -->          <dependency>              <groupId>org.hibernate</groupId>              <artifactId>hibernate-core</artifactId>              <version>${hibernate.version}</version>          </dependency>            <!-- jsr303 validation -->          <dependency>              <groupId>javax.validation</groupId>              <artifactId>validation-api</artifactId>              <version>1.1.0.Final</version>          </dependency>          <dependency>              <groupId>org.hibernate</groupId>              <artifactId>hibernate-validator</artifactId>              <version>5.1.3.Final</version>          </dependency>            <!-- MySQL -->          <dependency>              <groupId>mysql</groupId>              <artifactId>mysql-connector-java</artifactId>              <version>${mysql.connector.version}</version>          </dependency>            <!-- Joda-Time -->          <dependency>              <groupId>joda-time</groupId>              <artifactId>joda-time</artifactId>              <version>${joda-time.version}</version>          </dependency>            <!-- To map JodaTime with database type -->          <dependency>              <groupId>org.jadira.usertype</groupId>              <artifactId>usertype.core</artifactId>              <version>3.0.0.CR1</version>          </dependency>            <!-- Servlet+JSP+JSTL -->          <dependency>              <groupId>javax.servlet</groupId>              <artifactId>javax.servlet-api</artifactId>              <version>3.1.0</version>          </dependency>          <dependency>              <groupId>javax.servlet.jsp</groupId>              <artifactId>javax.servlet.jsp-api</artifactId>              <version>2.3.1</version>          </dependency>          <dependency>              <groupId>javax.servlet</groupId>              <artifactId>jstl</artifactId>              <version>1.2</version>          </dependency>              <!-- Testing dependencies -->          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-test</artifactId>              <version>${springframework.version}</version>              <scope>test</scope>          </dependency>          <dependency>              <groupId>org.testng</groupId>              <artifactId>testng</artifactId>              <version>${testng.version}</version>              <scope>test</scope>          </dependency>          <dependency>              <groupId>org.mockito</groupId>              <artifactId>mockito-all</artifactId>              <version>${mockito.version}</version>              <scope>test</scope>          </dependency>          <dependency>              <groupId>com.h2database</groupId>              <artifactId>h2</artifactId>              <version>${h2.version}</version>              <scope>test</scope>          </dependency>          <dependency>              <groupId>dbunit</groupId>              <artifactId>dbunit</artifactId>              <version>${dbunit.version}</version>              <scope>test</scope>          </dependency>        </dependencies>        <build>          <pluginManagement>              <plugins>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-compiler-plugin</artifactId>                      <version>3.2</version>                      <configuration>                          <source>1.7</source>                          <target>1.7</target>                      </configuration>                  </plugin>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-war-plugin</artifactId>                      <version>2.4</version>                      <configuration>                          <warSourceDirectory>src/main/webapp</warSourceDirectory>                          <warName>SpringHibernateWithTesting</warName>                          <failOnMissingWebXml>false</failOnMissingWebXml>                      </configuration>                  </plugin>              </plugins>          </pluginManagement>          <finalName>SpringHibernateWithTesting</finalName>      </build>  </project> |

Just to recall that dependencies spring-test, testNG, mockito, h2 & DBUnit are all here for Testing purpose only.

Spring-test : We will be using spring-test annotations in our test classes.  
TestNG : We will be using TestNG as our testing framework (& it’s my favorite anyway).  
Mockito : We would be time to time doing some mocking, like mocking dao when testing service.  
DBUnit : We will use DBUnit to manage our data during data/dao layer testing  
H2 Database : For database layer, it’s more of integration-test than unit-test. IMO, unit tests does not bring real value while testing data layer. We will be using in-memory H2 database to do our integration-tests.

Let’s Start some real testing.

1) Testing Controllers:

|  |
| --- |
| package com.javalive.springmvc.controller;    import static org.mockito.Matchers.any;  import static org.mockito.Matchers.anyString;  import static org.mockito.Matchers.anyInt;  import static org.mockito.Mockito.doNothing;  import static org.mockito.Mockito.when;  import static org.mockito.Mockito.verify;    import java.math.BigDecimal;  import java.util.ArrayList;  import java.util.List;    import org.joda.time.LocalDate;  import org.mockito.InjectMocks;  import org.mockito.Mock;  import org.mockito.MockitoAnnotations;  import org.mockito.Spy;  import static org.mockito.Mockito.atLeastOnce;    import org.springframework.context.MessageSource;  import org.springframework.ui.ModelMap;  import org.springframework.validation.BindingResult;  import org.testng.Assert;  import org.testng.annotations.BeforeClass;  import org.testng.annotations.Test;      import com.javalive.springmvc.model.Employee;  import com.javalive.springmvc.service.EmployeeService;    public class AppControllerTest {        @Mock      EmployeeService service;        @Mock      MessageSource message;        @InjectMocks      AppController appController;        @Spy      List<Employee> employees = new ArrayList<Employee>();        @Spy      ModelMap model;        @Mock      BindingResult result;        @BeforeClass      public void setUp(){          MockitoAnnotations.initMocks(this);          employees = getEmployeeList();      }        @Test      public void listEmployees(){          when(service.findAllEmployees()).thenReturn(employees);          Assert.assertEquals(appController.listEmployees(model), "allemployees");          Assert.assertEquals(model.get("employees"), employees);          verify(service, atLeastOnce()).findAllEmployees();      }        @Test      public void newEmployee(){          Assert.assertEquals(appController.newEmployee(model), "registration");          Assert.assertNotNull(model.get("employee"));          Assert.assertFalse((Boolean)model.get("edit"));          Assert.assertEquals(((Employee)model.get("employee")).getId(), 0);      }          @Test      public void saveEmployeeWithValidationError(){          when(result.hasErrors()).thenReturn(true);          doNothing().when(service).saveEmployee(any(Employee.class));          Assert.assertEquals(appController.saveEmployee(employees.get(0), result, model), "registration");      }        @Test      public void saveEmployeeWithValidationErrorNonUniqueSSN(){          when(result.hasErrors()).thenReturn(false);          when(service.isEmployeeSsnUnique(anyInt(), anyString())).thenReturn(false);          Assert.assertEquals(appController.saveEmployee(employees.get(0), result, model), "registration");      }          @Test      public void saveEmployeeWithSuccess(){          when(result.hasErrors()).thenReturn(false);          when(service.isEmployeeSsnUnique(anyInt(), anyString())).thenReturn(true);          doNothing().when(service).saveEmployee(any(Employee.class));          Assert.assertEquals(appController.saveEmployee(employees.get(0), result, model), "success");          Assert.assertEquals(model.get("success"), "Employee Axel registered successfully");      }        @Test      public void editEmployee(){          Employee emp = employees.get(0);          when(service.findEmployeeBySsn(anyString())).thenReturn(emp);          Assert.assertEquals(appController.editEmployee(anyString(), model), "registration");          Assert.assertNotNull(model.get("employee"));          Assert.assertTrue((Boolean)model.get("edit"));          Assert.assertEquals(((Employee)model.get("employee")).getId(), 1);      }        @Test      public void updateEmployeeWithValidationError(){          when(result.hasErrors()).thenReturn(true);          doNothing().when(service).updateEmployee(any(Employee.class));          Assert.assertEquals(appController.updateEmployee(employees.get(0), result, model,""), "registration");      }        @Test      public void updateEmployeeWithValidationErrorNonUniqueSSN(){          when(result.hasErrors()).thenReturn(false);          when(service.isEmployeeSsnUnique(anyInt(), anyString())).thenReturn(false);          Assert.assertEquals(appController.updateEmployee(employees.get(0), result, model,""), "registration");      }        @Test      public void updateEmployeeWithSuccess(){          when(result.hasErrors()).thenReturn(false);          when(service.isEmployeeSsnUnique(anyInt(), anyString())).thenReturn(true);          doNothing().when(service).updateEmployee(any(Employee.class));          Assert.assertEquals(appController.updateEmployee(employees.get(0), result, model, ""), "success");          Assert.assertEquals(model.get("success"), "Employee Axel updated successfully");      }          @Test      public void deleteEmployee(){          doNothing().when(service).deleteEmployeeBySsn(anyString());          Assert.assertEquals(appController.deleteEmployee("123"), "redirect:/list");      }        public List<Employee> getEmployeeList(){          Employee e1 = new Employee();          e1.setId(1);          e1.setName("Axel");          e1.setJoiningDate(new LocalDate());          e1.setSalary(new BigDecimal(10000));          e1.setSsn("XXX111");            Employee e2 = new Employee();          e2.setId(2);          e2.setName("Jeremy");          e2.setJoiningDate(new LocalDate());          e2.setSalary(new BigDecimal(20000));          e2.setSsn("XXX222");            employees.add(e1);          employees.add(e2);          return employees;      }  } |

Run above test class using [TestNG Eclipse Plugin](http://testng.org/doc/eclipse.html) or Maven **[mvn clean test e.g.]**  
Here I used TestNG Eclipse plugin to run this test class.

|  |
| --- |
| PASSED: deleteEmployee  PASSED: editEmployee  PASSED: listEmployees  PASSED: newEmployee  PASSED: saveEmployeeWithSuccess  PASSED: saveEmployeeWithValidationError  PASSED: saveEmployeeWithValidationErrorNonUniqueSSN  PASSED: updateEmployeeWithSuccess  PASSED: updateEmployeeWithValidationError  PASSED: updateEmployeeWithValidationErrorNonUniqueSSN    ===============================================      Default test      Tests run: 10, Failures: 0, Skips: 0  =============================================== |

Explanation of AppControllerTest class:

If you revisit **AppController** class in project 'SpringHibernateExample', you will see that AppController basically depends on EmployeeService , MessageSource, Employee, ModelMap & BindingResult to fullfill all of it’s duties. Each of the AppController method is using only of these objects to do it’s real job.

So in order to test AppController, we would need to provide these dependencies. In our example, we do it using Mockito framework. We provide **mock** of EmployeeService & MessageSource by applying @Mock annotation on them. We also provide **spy** objects of ModelMap , BindingResult & Employee by applying @Spy annotations on them.

It’s important to understand that Mockito’s **@Mock** objects are not real instances, they are just bare-bones of instance created using Class of type. But their main capability is that they can remember all the interactions [operations performed] on them.

**@Spy** objects are on the other hand real instances, but with additional capabilities of remembering all the interactions [operations performed] on them.

@InjectMocks creates an instance of the class and injects the mocks that are created with the @Mock/@Spy objects in it.

MockitoAnnotations.initMocks(this); initializes objects annotated with Mockito annotations [@Mock, @Spy, @Captor, @InjectMocks]

Make sure to call **MockitoAnnotations.initMocks** when using Mockito annotations, else those mocks will be useless for your tests.

Annotations @Test & @BeforeClass are TestNG specific annotations.

Assert is the TestNG api for doing assertions on expected result and actual result.

**when..then & verify** are popular stubbing and verification techniques used in tests to define the behavior and then optionally verifying that behavior was indeed executed.

There are many more. Please refer to [TestNG tutorial](http://websystique.com/java/testng-tutorial/) for in depth details about writing tests using TestNG, mockito and other supportive libraries.

**2) Testing Service Layer:**

|  |
| --- |
| package com.javalive.springmvc.service;    import static org.mockito.Matchers.any;  import static org.mockito.Matchers.anyString;  import static org.mockito.Matchers.anyInt;  import static org.mockito.Mockito.atLeastOnce;  import static org.mockito.Mockito.doNothing;  import static org.mockito.Mockito.verify;    import java.math.BigDecimal;  import java.util.ArrayList;  import java.util.List;    import static org.mockito.Mockito.when;    import org.joda.time.LocalDate;  import org.mockito.InjectMocks;  import org.mockito.Mock;  import org.mockito.MockitoAnnotations;  import org.mockito.Spy;  import org.testng.Assert;  import org.testng.annotations.BeforeClass;  import org.testng.annotations.Test;    import com.javalive.springmvc.dao.EmployeeDao;  import com.javalive.springmvc.model.Employee;    public class EmployeeServiceImplTest {        @Mock      EmployeeDao dao;        @InjectMocks      EmployeeServiceImpl employeeService;        @Spy      List<Employee> employees = new ArrayList<Employee>();        @BeforeClass      public void setUp(){          MockitoAnnotations.initMocks(this);          employees = getEmployeeList();      }        @Test      public void findById(){          Employee emp = employees.get(0);          when(dao.findById(anyInt())).thenReturn(emp);          Assert.assertEquals(employeeService.findById(emp.getId()),emp);      }        @Test      public void saveEmployee(){          doNothing().when(dao).saveEmployee(any(Employee.class));          employeeService.saveEmployee(any(Employee.class));          verify(dao, atLeastOnce()).saveEmployee(any(Employee.class));      }        @Test      public void updateEmployee(){          Employee emp = employees.get(0);          when(dao.findById(anyInt())).thenReturn(emp);          employeeService.updateEmployee(emp);          verify(dao, atLeastOnce()).findById(anyInt());      }        @Test      public void deleteEmployeeBySsn(){          doNothing().when(dao).deleteEmployeeBySsn(anyString());          employeeService.deleteEmployeeBySsn(anyString());          verify(dao, atLeastOnce()).deleteEmployeeBySsn(anyString());      }        @Test      public void findAllEmployees(){          when(dao.findAllEmployees()).thenReturn(employees);          Assert.assertEquals(employeeService.findAllEmployees(), employees);      }        @Test      public void findEmployeeBySsn(){          Employee emp = employees.get(0);          when(dao.findEmployeeBySsn(anyString())).thenReturn(emp);          Assert.assertEquals(employeeService.findEmployeeBySsn(anyString()), emp);      }        @Test      public void isEmployeeSsnUnique(){          Employee emp = employees.get(0);          when(dao.findEmployeeBySsn(anyString())).thenReturn(emp);          Assert.assertEquals(employeeService.isEmployeeSsnUnique(emp.getId(), emp.getSsn()), true);      }          public List<Employee> getEmployeeList(){          Employee e1 = new Employee();          e1.setId(1);          e1.setName("Axel");          e1.setJoiningDate(new LocalDate());          e1.setSalary(new BigDecimal(10000));          e1.setSsn("XXX111");            Employee e2 = new Employee();          e2.setId(2);          e2.setName("Jeremy");          e2.setJoiningDate(new LocalDate());          e2.setSalary(new BigDecimal(20000));          e2.setSsn("XXX222");            employees.add(e1);          employees.add(e2);          return employees;      }    } |

Run above test class using [TestNG Eclipse Plugin](http://testng.org/doc/eclipse.html) or Maven **[mvn clean test e.g.]**

|  |
| --- |
| PASSED: deleteEmployeeBySsn  PASSED: findAllEmployees  PASSED: findById  PASSED: findEmployeeBySsn  PASSED: isEmployeeSsnUnique  PASSED: saveEmployee  PASSED: updateEmployee    ===============================================      Default test      Tests run: 7, Failures: 0, Skips: 0  =============================================== |

**Explanation of EmployeeServiceImplTest class:**

If you revisit **EmployeeServiceImpl** class in [Previous post](http://websystique.com/springmvc/spring-4-mvc-and-hibernate4-integration-example-using-annotations/), you will see that EmployeeServiceImpl basically depends on EmployeeDao & Employee to fulfill all of it’s duties. Each of the EmployeeServiceImplmethod is using only of these objects to do it’s real job.

As explained above, in order to test EmployeeServiceImpl, we would need to provide these dependencies. In our example, we do it using Mockito framework. We provide **mock** of EmployeeDao by applying @Mock annotation on it. We also provide **spy** objects of Employee by applying @Spy annotations on them.

@InjectMocks creates an instance of the class and injects the mocks that are created with the @Mock/@Spy objects in it.

MockitoAnnotations.initMocks(this); initializes objects annotated with Mockito annotations [@Mock, @Spy, @Captor, @InjectMocks]

So, Make sure to call **MockitoAnnotations.initMocks** when using Mockito annotations, else those mocks will be useless for your tests.

Please refer to [TestNG tutorial](http://websystique.com/java/testng-tutorial/) for in depth details about writing tests using TestNG, mockito and other supportive libraries.

**3) Testing Data Layer:**

Testing DAO or data layer is always a subject of debate. What exactly we want to test? Are we just testing the methods from DAO implementation class and making sure that each and every line of code in those methods is covered?

If we think in terms of unit-test, than our goal becomes testing every line of DAO code while really mocking all the external systems/dependencies. IMO, we can’t truly test a data-layer without really interacting with the database itself. And then it becomes an integration test.

Anyway, we will perform **integration-test** on our DAO layer to make sure that it works as expected. We will be using in-memory H2 database to do our integration-tests.

If we look back at actual dao/data-layer class **EmployeeDaoImpl** in [Previous post](http://websystique.com/springmvc/spring-4-mvc-and-hibernate4-integration-example-using-annotations/), it relies on hibernate for database interactions. There all hibernate setup related stuff was defined in **HibernateConfiguration** class. We will need similar setup in tests in order to connect to database and perform hibernate session related operations during our tests.

Below class is a setup class [with Annotations] for all hibernate configuration related activities during tests.

|  |
| --- |
| package com.javalive.springmvc.configuration;    import java.util.Properties;    import javax.sql.DataSource;    import org.hibernate.SessionFactory;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.core.env.Environment;  import org.springframework.jdbc.datasource.DriverManagerDataSource;  import org.springframework.orm.hibernate4.HibernateTransactionManager;  import org.springframework.orm.hibernate4.LocalSessionFactoryBean;  import org.springframework.transaction.annotation.EnableTransactionManagement;    /\*   \* This class is same as real HibernateConfiguration class in sources.   \* Only difference is that method dataSource & hibernateProperties   \* implementations are specific to Hibernate working with H2 database.   \*/    @Configuration  @EnableTransactionManagement  @ComponentScan({ "com.javalive.springmvc.dao" })  public class HibernateTestConfiguration {        @Autowired      private Environment environment;        @Bean      public LocalSessionFactoryBean sessionFactory() {          LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();          sessionFactory.setDataSource(dataSource());          sessionFactory.setPackagesToScan(new String[] { "com.javalive.springmvc.model" });          sessionFactory.setHibernateProperties(hibernateProperties());          return sessionFactory;      }        @Bean(name = "dataSource")      public DataSource dataSource() {          DriverManagerDataSource dataSource = new DriverManagerDataSource();          dataSource.setDriverClassName("org.h2.Driver");          dataSource.setUrl("jdbc:h2:mem:test;DB\_CLOSE\_DELAY=-1;DB\_CLOSE\_ON\_EXIT=FALSE");          dataSource.setUsername("sa");          dataSource.setPassword("");          return dataSource;      }        private Properties hibernateProperties() {          Properties properties = new Properties();          properties.put("hibernate.dialect", "org.hibernate.dialect.H2Dialect");          properties.put("hibernate.hbm2ddl.auto", "create-drop");          return properties;      }        @Bean      @Autowired      public HibernateTransactionManager transactionManager(SessionFactory s) {          HibernateTransactionManager txManager = new HibernateTransactionManager();          txManager.setSessionFactory(s);          return txManager;      }  } |

Above class is exactly same as HibernateConfiguration class defined in src in [Previous post](http://websystique.com/springmvc/spring-4-mvc-and-hibernate4-integration-example-using-annotations/). Only difference is that mathods **dataSource() & hibernateProperties()** are implemented specific to Hibernate & H2 combination. I choose to make a separate test configuration class and not to pollute existing class in Sources with testing related stuff.

It does everything exactly same as the one in Sources folder: it creates a SessionFacoty using a dataSource which is configured to work with in-memory database H2. In order to make hibernate work with H2, we also need to specify the dialect being used [H2 Dialect].

This SessionFactory will be injected in our **AbstractDao** class defined in [Previous post](http://websystique.com/springmvc/spring-4-mvc-and-hibernate4-integration-example-using-annotations/). And from then on, the actual DAO implementation classes [ EmployeeDaoImpl] will use this sessionFactory when running tests against them.

Additionally, we will be using DBUnit to clean-insert sample data in test database[H2] before each test case execution, in order to prepare database before each Dao method execution. This way we make sure that the tests method do not interfere with each other.

Below is a sample class which will act as a base class for all our test classes.

|  |
| --- |
| package com.javalive.springmvc.dao;    import javax.sql.DataSource;    import org.dbunit.database.DatabaseDataSourceConnection;  import org.dbunit.database.IDatabaseConnection;  import org.dbunit.dataset.IDataSet;  import org.dbunit.operation.DatabaseOperation;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.test.context.ContextConfiguration;  import org.springframework.test.context.testng.AbstractTransactionalTestNGSpringContextTests;  import org.testng.annotations.BeforeMethod;    import com.javalive.springmvc.configuration.HibernateTestConfiguration;      @ContextConfiguration(classes = { HibernateTestConfiguration.class })  public abstract class EntityDaoImplTest extends AbstractTransactionalTestNGSpringContextTests {        @Autowired      DataSource dataSource;        @BeforeMethod      public void setUp() throws Exception {          IDatabaseConnection dbConn = new DatabaseDataSourceConnection(                  dataSource);          DatabaseOperation.CLEAN\_INSERT.execute(dbConn, getDataSet());      }        protected abstract IDataSet getDataSet() throws Exception;    } |

AbstractTransactionalTestNGSpringContextTests can (at some extent) be considered as JUnit equivalent of RunWith. This abstract class integrates Spring TestContext support in TestNG environment. It requires a class-level @ContextConfiguration in order to load ApplicationContext using XML configuration files or annotated @Configuration classes.

It also requires a datasource and a transactionManager to be defined in ApplicationContext in order to provide data-access support during testing. We have already defined both datasource & transactionManager in our @Configuration class.

Thanks to transaction support, by default a transaction will be started before each test, and then this transaction will be rolled back at the end of test. You may override the rollback behavior.

Look at setup method annotated with @BeforeMethod. Method annotated with @BeforeMethod is called before each test, so it is an ideal place to do something which is required before each test. In our case , we want the in-memory database to be clean and predefined sample data to be inserted before each test. We will do it right here.

Additionally, for DBUnit to connect to database in order to perform clean-insert, we have to provide a dataSource for it. That’s why we declared a dataSource here, which will be autowired with dataSource defined in HibernateTestConfiguration class.

As shown in above setUp method, firstly we create a connection to database using dataSource available( which will be test dataSource), and execute clean-insert on DB via DBUnit.

Notice the abstract method **getDataSet** above. This method will be implemented in our tests classes in order to provide the actual test data to be inserted before each test.

Finally, the actual test class which tests methods from our DAO implementation class.

|  |
| --- |
| package com.javalive.springmvc.dao;    import java.math.BigDecimal;    import org.dbunit.dataset.IDataSet;  import org.dbunit.dataset.xml.FlatXmlDataSet;  import org.joda.time.LocalDate;  import org.springframework.beans.factory.annotation.Autowired;  import org.testng.Assert;  import org.testng.annotations.Test;    import com.javalive.springmvc.model.Employee;      public class EmployeeDaoImplTest extends EntityDaoImplTest{        @Autowired      EmployeeDao employeeDao;        @Override      protected IDataSet getDataSet() throws Exception{          IDataSet dataSet = new FlatXmlDataSet(this.getClass().getClassLoader().getResourceAsStream("Employee.xml"));          return dataSet;      }        /\* In case you need multiple datasets (mapping different tables) and you do prefer to keep them in separate XML's      @Override      protected IDataSet getDataSet() throws Exception {        IDataSet[] datasets = new IDataSet[] {                new FlatXmlDataSet(this.getClass().getClassLoader().getResourceAsStream("Employee.xml")),                new FlatXmlDataSet(this.getClass().getClassLoader().getResourceAsStream("Benefits.xml")),                new FlatXmlDataSet(this.getClass().getClassLoader().getResourceAsStream("Departements.xml"))        };        return new CompositeDataSet(datasets);      }      \*/        @Test      public void findById(){          Assert.assertNotNull(employeeDao.findById(1));          Assert.assertNull(employeeDao.findById(3));      }          @Test      public void saveEmployee(){          employeeDao.saveEmployee(getSampleEmployee());          Assert.assertEquals(employeeDao.findAllEmployees().size(), 3);      }        @Test      public void deleteEmployeeBySsn(){          employeeDao.deleteEmployeeBySsn("11111");          Assert.assertEquals(employeeDao.findAllEmployees().size(), 1);      }        @Test      public void deleteEmployeeByInvalidSsn(){          employeeDao.deleteEmployeeBySsn("23423");          Assert.assertEquals(employeeDao.findAllEmployees().size(), 2);      }        @Test      public void findAllEmployees(){          Assert.assertEquals(employeeDao.findAllEmployees().size(), 2);      }        @Test      public void findEmployeeBySsn(){          Assert.assertNotNull(employeeDao.findEmployeeBySsn("11111"));          Assert.assertNull(employeeDao.findEmployeeBySsn("14545"));      }        public Employee getSampleEmployee(){          Employee employee = new Employee();          employee.setName("Karen");          employee.setSsn("12345");          employee.setSalary(new BigDecimal(10980));          employee.setJoiningDate(new LocalDate());          return employee;      }    } |

Below is the content of input file used by DBUnit:

src/test/resources/Employee.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <dataset>      <employee id="1" NAME="SAMY" JOINING\_DATE="2014-04-16"       SALARY="20000"  SSN="11111" />      <employee id="2" NAME="TOMY" JOINING\_DATE="2014-05-17"       SALARY="23000"  SSN="11112" />  </dataset> |

Run above test class using [TestNG Eclipse Plugin](http://testng.org/doc/eclipse.html) or Maven **[mvn clean test e.g.]**

|  |
| --- |
| PASSED: deleteEmployeeByInvalidSsn  PASSED: deleteEmployeeBySsn  PASSED: findAllEmployees  PASSED: findById  PASSED: findEmployeeBySsn  PASSED: saveEmployee    ===============================================      Default test      Tests run: 6, Failures: 0, Skips: 0  =============================================== |

Let’s take **saveEmployee** test case and understand how things happening here.

1) Before any of the test from the classes (which are extending **EntityDaoImplTest**) starts executing, Spring will load the text context from the configuration classes associated with **@ContextConfiguration** annotation & create the beans instances defined in those classes, thanks to **AbstractTransactionalTestNGSpringContextTests**. This will happen only once.

2) During @Bean instance creation, Spring will create the **SessionFactory** Bean which will be injected with dataSource bean (as defined in HibernateTestConfiguration class) based on database & hibernate properties. Look at the following property

|  |
| --- |
| properties.put("hibernate.hbm2ddl.auto", "create-drop"); |

Thanks to this **hbm2ddl** property, when the SessionFactory is created, the schema related to our Model classes will be **validated and exported to database**. That means Employee table will be **created in H2 database**.

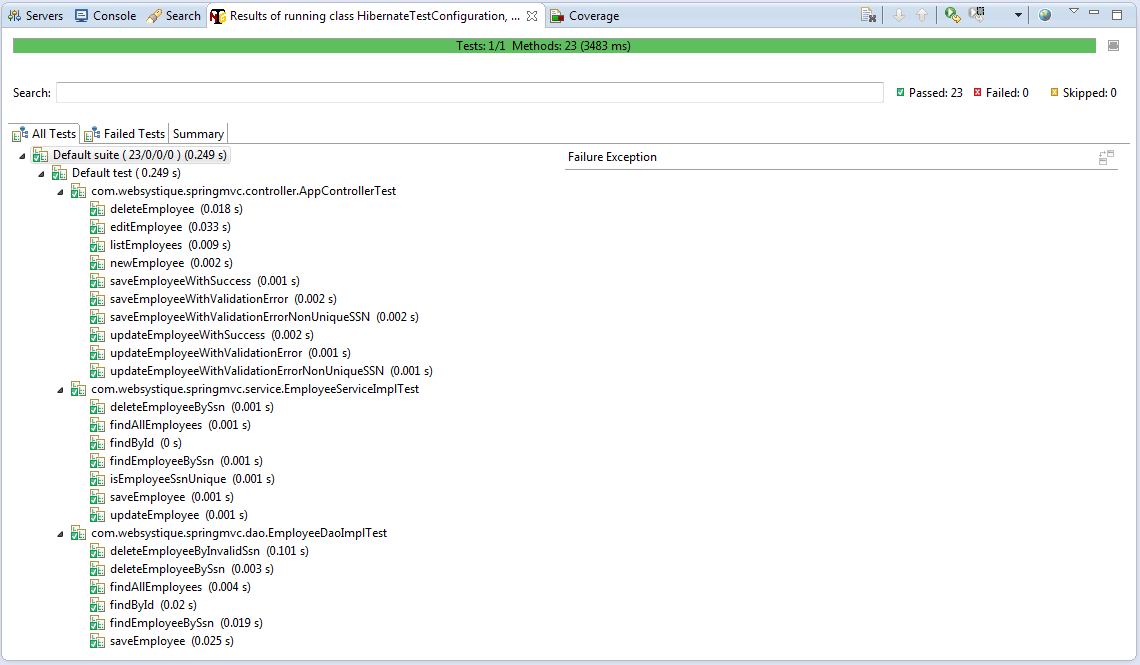
3) Before the test start, **@BeforeMethod** will be called, which will instruct **DBUnit** to connect to database and perform clean-insert.It will insert 2 rows in Employee table (look at Employee.xml content)

4) Now the actual test case **saveEmployee** is about to start. Just before execution start, a **transaction** will be started. Method itself will run within this transaction. Once method finished it’s execution, transaction will be rolled back which is default setup. You can override this behavior by annotating the test method with @Rollback(true) annotation. It is defined in [org.springframework.test.annotation.Rollback]

5) Now the actual test case **saveEmployee** finally starts it’s execution. It will call **employeeDao.saveEmployee(getSampleEmployee());** which in-turn will insert the one pre-defined Employee object into H2 database using hibernate. This is the core logic of saveEmployee method anyway. After this operation there will be total 3 rows in Employee table in H2 database. We will assert it for success/failure. Test completes.

6) For next test case, again @BeforeMethod will be called which will delete everything from table and re-insert two rows as defined in Employee.xml. Story continues…

7) When all our tests are done, session will be closed and schema will be dropped.



Shown below is the output of running **all the tests** using maven [**mvn clean test**]

|  |
| --- |
| E:\workspace7\SpringHibernateExample>mvn clean test  [INFO] Scanning for projects...  [INFO]  [INFO] ------------------------------------------------------------------------  [INFO] Building SpringHibernateExample 1.0.0  [INFO] ------------------------------------------------------------------------  [WARNING] The artifact dbunit:dbunit:jar:2.2 has been relocated to org.dbunit:dbunit:jar:2.2  [INFO]  [INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ SpringHibernateExample ---  [INFO] Deleting E:\workspace7\SpringHibernateExample\target  [INFO]  [INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ SpringHibernateExample ---  [WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!  [INFO] Copying 2 resources  [INFO]  [INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ SpringHibernateExample ---  [WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!  [INFO] Compiling 10 source files to E:\workspace7\SpringHibernateExample\target\classes  [INFO]  [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ SpringHibernateExample ---  [WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!  [INFO] Copying 1 resource  [INFO]  [INFO] --- maven-compiler-plugin:2.5.1:testCompile (default-testCompile) @ SpringHibernateExample ---  [WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!  [INFO] Compiling 5 source files to E:\workspace7\SpringHibernateExample\target\test-classes  [INFO]  [INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ SpringHibernateExample ---  [INFO] Surefire report directory: E:\workspace7\SpringHibernateExample\target\surefire-reports    -------------------------------------------------------   T E S T S  -------------------------------------------------------  Running TestSuite  Configuring TestNG with: org.apache.maven.surefire.testng.conf.TestNG652Configurator@556e7212  Jul 12, 2020 1:46:48 AM org.springframework.context.support.GenericApplicationContext prepareRefresh  INFO: Refreshing org.springframework.context.support.GenericApplicationContext@8be1456: startup date [Sun Jul 12 01:46:48 CEST 2020]; root of context hierarchy  Jul 12, 2020 1:46:48 AM org.springframework.jdbc.datasource.DriverManagerDataSource setDriverClassName  INFO: Loaded JDBC driver: org.h2.Driver  Jul 12, 2020 1:46:48 AM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>  INFO: HCANN000001: Hibernate Commons Annotations {4.0.5.Final}  Jul 12, 2020 1:46:48 AM org.hibernate.Version logVersion  INFO: HHH000412: Hibernate Core {4.3.6.Final}  Jul 12, 2020 1:46:48 AM org.hibernate.cfg.Environment <clinit>  INFO: HHH000206: hibernate.properties not found  Jul 12, 2020 1:46:48 AM org.hibernate.cfg.Environment buildBytecodeProvider  INFO: HHH000021: Bytecode provider name : javassist  Jul 12, 2020 1:46:49 AM org.hibernate.dialect.Dialect <init>  INFO: HHH000400: Using dialect: org.hibernate.dialect.H2Dialect  Jul 12, 2020 1:46:49 AM org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService  INFO: HHH000399: Using default transaction strategy (direct JDBC transactions)  Jul 12, 2020 1:46:49 AM org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>  INFO: HHH000397: Using ASTQueryTranslatorFactory  Jul 12, 2020 1:46:49 AM org.hibernate.validator.internal.util.Version <clinit>  INFO: HV000001: Hibernate Validator 5.1.3.Final  Jul 12, 2020 1:46:49 AM org.hibernate.tool.hbm2ddl.SchemaExport execute  INFO: HHH000227: Running hbm2ddl schema export  Jul 12, 2020 1:46:49 AM org.hibernate.tool.hbm2ddl.SchemaExport execute  INFO: HHH000230: Schema export complete  Jul 12, 2020 1:46:49 AM org.springframework.orm.hibernate4.HibernateTransactionManager afterPropertiesSet  INFO: Using DataSource [org.springframework.jdbc.datasource.DriverManagerDataSource@3e2798e6] of Hibernate SessionFactory for HibernateTransactionManager  Jul 12, 2020 1:46:49 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener startNewTransaction  INFO: Began transaction (1) for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive.springmvc.dao.Employee  DaoImplTest@4fc55da3, testMethod = deleteEmployeeByInvalidSsn@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedContextConfigurat  ion@49dc008c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguration}', contextI  nitializerClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent = [null]]]; tra  nsaction manager [org.springframework.orm.hibernate4.HibernateTransactionManager@aa80d36]; rollback [true]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener endTransaction  INFO: Rolled back transaction after test execution for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive  .springmvc.dao.EmployeeDaoImplTest@4fc55da3, testMethod = deleteEmployeeByInvalidSsn@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [  MergedContextConfiguration@49dc008c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestCo  nfiguration}', contextInitializerClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader',   parent = [null]]]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener startNewTransaction  INFO: Began transaction (2) for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive.springmvc.dao.Employee  DaoImplTest@4fc55da3, testMethod = deleteEmployeeBySsn@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedContextConfiguration@49d  c008c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguration}', contextInitiali  zerClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent = [null]]]; transactio  n manager [org.springframework.orm.hibernate4.HibernateTransactionManager@aa80d36]; rollback [true]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener endTransaction  INFO: Rolled back transaction after test execution for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive  .springmvc.dao.EmployeeDaoImplTest@4fc55da3, testMethod = deleteEmployeeBySsn@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedC  ontextConfiguration@49dc008c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfigura  tion}', contextInitializerClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent   = [null]]]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener startNewTransaction  INFO: Began transaction (3) for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive.springmvc.dao.Employee  DaoImplTest@4fc55da3, testMethod = findAllEmployees@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedContextConfiguration@49dc00  8c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguration}', contextInitializer  Classes = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent = [null]]]; transaction m  anager [org.springframework.orm.hibernate4.HibernateTransactionManager@aa80d36]; rollback [true]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener endTransaction  INFO: Rolled back transaction after test execution for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive  .springmvc.dao.EmployeeDaoImplTest@4fc55da3, testMethod = findAllEmployees@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedCont  extConfiguration@49dc008c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguratio  n}', contextInitializerClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent =  [null]]]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener startNewTransaction  INFO: Began transaction (4) for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive.springmvc.dao.Employee  DaoImplTest@4fc55da3, testMethod = findById@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedContextConfiguration@49dc008c testC  lass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguration}', contextInitializerClasses  = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent = [null]]]; transaction manager [  org.springframework.orm.hibernate4.HibernateTransactionManager@aa80d36]; rollback [true]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener endTransaction  INFO: Rolled back transaction after test execution for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive  .springmvc.dao.EmployeeDaoImplTest@4fc55da3, testMethod = findById@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedContextConfi  guration@49dc008c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguration}', con  textInitializerClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent = [null]]]    Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener startNewTransaction  INFO: Began transaction (5) for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive.springmvc.dao.Employee  DaoImplTest@4fc55da3, testMethod = findEmployeeBySsn@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedContextConfiguration@49dc0  08c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguration}', contextInitialize  rClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent = [null]]]; transaction  manager [org.springframework.orm.hibernate4.HibernateTransactionManager@aa80d36]; rollback [true]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener endTransaction  INFO: Rolled back transaction after test execution for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive  .springmvc.dao.EmployeeDaoImplTest@4fc55da3, testMethod = findEmployeeBySsn@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedCon  textConfiguration@49dc008c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfigurati  on}', contextInitializerClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent =   [null]]]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener startNewTransaction  INFO: Began transaction (6) for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive.springmvc.dao.Employee  DaoImplTest@4fc55da3, testMethod = saveEmployee@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedContextConfiguration@49dc008c t  estClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguration}', contextInitializerClas  ses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent = [null]]]; transaction manag  er [org.springframework.orm.hibernate4.HibernateTransactionManager@aa80d36]; rollback [true]  Jul 12, 2020 1:46:50 AM org.springframework.test.context.transaction.TransactionalTestExecutionListener endTransaction  INFO: Rolled back transaction after test execution for test context [DefaultTestContext@50dcb4b5 testClass = EmployeeDaoImplTest, testInstance = com.javalive  .springmvc.dao.EmployeeDaoImplTest@4fc55da3, testMethod = saveEmployee@EmployeeDaoImplTest, testException = [null], mergedContextConfiguration = [MergedContextC  onfiguration@49dc008c testClass = EmployeeDaoImplTest, locations = '{}', classes = '{class com.javalive.springmvc.configuration.HibernateTestConfiguration}',   contextInitializerClasses = '[]', activeProfiles = '{}', contextLoader = 'org.springframework.test.context.support.DelegatingSmartContextLoader', parent = [nul  l]]]  Tests run: 23, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.982 sec    Results :    Tests run: 23, Failures: 0, Errors: 0, Skipped: 0    [INFO] ------------------------------------------------------------------------  [INFO] BUILD SUCCESS  [INFO] ------------------------------------------------------------------------  [INFO] Total time: 9.481s  [INFO] Finished at: Sun Jul 12 01:46:50 CEST 2020  [INFO] Final Memory: 17M/224M  [INFO] ------------------------------------------------------------------------ |

That’s it.