INSTRUCTIONS

Spring Boot Introduction

Purpose

In this lab you will gain experience with creating a Spring Boot project from the beginning, taking advantage of the Spring starters to easily configure your project.

Learning Outcomes

What you will learn:

1. How to initialize a project
2. How to select the right starters for your project

Specific subjects you will gain experience with:

1. Using https://start.spring.io to create a Boot-enabled project
2. Adding starter dependencies to your Maven or Gradle project
3. Determining what configuration is enabled

There is no project for this lab - you will use the Initializr to create it.

Estimated time to complete: 30 minutes.

(A solution is provided for reference: *30-jdbc-boot-solution*).

Getting Started

If you attended the Spring *Essentials* segment of the *Core Spring* course you should already be familiar with the Rewards Application.

Otherwise, if this is the first day of your Spring Boot course, read about the Rewards Application now before continuing. We use this application as the basis for the remaining labs.

Use Case

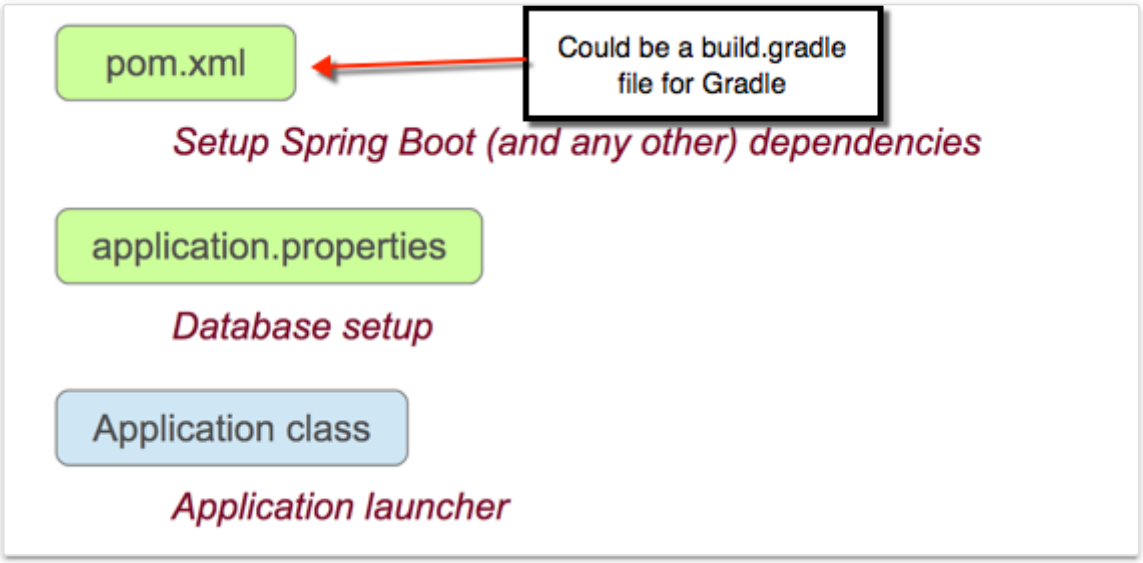
Imagine you wanted to create a very simple application to access and display information about the accounts in your Reward database.

In a traditional Spring project, you enjoy the developer productivity provided by the framework to simplify application development. However, configuring a project with the right dependencies to ensure you have all the right Spring libraries can be challenging and time-consuming.

Consider the necessary setup just to configure a JDBC project.

- You need to obtain the right JDBC drivers for the database you are using
- You need to obtain the core Spring and Spring JDBC libraries
- You need to configure your JDBC connections and potentially initialize
- You may wish to add JDBC connection pooling for efficiency
- You may need to set up database initialization with schema definitions and data population

As you will discover, the most basic Spring Boot application really only requires the following three files to get started.



You will use the *Spring Initializr* to create a basic JDBC Spring Boot Project using an HSQLDB (Hyper SQL) database. You will make a few minor modifications to add SQL scripts to create tables and populate the database with data for the Reward Network application. The initial application will use a very simple JDBC call to query the *T_ACCOUNT* table and display the

total number of accounts. Using Spring Boot, you will do this with a minimal amount of setup and configuration.

Instructions

Create and run a Spring Boot project

You are going to create a Spring Boot project called **jdbc-boot** that will serve as a very basic application to expose account details.

Depending on your IDE, a new project wizard may be available that uses the *Spring Initializr* for you without leaving the IDE:

- **Spring Tool Suite:** *File -> New -> Spring Starter Project* (if not there use *File -> New -> Other -> Spring Boot -> Spring Starter Project*)
- **IntelliJ IDEA Ultimate Edition:** *File -> New -> Project -> Spring Initializr*

Alternatively, use the website directly, download the generated zip and import into your IDE as described below.

1. Use either your IDE or the Spring Initializr Site to create a new Spring Boot project with the following project values

Field	Value
Project	Maven Project or Gradle Project
Language	Java
Spring Boot Version	2.7.5 (or latest 2.x minor version)
Group	io.spring.training.boot
Artifact	jdbc-boot
Name	jdbc-boot
Description	First Boot Project
Package name	rewards
Packaging	JAR
Java	11 or later version based on your workstation setup

Refer to the appendix for details on how to create a project or module using Spring Boot Initializr.

Select the following dependencies to add to the project.

- **JDBC API** (under the SQL category) - Adds general JDBC support, including the Spring JDBC libraries
- **HyperSQL Database** (also under the SQL category) - Adds specific HSQL libraries and drivers needed for the HSQL database

If using a wizard in the IDE, be sure to specify the final project folder name as **jdbc-boot**

2. If you did not use the wizard to create the project, you will need to import the project
3. Take a moment to explore the project that was just created.
   - Open the `pom.xml` or `build.gradle` file and note the dependencies and the parent reference. What 3 dependencies are defined?
   - Now, open the `JdbcBootApplication` (assuming you've named your application as `jdbc-boot` in the previous step). What are the key elements of this file that might make it behave as a Spring Boot application?
4. Set the logging level to DEBUG by adding the following line to the `src/main/resources/application.properties` file

   ```
   logging.level.root=DEBUG
   ```

5. Run the `JdbcBootApplication`.

   The application should run successfully. And you should see `CONDITIONS EVALUATION REPORT`, which shows that a set of AutoConfiguration classes including `DataSourceAutoConfiguration` are executed. (Search for "DataSourceAutoConfiguration matched:" string using IDE's search capability.)

```
============================
CONDITIONS EVALUATION REPORT
============================


Positive matches:
-----------------

   AopAutoConfiguration matched:
      - @ConditionalOnProperty (spring.aop.auto=true) matched (OnPropertyCondition)

   AopAutoConfiguration.ClassProxyingConfiguration matched:
      - @ConditionalOnMissingClass did not find unwanted class 'org.aspectj.weaver.Advice' (On
      - @ConditionalOnProperty (spring.aop.proxy-target-class=true) matched (OnPropertyConditi

   DataSourceAutoConfiguration matched:
      - @ConditionalOnClass found required classes 'javax.sql.DataSource', 'org.springframewor
      - @ConditionalOnMissingBean (types: io.r2dbc.spi.ConnectionFactory; SearchStrategy: all)
```

Now what can you determine from the output?

- How was your application able to create a JDBC Datasource?
- Ordinarily, a JDBC DataSource is initialized using a JDBC driver class, the URL to the database server and more. Since these were not provided explicitly by you, where did those values come from?

At this point, your application does not do too much since you are merely running the default application created from the Spring Boot Initializr.

Use SQL scripts

1. Copy the `schema.sql` and `data.sql` files from the `26-jdbc-solution` project. You will find them in `src/test/resources/rewards/testdb`. Copy the files to the `src/main/resources` folder of your new project. Make sure the two files reside directly under the `resources` folder.
2. Re-run the `JdbcBootApplication` and notice the console output now shows a bit more taking place.
   - Note that a `HikariPool` JDBC connection pool was created and configured for the DataSource.
   - Notice also that the two scripts you copied to the `resources` folder now automatically get executed, resulting in the database being initialized with the tables defined in the `schema.sql` file and loaded with the data in the `data.sql` file. (Search for `Executed SQL script` string in the console.)

     It turns out that there was something special about the location where you copied the SQL scripts (`src/main/resources`) because in a Maven or Gradle project, this directory is *always* on the classpath. The SQL files were detected and invoked

because Spring Boot automatically looks for and runs two files `schema.sql` and `data.sql` if it finds them on the classpath.

Alternatively, you could have placed those files anywhere in your project and named them anything you wanted. In that case, you would need to add the following entries to your `application.properties` file to tell your Spring Boot application where to find these files.

```
spring.sql.init.schema-locations=<path to your schema SQL file>
spring.sql.init.data-locations=<path to your data SQL file>
```

Use CommandLineRunner Bean

The `run` method of `CommandLineRunner` bean gets executed when a Spring Boot application gets started. (We are going to learn on `CommandLineRunner` in the next module.)

In this step, you are going to use `JdbcTemplate` bean to retrieve number of accounts.

1. Open the `JdbcBootApplication` class. Add the following code in place of the default template code.

```
package rewards;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.core.JdbcTemplate;

@SpringBootApplication
public class JdbcBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(JdbcBootApplication.class, args);
    }

    @Bean
    CommandLineRunner commandLineRunner(JdbcTemplate jdbcTemplate){

        String QUERY = "SELECT count(*) FROM T_ACCOUNT";

        // Use Lambda expression to display the result
        return args -> System.out.println("Hello, there are "
            + jdbcTemplate.queryForObject(QUERY, Long.class)
```

```
            + " accounts");
    }
}
```

2. Re-run the `JdbcBootApplication`.

   Note that amidst the typical Spring Boot application output to the console, there is a line that says: (you might need to perform text search in the console)

   `Hello, there are 21 accounts`

Here are some questions to consider

- Where did the `JdbcTemplate` bean come from?
- You typically initialize or create the `JdbcTemplate` with an instance of `DataSource`. How did this get created and configured and dependency injected into the `JdbcTemplate`?

Set Logging level

1. Set logging level of a package (`root` in this example) in the `application.properties` file

```
logging.level.root=ERROR
```

Customize banner

1. Create custom banner by creating `banner.txt` under `src/main/resources` folder. You can generate a banner text from here and copy the generated text into the `banner.txt` file.

2. Re-run the application. You should now see the following output if everything is configured properly.

```
  _________             .__                 __________                 __
 /   _____/_____________|__| ____    ____   \______   \ ____   _____/  |_
 \_____  \\____ \_  __ \  |/    \  / ___\   |    |  _//  _ \ /  _ \   __\
 /        \  |_> >  | \/  |   |  \/ /_/  >  |    |   (  <_> |  <_> )  |
/_______  /   __/|__|  |__|___|  /\___  /   |______  /\____/ \____/|__|
        \/|__|                 \//_____/           \/

Hello, there are 21 accounts
```

```
Process finished with exit code 0
```

Write integration testing code

1. Write testing code using `@SpringBootTest` annotation.

```
package rewards;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.jdbc.core.JdbcTemplate;

import static org.assertj.core.api.Assertions.*;

@SpringBootTest
public class JdbcBootApplicationTests {
    public static final String QUERY = "SELECT count(*) FROM T_ACCOUNT";

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Test
    public void testNumberOfAccount() {

        long count = jdbcTemplate.queryForObject(QUERY, Long.class);

        assertThat(count).isEqualTo(21L);
    }

}
```

2. Run the test and verify it succeeds.

Summary

In this lab, you have possibly gotten your first taste of the workings of Spring Boot.

- You have seen the basic steps to quickly bootstrapping a Spring Boot project with a technology such as JDBC, using one of the various methods of accessing the Spring Initializer.

- You have seen how, simply by having certain dependencies and their associated classes on the classpath, certain beans will be created.
- You have seen how simple it is to create a basic JDBC application with a minimal amount of configuration and writing code.
- You have seen how basic configuration of your Spring Boot application can be performed through the `application.properties` file.

Many of the features you have seen here will be explored further in later modules of the course.

Congratulations, you are done with the lab!