TRAINING MANUAL

Springing the Boot
With VS Code

—

Haris Hashim
© 2020 harishashim@gmail.com

Overview

A step by step training manual style instruction to accompany mentor and mentee discussion during the learning session. This is an introduction to Spring Boot.

IMPORTANT This document does not include explanations that are done during one to one mentorship session.

Table of Contents

Prerequisite

Knowledge

1. Java!

Software

1. VSCode
2. Java JDK
3. Maven

VSCode Extension

1. CTRL-SHIFT-X to open extension pane
2. Install the following Extension
   1. Java Extension Pack by Microsoft
   2. Spring Initializr Java Support
   3. Spring Boot Dashboard by Microsoft
   4. Java Code Generators by Sohibe

Steps

Creating & Running Spring Boot Project

1. CTRL-SHIFT-P to open VSCode command palette
2. Type Spring as a shortcut to get the full command and select "Spring Initializr: Generate a Maven Project.
3. At the "Specify project language" prompt, select Java.
4. At the next prompt, press Enter for "com.example" or enter another more suitable for Maven project group id.
5. After that just specify "task" as maven artifact id.
6. Select the suggested Spring version for "Specify Spring Boot version" prompt. In my case it is 2.2.4.
7. Next step is to specify dependencies. These depencencies can be found later in the pom.xml file. Go ahead and choose bellow dependencies:
   1. Spring BootDevtools
   2. Spring web
8. After the above is selected, the top most choice in "Search for dependencies" prompt is "Selected 2 dependencies". Select that option with mouse or press Enter to proceed.
9. A dialog box showing where you are saving the project will be shown. It will ask you to select a parent folder for your project folder. By the way, your project folder will be named "task" as you already specified above. So there is no need to create an additional folder and name it task. Simply select a folder that will be parent folder to your project folder.
10. VSCode will display notification that it successfully creates your project folder. Click open so that VSCode can open your project folder.
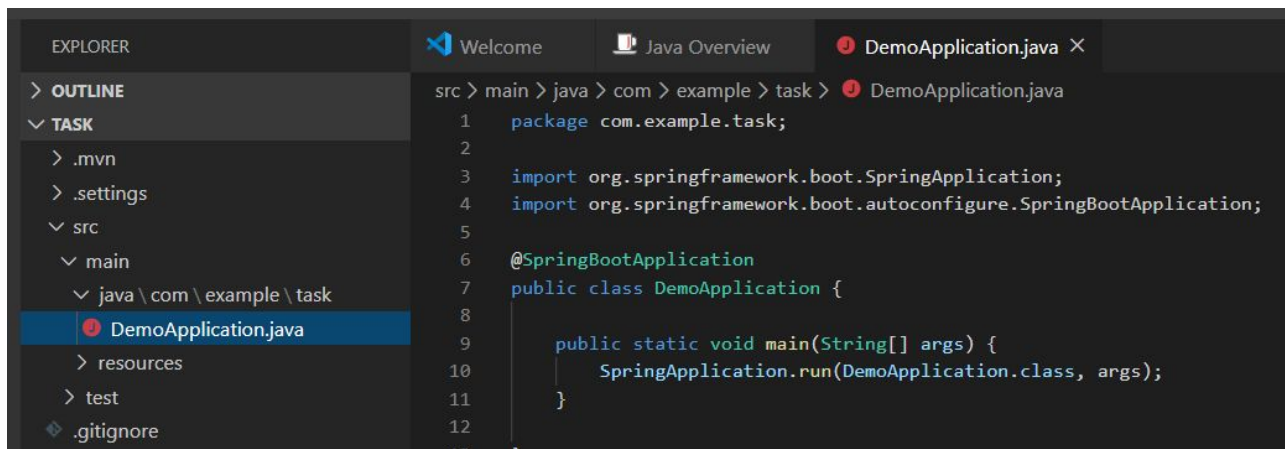
11. In Explorer pane, expand src folder to find DemoApplication.java file. Rename that file to TaskApplication.java and apply a quick fix after that to rename DemoApplication class to TaskApplication.
12. At the bottom of the Explorer pane, find Spring Boot Dashboard drop down. Expand the drop down, select your Spring Boot project named "task" and run it!
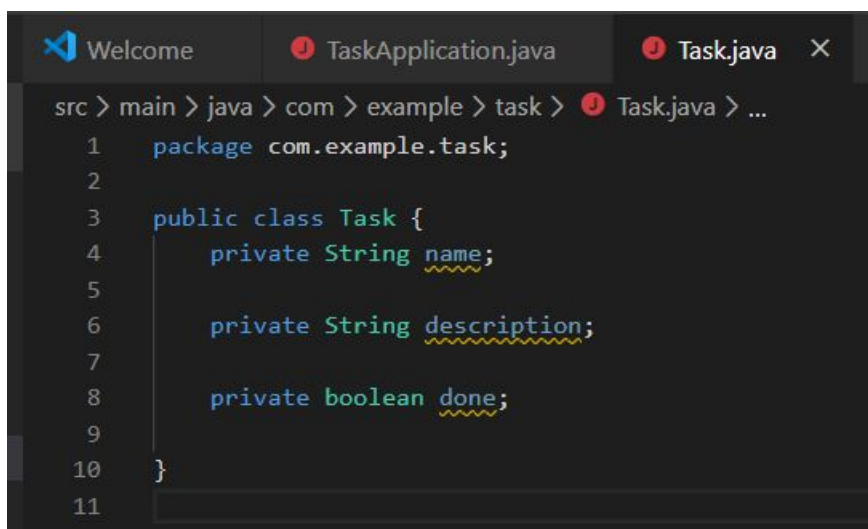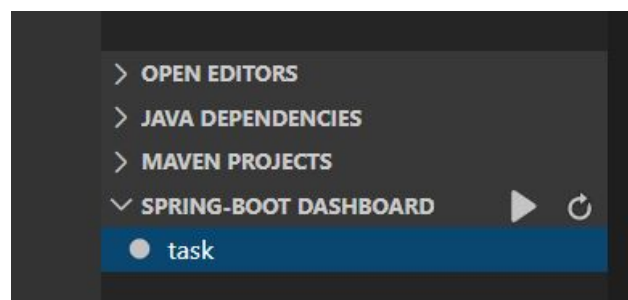

Doing the above will run you first Spring Boot project in VSCode. Congratulations!

Creating Your First Rest Service

In the next exercise, we will create a Plain Old Java Object (POJO) that can be accessed over REST service and test it using Postman app. After that, two items will be created; a REST controller and a service. In MVC, a controller is an example of a presentation layer while service is the service layer that holds business logic.

1. Create a POJO, that represents a task with task name, description and done status.
    1. This means create a file using VSCode in your package and named it Task.java
    2. Create a class named Task inside that file.
    3. Add the class private members. The class should look as follow:

1. Now we are going to generate a lot of codes using Java Code Generators extension. Do CTRL-SHIFT-P and select 'Java: Generate all "Quick"'. Or simply right click on the code and look for the same option from the context menu.
2. The code generator will generate a lot of codes, normally I will delete some of them. In this case I am going to delete
    - public boolean getDone()
    - public Task name(String name)
    - public Task description(String description)
    - public Task done(boolean done)

3. Create an interface for a service that gets the task.
    1. Create a file in your package and name it TaskServiceInterface.java
    2. In that file create an Interface with getTask() stub inside it. The end result should be as follows.

```
src > main > java > com > example > task > ⊘ TaskServiceInterface.java
 1    package com.example.task;
 2    💡
 3    /**
 4     * TaskServiceInterface
 5     */
 6    public interface TaskServiceInterface {
 7
 8        public Task getTask(int index);
 9
10    }
11
```

4. Implement that interface and decorate it with @Service
    1. Create a file in your package and name it TaskServiceImpl.java.
    2. Create a class that implements TaskServiceInterface, make sure getTask() stub is implemented and the class is started with @Service (org.springframework.stereotype.Service).
    3. To store the task, we will simply use ArrayList and to initialize the array, we will add a few tasks in constructor. The end result should look as follows.

```
src > main > java > com > example > task > ⊘ TaskServiceImpl.java > ...
11    @Service
12    public class TaskServiceImpl implements TaskServiceInterface {
13
14        private List<Task> tasks = new ArrayList<>();
15
16        public TaskServiceImpl() {
17            tasks.add(new Task("Task 1", "Task one", false));
18            tasks.add(new Task("Task 2", "Task two", true));
19            tasks.add(new Task("Task 3", "Task three", false));
20        }
21
22        @Override
23        public Task getTask(int index) {
24            if (tasks.size() == 0) {
25                return null;
26            } else {
27                return tasks.get(index);
28            }
29        }
30
31    }
```

1. Create a controller that presents Task service through REST service.
    1. Create a file in your package and name it TaskController.java.
    2. Create a TaskController class with the following decoration
        1. @RestController - Because it is a rest controller, duh!
        2. @RequestMapping("/api/v1") - Because this is the path after your url
        3. @CrossOrigin - Because as API provider, the request will be originating from other server host.
    3. Bring in task service into the controller by using @AutoWired decoration. Take note that DI (Dependency Injection) makes use of TaskServiceInterface instead of TaskServiceImpl.
    4. Add a getTask function decorated with @RequestMapping that simply executes task service getTask function. The end result should be as follows.

```
src > main > java > com > example > task > 🔴 TaskController.java > Spring Boot Tools > •o @CrossOrigin
15    @RestController
16    @RequestMapping("/api/v1")
17    @CrossOrigin
18    public class TaskController {
19
20        @Autowired
21        TaskServiceInterface taskService;
22
23        @RequestMapping(value = "/tasks/{id}", method = RequestMethod.GET)
24        public ResponseEntity<Task> getTask(@PathVariable(value = "id") int id) {
25            return new ResponseEntity<Task>(taskService.getTask(id), HttpStatus.OK);
26        }
27
28    }
```

    5. Take note that @RequestMapping decoration specify which part of the url is a parameter. In this case, id is a parameter that specify the task index. While @PathVariable decoration map that id to one of the function parameters.
    6. Run the code, troubleshoot errors and execute url "localhost:8080/api/v1/tasks/0" in Postman. Verify that the result is indeed the first task that we initialize in TaskServiceImpl constructor.

If the above is successful, you already manage to run your first REST service using Spring Boot. Congratulations again!