Training: Spring Framework

Workbook
(edition 2.2, March, 2012)

Training: Spring Framework

The authors:	Evgeniy Krivosheev (EKrivosheev@luxoft.com) Dariya Linnik (DLinnik@luxoft.com) Maxim Potanin (MPotanin@luxoft.com) Vyacheslav Yakovenko (VYakovenko@luxoft.com)
Product version:	Spring 3.x

Exercise Specifications

Virtual Machine Properties

IMAGE_SOURCE_LOCATION	spring31.ova
VM_NAME	jva-spring
GUEST_OS_TYPE	Linux/Ubuntu
GUEST_OS_RAM	>1Gb
GUEST_LOGIN	user
GUEST_PASS	user

Exercise Properties for the Virtual Machine

ECLIPSE_HOME	~/bin/eclipse
LABS_WORKSPACE	~/Documents/labs
SOLUTIONS_WORKSPACE	~/Documents/solutions
TOMCAT_HOME	~/bin/tomcat
SPRING_HOME	~/bin/spring-framework-2.5.5
SPRING3_HOME	~/bin/spring-framework-3.1.1.RELEASE
SPRING3_HOME_SLINK	~/bin/spring
HSQL_HOME	~/bin/hsqldb/lib/hsqldb.jar

Exercise 1
Installing and adjusting runtime environment

Duration

30 min

Objectives

Install and adjust runtime environment.

Description

The runtime implementation is based on *virtual machine* concept. Virtual machine is an emulation of comprehensive PC (machine) within the framework of preinstalled OS. The most popular software products that implement above-mentioned virtual machines are Microsoft VirtualPC, VMWare (Server and Player) and Oracle Virtual Box. Oracle Virtual Box is used in the present runtime.

The virtual machine as well as a usual one shall be equipped with OS. The operating system of physical computer on which the virtual machine was installed is called *Host operating system.* The operating system running inside the virtual machine is called *Guest operating system.* Multiple guest systems can be run on a host system. Their number is only limited by the resource of the host system.

The virtual machine, as well as usual one, uses HDD for storing data and installed software. However, virtual machine's HDD serves as an *image* and the host system treats it as a single file. For Oracle VirtualBox this is the file with `ova (vdi)` extension.

The image can be set up by installing OS, necessary software and by copying data required. After that, HDD image is copied to the required host systems and virtual machines are configured to use this image.

In this course the runtime is implemented as HDD image for Oracle VirtualBox. To install the runtime, start Oracle VirtualBox and create new virtual machine. The virtual machine is a set of parameters. One can create multiple virtual machines. These settings include machine name, allocated memory and HDD image location.

After setting up new virtual machine, it can be started. Its window will be represented as the usual window of the host system. Switching to full-screen mode and vice versa can be done by pressing key combination `RightCtrl+F.`

Minimizing the virtual machine window will not turn it off. When the virtual machine is not in use, press `RightCtrl+P,` to save processor time.

To stop the virtual machine one of the following can be done:

- Send shutdown signal to guest operating system (the machine will shutdown properly);
- Tough shutdown (emulating power off);
- Shutdown with saving the machine state (at the next start the state will be the same as it was left).

Tasks

1. Copy virtual disk image to local machine, if required *.
2. If the file is archived, unpack it*.
3. Configure new virtual machine using the virtual disk.
4. Launch the virtual machine.
5. Freeze it.

Detailed guidelines

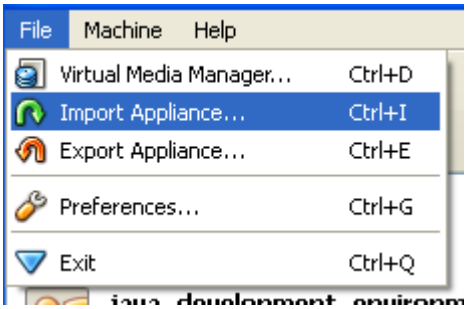
Part 1: Importing virtual machine

Necessary software:

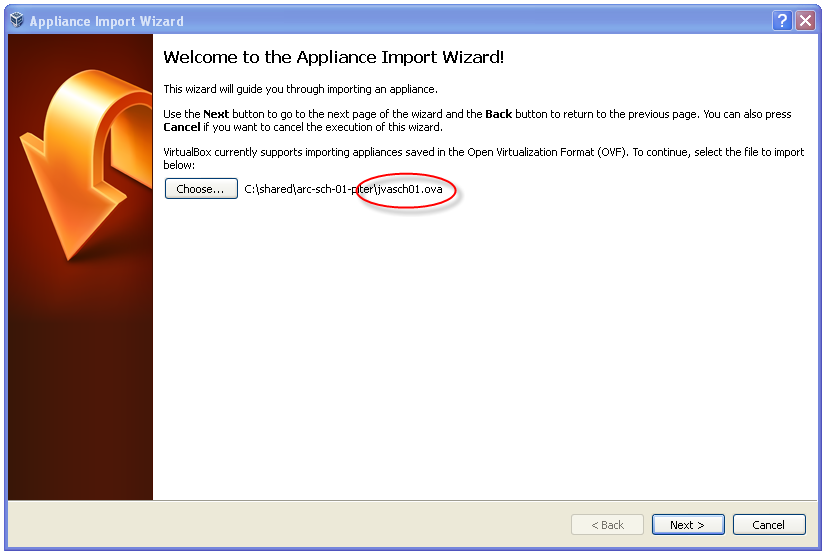
Oracle VirtualBox 4.1.* or higher (*www.virtualbox.org*)

Installation:

1. Launch VirtualBox
2. On menu {File}, choose {Import Appliance…};



3. By clicking [Choose…], select relevant image and click Next>];



4. Specify virtual machine name that will be used by VirtualBox manager (by default, jva-spring) and click [Import] button;

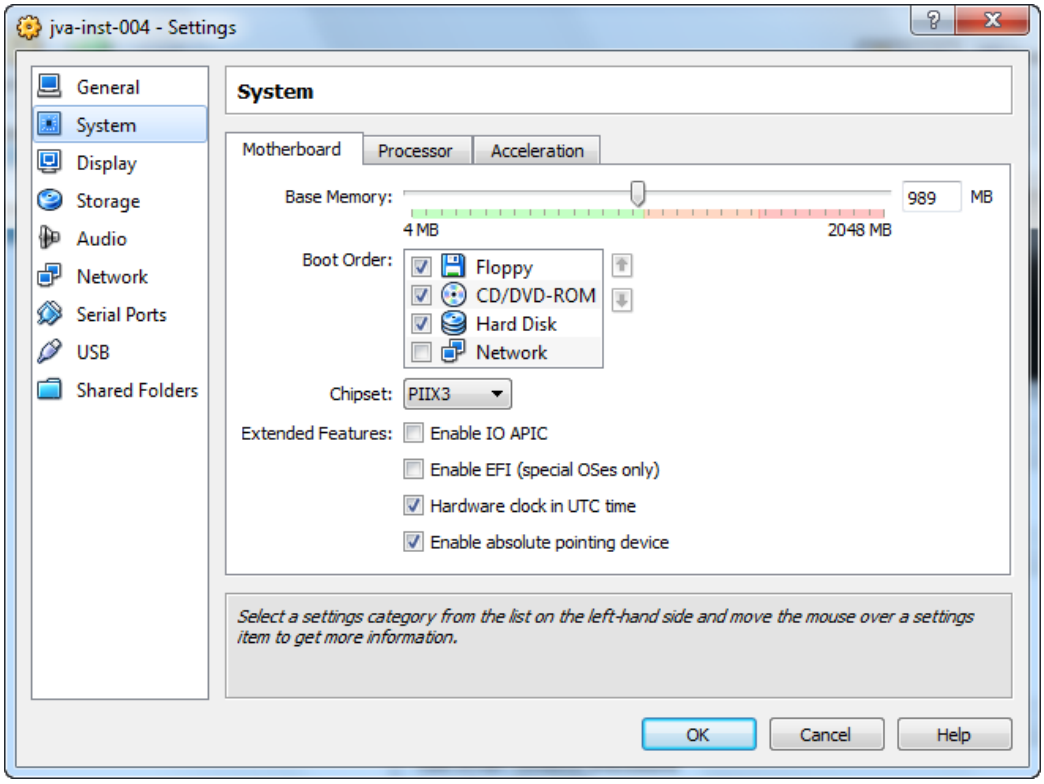
   Launch virtual machine and verify its operability.

---

* Optional

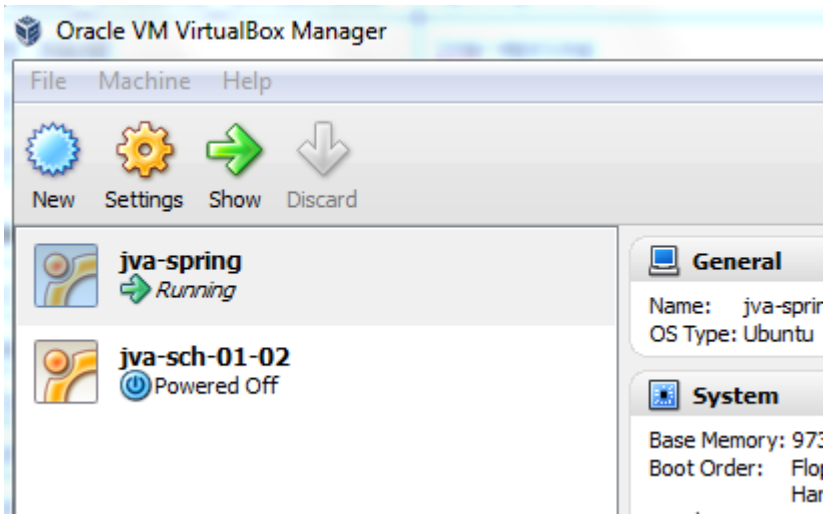
Part 2: Configuring virtual machine parameters

Enter System section (without launching virtual machine) and check that scroll box Base Memory is at the maximum of green area. It is desirable that you could allocate not less than 1GB for guest operating system. Click [OK].



Part 3: Launching virtual machine

1. Launch Oracle VM VirtualBox Manager. In the left part of the window select required virtual machine and click [Start]. Wait until your virtual machine is loaded.



Exercise 2
Adjusting auxiliary services

Duration

20 min

Objectives

Get acquainted with auxiliary development tools.

Description

Eclipse is used as a development environment. Practical exercises are in LABS_WORKSPACE. Solutions are in SOLUTIONS_WORKSPACE.

`HSQL` is used as DBMS. To demonstrate Spring-based data handling, an existing database is needed. All actions will be done on it.

We will use Apache Tomcat as a container.

Tasks

1. Get acquainted with Eclipse development environment.
2. Get acquainted with `HSQL` DBMS.

Detailed guidelines

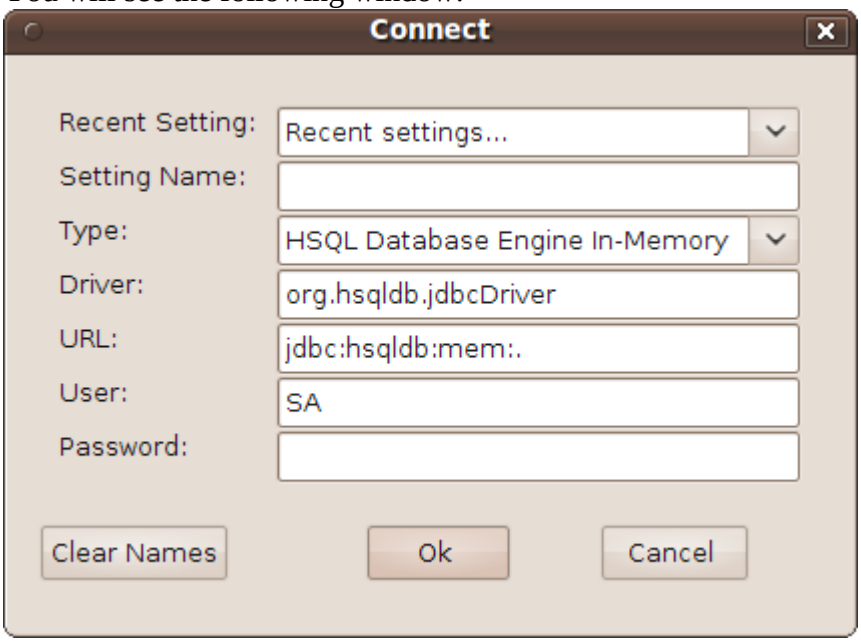
Part 1: Eclipse development environment

1. Launch `ECLIPSE_HOME/eclipse`
2. Study template projects and solutions by switching the workspace (File -> Switch Workspace).
3. Check switching between Java EE and Java perspectives (Window -> Open Perspective).

Part 2: HSQL DBMS.

4. Launch text client java -cp ~/bin/hsqldb/lib/hsqldb.jar org.hsqldb.util.DatabaseManagerSwing

5. You will see the following window:



Please remember these parameters, because you will need them in future to configure DataSource. Pay attention to url:jdbc:hsqldb:mem as well. The url indicates that a temporary [.] database will be created in RAM. It will be deleted after ending the work; this is especially useful during the testing.

Exercise 3
HelloWorld Application

Project

`lab-3-core`

Duration

Practice: 20-25 min

Discussion: 5 min

Objectives

Find out more about Spring IoC using the simplest application as an example. Understand basics of dependency injection and creation of application context.

Get acquainted with an application, core entities and relationship between them.

Description

Spring allows for declarative dependency injection between classes, and creation of common application context. Spring allows declaring usual java classes as beans (POJO).

Tasks

1. Understand basics of application context definition in xml file.
2. Understand dependency injection and population of bean fields' values.
3. Analyze how application context is created from xml file.
4. Analyze the existing entities and their relationship.

Detailed guidelines

Part 1: Basics of context definition in xml file.

1. Open the project `lab-3-core;`
2. Open the file `{lab-3-core}/resources/application-context.xml`.
3. Analyze xml file structure.

Part 2: Dependency Injection. Populating the field values

4. Pay attention to definition of `person` and `country` beans.
5. *How fields of the* `person` *bean are populated?* Compare bean definition and the `Person` class.
6. *How the* `person` *and* `country` *beans are related?*
7. Open the `HelloWorldTest` test. Compare retrieving reference object (`getExpectedPerson` () method) and its definition in xml file.

Part 3: Retrieving the context from xml file.

8. Open the `HelloWorldTest` class, the `setUp()` method.
9. *How the context is retrieved from xml file? Where should this file be located?*
10. Open the `HelloWorldTest` test and launch it using jUnit 4 testing framework*.

Part 4: Analyzing the entities.

11. Open `package model`. *Which classes and interfaces are defined there? How are they related?*
12. Study other `packages` and applications. *What are the purposes of various entities?*

Part 5: Analyzing libraries connected to the project.

13. Enter the lib directory. *Which libraries are located there?*
14. *What products are they related to?*
15. *Why not all libraries from SPRING3_HOME/dist were included in this project?*

Exercise 4
Developing primitive application

Project

`lab-4-core`

Duration

30 min

Objectives

- Learn to develop primitive applications using Spring.
- Learn how to use basics of dependency injection and bean wiring using Spring IoC container.
- Learn to define and create application context.
- Learn to use autowiring.

Description

Spring allows for declarative dependency injection between classes, and creation of common application context. Spring allows declaring usual java classes as beans (POJO). It is possible to automatically wire beans by type or name.

Tasks

1. Define and create application context for existing model classes `UsualPerson`, `Country and Contact.`
2. Specify all fields' values for all classes, so that the `SimpleAppTest` test would be executed properly.
3. Define all necessary dependencies by any means in accordance with the `SimpleAppTest` test.
4. Create application context.
5. Implement the situation: a man lives in a particular country. He has several contacts. Specify all initial values for all parameters and render all information about this person. Use the `SimpleAppTest` test for checking.

Detailed guidelines

Part 1: Creating application context

1. Open the project `lab-4-core;`
2. Open the file `{lab-4-core}/resources/application-context.xml`. It already contains initial definition of the `country` and `person` beans.
3. Open the `SimpleAppTest` test, the `getExpectedPerson()` method. *Which definitions are missing in existing xml file to comply with reference class?*

Part 2: Specifying fields' values

4. Specify the `height` and `isProgrammer` fields for the `UsualPerson` class.
5. This is done in the `<property name=… value=… >` tag inside the `<bean>` tag. Simple `property (String, int, Boolean, etc` are specified by values `(value=)`, their name are defined by `name=` attribute.

```
<property name="height" value="1.78"/>
<property name="isProgrammer" value="true"/>
```

6. Specify the `Contacts` contact list. For that in the `property` tag, along with `name=contacts`, the `list` tag is created. Inside this tag you create `value` tags and define one contact for each tag.

```
<property name="contacts">
    <list>
        <value>asd@asd.ru</value>
        <value>+7-234-456-67-89</value>
    </list>
</property>
```

Part 3: Dependency Injection

7. Along with simple `property` it is necessary to specify references to other classes. *What are the ways of doing so?*
8. One way to wire beans is to explicitly specify a reference to necessary bean `<property name=… ref=…>`. *What are the advantages and disadvantages of the approach?*
9. Another way to wire them is autowire (`autowire`, do not forget to add `<context:annotation-config/>`) either by type or by name. *What are the advantages and disadvantages of the approach?*

Part 4: Creating application context

10. Open the `@Before setUp()` method.
11. There is a variable like `AbstractApplicationContext`. The `ClassPathXmlApplicationContext` value is specified for it. Its parameter is a disk path to file with context definition.
12. What is the difference between this declaration and one from previous example?

Part 4: Implementing test situation

13. Open the `SimpleAppTest` test, the `testInitPerson()` method.
14. Retrieve a bean responsible for the `UsualPerson` class from the context `context.getBean(bean name)`. Not that this method will return object of `Object` type. It shall be rendered to the `UsualPerson` type.

15. Pay attention to alternative features of bean retrieval from the context. For that, enter the documentation for AbstractApplicationContext class: `{SPRING3-HOME}/docs/javadoc-api/index.html` and study its API.
16. Invoke the `toString()` method for object from just-retrieved bean.
17. Execute test. If everything is correct, the test will be executed without errors and information about the man will be rendered on console.

Part 5: Using Spring TestContext Framework

18. Open the `SpringTCFAppTest` class;
19. Compare it with previous test (`SimpleAppTest`). *What is the cardinal difference between these classes?*
20. *What library the `@RunWith` annotation belongs to?*
21. Examine documentation for the `@ContextConfiguration` annotation (see 10.3.5.2 Context management, Spring Reference v.3.1.1). *What other variants and application context definitions does it support?*

Exercise 5
Using Spring AOP. @AspectJ style

Project

`lab-5-aop`

Duration

40 min

Objectives

Learn how to use Spring AOP with @AspectJ style approach through the example of Before, After, and Around advices.

Description

There are two basic approaches in using AOP in Spring: @AspectJ style and Schema-based. The @AspectJ approach can be used beginning with JDK 5. When using this approach, all definitions are made through annotations, i.e. are embedded into the code. In such cases one cannot use classes from external libraries as advices or pointcuts.

Tasks

1. Through the example of `Before` advice and `After returning advice (Politeness class)`, create other `After advices` types for the `sellSquishee(…)` method from the `ApuBar` class using the `@AspectJ` approach. Verify whether it runs correctly or not using the `AopAspectJTest` test.
2. Create `Around advice` for the `sellSquishee(…)` method using the `@AspectJ` approach. Verify whether it runs correctly or not using the `AopAspectJTest` test.

Detailed guidelines

Part 1: Creating `after advices`.

1. Open the project `lab-5-aop`;
2. Open the file with the `application-context.xml` configuration. Add the `<aop:aspectj-autoproxy/>` string. It indicates that all object definitions will be located in declarations.
3. Open the `Politeness` class. *What is the meaning of parameters in* `sayHello` *and* `askOpinion` *methods' annotations? How the name of the drink sold is retrieved in the* `askOpinion` *method?*
4. Create `advice` based on the `sayGoodBye` method that will be executed after an attempt to sell the drink anyway. (`After`). For that, add the `@After("execution(* sellSquishee(..))")` string before method declaration.

5. Launch the `AopAspectJTest` test for various `Customer` configurations (whether he has money or not: parameter `broke`). S*pecify the conditions for executing each of the* `testAfterReturningAdvice` *tests correctly.*

Part 2: Creating `around advices`.

6. Create `Around advice` for the `sellSquishee` method, so that it included the invocation of the `target` method. Take the `sayPoliteWordsAndSell` method as a reference. For that, add the `@Around("execution(* sellSquishee(..))")` string before method declaration.
7. Launch the `testAroundAdvice` test for check.
8. *Can you avoid invoking the* `target` *method in* `around advice?`

Part 3: Creating `after throwing advices`.

9. Create `advice` on the basis of the `sayYouAreNotAllowed` method that will be executed if the customer is broke that is when the exception `CustomerBrokenException (After throwing)` will be thrown. For that, add the `@AfterThrowing("execution(* sellSquishee(..))")` string before method declaration;
10. Open the `application-context.xml` and modify it so that the customer would be “broke”;
11. Open the `AopAspectJExceptionTest` class and execute the unit test;

Exercise 6
Using JDBC in Spring when handling data

Project

`lab-6-jdbc`

Duration

1 hour

Objectives

Learn how to use JDBC with Spring. Learn to perform basic JDBC operations (SELECT, INSERT, UPDATE) via Spring.

Description

Spring offers auxiliary API to work with JDBC.

To work with JDBC via Spring, it is necessary to configure DataSource. This is done through application-context.xml.

The HSQLDB is used as a database.

Objectives

1. Retrieve and print complete country list using `CountryDao`. *All fields of the* `Country` *class are filled correctly?* To check, use the `JdbcTest` test, the `testCountryList` method.
2. Correct the `CountryRowMapper` so that all the `Country` fields would be filled.
3. Retrieve and print complete list of countries whose names begin with A letter. To check, use the `JdbcTest` test, the `testCountryListStartsWithA` method.
4. Change the name of any country in the database. To check, use the `JdbcTest` test, the `testCountryChange` method (the reference country shall be changed in this test).

Detailed guidelines

Part 1: Retrieving the complete country list

1. Open the project `lab-6-jdbc;`
2. Open the directory `{lab-6-jdbc}/lib`. *Which libraries have been added to this project? Why?*
3. Open the file with the `application-countext.xml` configuration. Analyze configuration of every bean.
4. Open the `CountryDao` class. Implement the `getCountryList()` method so that it would return the complete country list. Use the existing methods as an example.

5. For that, retrieve `JdbcTemplate` and invoke the `query(String s, RowMapper rowMapper)` method.
6. As a `sql` query the `GET_ALL_COUNTRIES_SQL` can be used.
7. As a `RowMapper: COUNTRY_ROW_MAPPER`.
8. Open the `JdbcTest` test, execute the `testCountryList` method.
9. *Did the test run correctly? Why?*

Part 2: Correcting RowMapper

10. Open the `CountryRowMapper` class. *What has to be done in order to have all the* `Country` *fields filled?*
11. Specify `codeName` for `Country` in the `mapRow` method, similarly to `id` and `name`.
12. Again, execute the `JdbcTest` test, the `testCountryList` method.

Part 2: Retrieving list of countries whose name begin with A letter.

13. Open the `CountryDao` class, the `getCountryListStartWith()` method. *What is* `NamedParameterJdbcTemplate`*? How does it differ from* `JdbcTemplate`*?*
14. Open the `JdbcTest` test, the `testCountryListStartsWithA` method. *Test runs correctly?*
15. Now, try to annotate the `@DirtiesContext` annotation for some tests (mixing it up). *How tests run now? Why?*

Part 3: Changing the country name

16. Open the `CountryDao` class.
17. Implement the method for changing country names with known `codename: updateCountryName`. You can use `UPDATE_COUNTRY_NAME_SQL_1, UPDATE_COUNTRY_NAME_SQL_2` as a SQL query.

    You are going to have the following:

```
getJdbcTemplate().execute(
UPDATE_COUNTRY_NAME_SQL_1 + newCountryName + "'" +
UPDATE_COUNTRY_NAME_SQL_2 + codeName + "'");
```

18. To execute the `JdbcTest` test, the `testCountryChange` method, create a country with new name and known `codeName`.
19. How can you avoid invoking `countryDao.loadCountries();` in the `@Before` method by changing the application-context.xml and JdbcTest? What are the advantages of this approach? (Additional information needed for answering the question: p. 13.8 Embedded database support, Spring Reference);

Exercise 7
Using ORM in Spring when handling data

Project

`lab-7-orm`

Duration

45 min

Objectives

Learn how to use ORM via Spring through the example of JPA (Hibernate v.4).

Description

Spring supports various ORM using one approach. Using ORM via Spring facilitates testing, exceptions handling, and resource management. Spring supports following ORM: JPA, Hibernate, JDO, iBATIS SQL Maps, and others.

We will use Entity classes annotated through javax.persistence (JPA v.2.0: JSR-317; JPA v.2.1: JSR-338) in the example.

Tasks

1. Create and configure `LocalContainerEntityManagerFactoryBean` on the basis of existing `DataSource`. Specify `lab.model.Country` as `@Entity`.
2. Create `mapping` for the `Country` class.
3. Implement the country save method `save(Country country)` in the `CountryJpaDaoImpl` class. Save the country in the database. Check with the `CountryDaoImplTest` test, `testSaveCountry()`.
4. Implement the method of retrieving all countries, the `getAllCountries()` method, in the `CountryJpaDaoImpl` class. Retrieve the list of all countries. Check using the `CountryDaoImplTest` test, `testGetAllCountries ()`.
5. Implement the method of retrieving country by its name in the `CountryJpaDaoImpl` class. Retrieve a country by its name. Check using the `CountryDaoImplTest` test, `testGetCountryByName (String name)`.

Detailed guidelines

Part 1: Creating and configuring `LocalContainerEntityManagerFactoryBean`.

1. Open the file with the `application-context.xml` configuration.
2. Create the `lcemf` bean. Indicate LocalContainerEntityManagerFactoryBean.as the bean class.

3. Specify for the bean `dataSource, persistenceUnitName, persistenceProviderClass`.

   You are going to have following configuration:

```
<bean id="lcemf"          class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="loadTimeWeaver">
                <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver" />
        </property>
        <property name="dataSource" ref="dataSource"></property>
        <property name="persistenceUnitName" value="springframework.lab.orm.jpa" />
        <property name="persistenceProviderClass" value="org.hibernate.ejb.HibernatePersistence"/>
</bean>
```

Specify in which packages Spring Framework shall lookup for components (@Entity, @Repository, etc.). For that, add the following string in the `application-context.xml`:

```
<context:component-scan base-package="lab.model, lab.dao" />
```

Part 2: Creating `mapping` of the `Country` class.

4. Open the `Country` class. Add the `@Entity and the @Table(name = "COUNTRY")` annotations before class declaration.
5. Add the
   `@Id`
   `@Column(nullable = false)`
   annotations before defining the `getId()` method.
6. Add the `@Column` annotation before defining the `getName()` method.
7. Add the `@Column(name="code_name")` annotation before defining the `getCodeName()` method.
8. At the end you are going to have the code with following strings:

```
@Entity
@Table(name = "COUNTRY")
public class Country implements Serializable {
        private int id;
        private String name;
        private String codeName;

        public Country() { }

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        public int getId() {
                return id;
        }
        …
```

```
public void setId(int id) {this.id = id;}

@Column(name = "NAME")
public String getName() {
     return name;
}
…
@Column(name = "CODE_NAME")
public String getCodeName() {
     return codeName;
}
…
```

Part 3: Saving a country

9. Open the `CountryJpaDaoImpl` class. *Why this class inherits* `AbstractJpaDao` *and implements the* `CountryDao` *interface?*
10. *Where from and in what way the* `EntityManagerFactory` *(*`emf` *variable) is obtained?*
11. In the save method retrieve `EntityManager`, the `emf.createEntityManager()` method.
12. In `entityManager` invoke the `persist()` method **within programmatic transactions**, and define the saved object as a parameter.
13. Verify whether the implementation is correct or not using the `CountryDaoImplTest, testSaveCountry()` test.

Part 4: Retrieving the list of all countries

14. Open the `getAllCountries()` method in the `CountryJpaDaoImpl` class.
15. Retrieve `EntityManager`, invoke the `createQuery` method. Define query row and `Country.class` as a parameter. Verify whether the implementation is correct or not using the `CountryDaoImplTest, testGetAllCountries()` test.

16. You are going to have a code like that:

```
List<Country> countryList = null;
EntityManager em = emf.createEntityManager();
if (em != null) {
      countryList =
            em.createQuery("from Country", Country.class)
            .getResultList();
      em.close();
}
return countryList;
```

Part 5: Retrieving a country by its name

17. Open the `getCountryByName(String name)` method in the `CountryJpaDaoImpl` class.
18. Create a query for lookup and pass it to the retrieved `EntityManager` as follows:

```
Country country = em
       .createQuery("SELECT c FROM Country c WHERE c.name LIKE :name",
       Country.class).setParameter("name", name)
       .getSingleResult();
```

19. Verify whether the implementation is correct or not using the `CountryDaoImpTest, testGetCountryByName()` test.
20. As a result, you will have a code like that:

```
EntityManager em = emf.createEntityManager();
Country country = null;
       if (em != null) {
              country = em
              .createQuery("SELECT c FROM Country c WHERE c.name LIKE :name",
                     Country.class).setParameter("name", name)
                     .getSingleResult();
              em.close();
       }
       return country;
```

Additional questions:

21. *What other strategies for retrieving primary key can be used for entities?* (`@GeneratedValue(strategy = GenerationType.AUTO)`)
22. Switch root logger to the DEBUG mode (for that, edit the file log4j.properties)
23. *Try out various strategies for retrieving primary key. What is the difference between them?*

Exercise 8
Transaction management in Spring

Project

`lab-8-tx`

Duration

30 min

Objectives

Learn how to use declarative and programmatic transaction management in Spring.

Description

Spring allows for declarative and programmatic transaction management. It is also possible in Spring to integrate with abstractions for accessing data via ORM or JDBC. Here, transaction management is exemplified through JDBC.

Objectives

1. Create `TransactionManager` based on `DataSource`. Using annotations define declarative transactions support.
2. Define the `countryService` bean. Make all `CountryServiceImpl` class' methods transactional. Specify different values of `propagation` for methods according to methods names (for example, for the getAllCountriesRequired `()` method the `Propagation.REQUIRED` value should be specified).
3. Test different propagation variants for the `CountryService` methods using the `DeclarativeTransactionTestTest` test. Compare behavior of methods that have been invoked within and out of the transaction at different `propagation` methods.

Detailed guidelines

Part 1: Creating `TransactionManager`

1. Open the file with the `application-context.xml` configuration.
2. Add following configuration:

```
<context:annotation-config/>
<context:component-scan base-package="lab.service" />
```

What is the purpose of each string?

3. Create the `transactionManager` bean for the `org.springframework.jdbc.datasource` class. `DataSourceTransactionManager`. Specify the reference to `DataSource`.

4. Add the `<tx:annotation-driven />` string that activates annotation-based declarative transaction management.

Part 2: Configuring `CountryServiceImpl`

5. Create the `countryService` bean for the `CountryServiceImpl` class. Specify a reference to `dao`.
6. Open the `CountryServiceImpl` class. Add the `@Transactional` string before class definition. All the class methods will become transactional with default settings.
7. To change settings of specific methods, it is necessary to add the `@Transactional(readOnly = false, propagation = Propagation.REQUIRED)` definition before method declaration. Specify relevant definitions for all methods like `getAllCountriesRequired()`, changing only `propagation` value according to method name.

Part 3: Testing declarative transaction definition

8. *How should method behave with* `propagation REQUIRED` *that was invoked within and out of the transaction? With* `propagation MANDATORY`*?*
9. Verify whether the implementation is correct or not using the `DeclarativeTransactionTest` test.

Exercise 9
Developing primitive application with Spring 3 MVC

Project

`lab-9-mvc`

Duration

1 hour

Code base

lab-9-mvc

Objectives

Develop the simplest working application using Spring 3 MVC.

Description

Any application that runs on Spring MVC should have several mandatory elements. In application descriptor, it is necessary to register at least one `DispatcherServlet` and define the requests it handles. It is also essential to register a listener like `ContextLoaderListener` for initialization of Spring context.

To see how the application works one has to create the controllers and views. Specify handled requests for each controller. For views, specify mapping between row names and views.

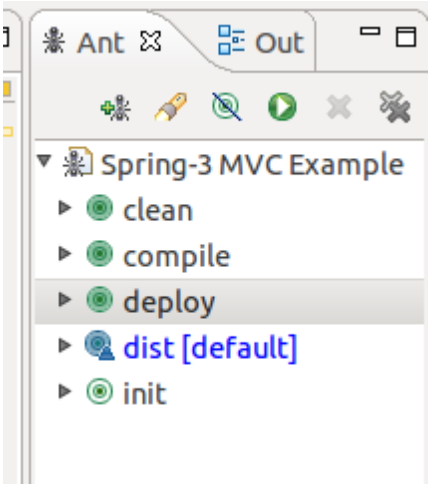
Objectives

1. Study the application, single out all components and examine them.
2. Rework the application so that it could execute similar operations for the Country entity (see previous exercises).

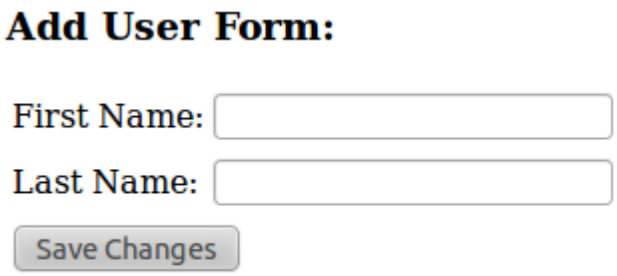
Detailed guidelines

Part 1

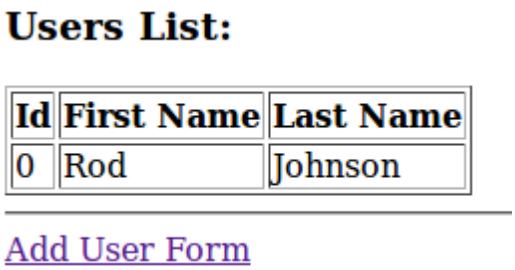
1. Make sure that the $CATALINA_HOME system variable is defined and indicates the directory with pre-installed servlet container TomCat. For that, enter following command in the line of command: echo $CATALINA_HOME. Displaying command shall indicate ~/bin/tomcat.
2. Open the application assembly control panel in Eclipse: Window → Show view -> Ant



3. Select the `deploy` target and execute it;
4. If application assembly is successful, start the TomCat servlet container;
5. Open the browser and indicate in address bar following url: http://localhost:8080/lab-9-mvc/adduser.form
6. If you do everything correctly, you will see the following form

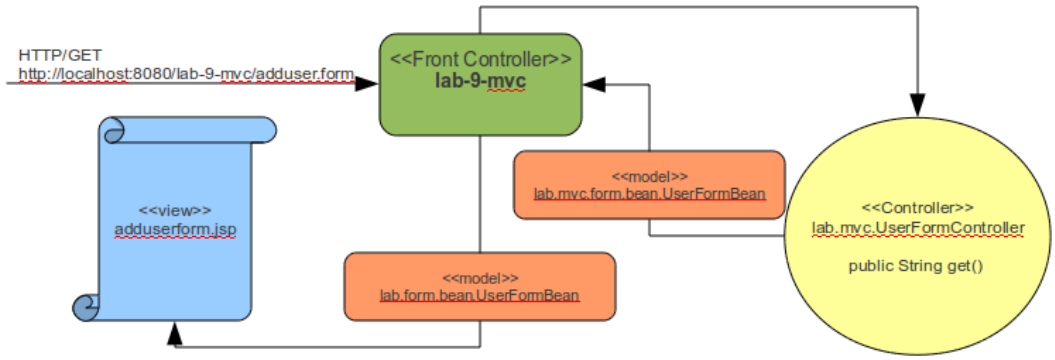


7. Fill the form and click [Save Changes]
8. If you do everything correctly, you will see the following screen:

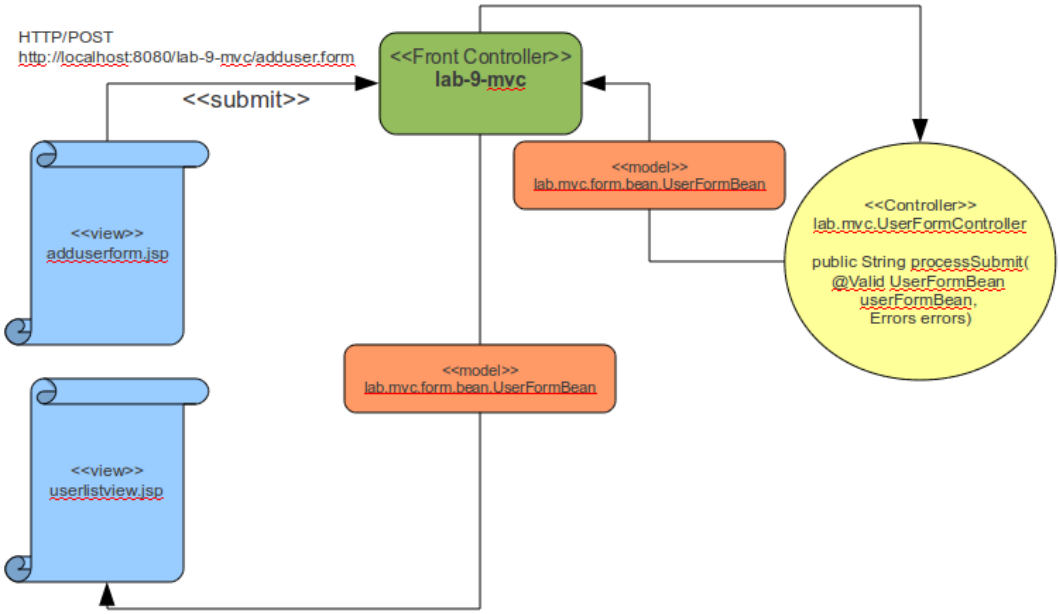


9. Repeat steps 6 and 7 leaving the fields blank and with less than 2 symbols. What is the result?

10. Thoroughly study all the project artifacts and MVC components dependency diagram:



GET



POST

Answer following questions:

1. Which artifact describes all beans of the given application?
2. To which controller the control is transferred, when handling the address http://localhost:8080/lab-9-mvc/adduser.form? Why?
3. How the address http://localhost:8080/lab-9-mvc/ is handled at the moment? How can you change this behavior?

4. When a browser request is done through the GET method, the execution is passed to `adduserform.jsp`, but if using the POST method, the control can return both to `adduserform.jsp` and `userlistview.jsp`. Why?
5. How do you verify whether the form was filled correctly or not?

Part 2

Develop the following artifacts:

1. `lab.dao.CountryDao`
2. `lab.dao.HsqlCountryDao`
3. `lab.domain.Country`
4. `lab.mvc.form.bean.CountryFormBean`
5. `lab.mvc.CountryFormController`
6. `web/WEB-INF/views/addcountryform.jsp`
7. `web/WEB-INF/views/countrylistview.jsp`

Detailed guidelines

1. Using the code base from the previous part of the exercise and developing above-listed artifacts, assemble the application.
2. Start the TomCat servlet container and enter http://localhost:8080/lab-9-mvc/addcountry.form in a browser address bar
3. Make sure that application saves added information about country in the database.

Exercise 10
Using task scheduling

Project

`lab-10-quartz`

Duration

30 min

Objectives

Learn how to launch tasks by timer using `Quartz scheduler` in `Spring`.

Description

`Spring` offers auxiliary `API` for work with timers and allows configuring them through configuration files.

`Quartz scheduler` is third-party software (http://www.quartz-scheduler.org/). It can run tasks in specified time intervals. Moreover, it makes it possible to run a task at some point in future (e.g., specific date or day of week).

`Spring` makes it possible for these schedulers to be configured via configuration files. It disposes of difficulty to start and stop the timer as well.

Objectives

1. Change configuration of the `schedulerFactoryBean` bean from `application-context.xml` and add necessary parameters for `reportTrigger`, so that the `PrintingJob` task would be executed every second, beginning with the fifth second after the start. To check, use the `QuartzJobTest` test, the `testRepeatableJob` method.
2. Change configuration of the `schedulerFactoryBean` bean from `application-context.xml` and add necessary parameters for `reportTrigger`, so that the `PrintingJob` task would be executed every three seconds only during the current hour, current day of week, current month. To check, use the `QuartzJobTest` test, the `testCronJob` method.

Detailed guidelines

Part 1: New features of Spring 3.1 that facilitate the testing

1. Open the file with the `application-context.xml` configuration.
2. Note that in this exercise some updates appear in the `application-context.xml` file: some beans are declared in blocks: `<beans profile="interval">`, and unit test classes are additionally annotated `@ActiveProfiles("interval")`. Therefore, when executing specific unit tests, only beans declared within the specific profile are loaded.

3. Analyze the context! *How do you think, why we waited so long before we represented this feature?*
4. Thoroughly study annotations to bean declarations: Spring 3.1 supports Quartz 2.* whose API underwent significant modifications.

Part 2: Configuring PrintingJob with SimpleTriggerFactoryBean.

5. Open the `PrintingJob` class. Analyze it. *Which class it inherits from? Which methods were redefined?*
6. Find bean declaration block for the `interval` profile.
7. Add following property for the `schedulerFactoryBean` bean

```
<property name="triggers">
      <list>
            <ref bean="reportTrigger" />
      </list>
</property>
```

8. Configure the `reportTrigger` bean by adding necessary properties:

```
<property name="jobDetail" ref="reportJob" />
      <property name="repeatInterval" value="1000" />
      <property name="startDelay" value="5000" />
```

9. *What is the meaning of each property?*
10. Execute the `QuartzJobTest` test, the `testRepeatableJob` method.

Part 3: Configuring PrintingJob with CronReportTrigger via Quartz Scheduler.

11. Find bean declaration block for the `cron` profile.

Configure the `reportTrigger` bean by adding necessary properties:

```
<property name="jobDetail" ref="reportJob" />
<property name="cronExpression" value="0/3 * * ? m dw" />
```

12. Substitute `m` for the month number, `dw` for the day of week number (where 1 is Sunday, 2 is Monday and so on).
13. *What is the meaning of each property?*
14. *What is the meaning of each* `cronExpression` *parameter*?
15. Execute the `QuartzCronJobTest` test.

Part 4: Using annotations in task scheduling

1. Open the `application-context-ad.xml` and examine the configuration and new features offered by Spring for annotation-based task scheduling.
2. Open the `ScheduledTask` class, examine the code. *How Spring knows that the* `doSomething()` *method shall be executed by scheduler?*
3. Execute the `ScheduledTest` test, the `testRepeatableJob()` method.