# Version History

|  |  |  |
| --- | --- | --- |
| Version | Date | Release Note |
| V0.1 | 10-Mar-2020 | Initial Draft |
| V1.0 | 2-Apr-2020 | 1st Release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

index

1 Version History 2

2 Exercise 1 4

2.1 PREREQUISITE INSTALLATION 4

2.2 TDD Exercise 7

# Exercise 1

The exercise will demostrate what is Test Driven Development

Prerequisite

Java installed (e.g. JDK 11)

Maven installed (e.g. version 3.6.3)

Spring Tool Suite (STS) (e.g. 4.5.1)

Postman (e.g. version 7.20.1)

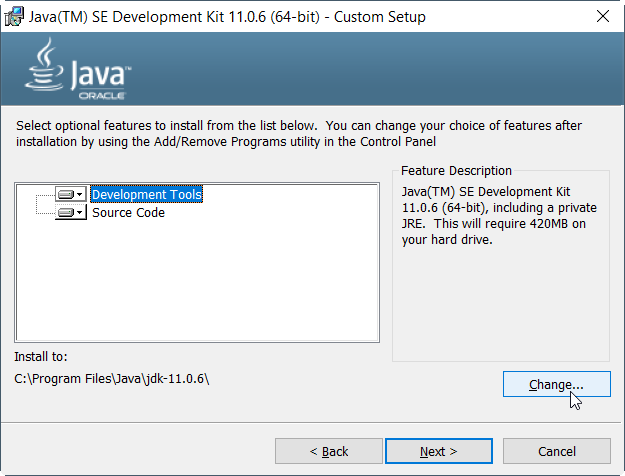
## PREREQUISITE INSTALLATION

### Java Development Kit

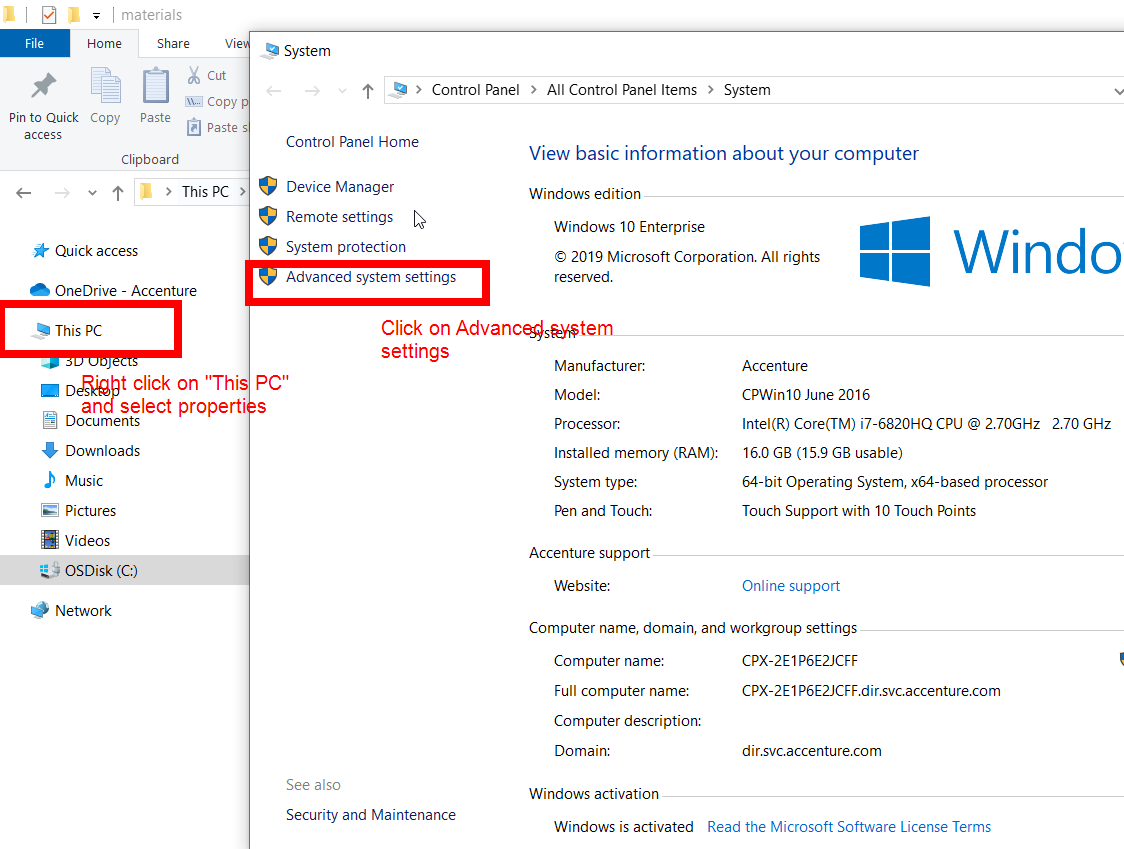
Download JDK 11 from <https://www.oracle.com/java/technologies/javase-jdk11-downloads.html>. You must create an account for downloading ( Make sure both Java and Spring Tool Suite are both 32bit or both 64bit). We are using 64bit in the exercises.

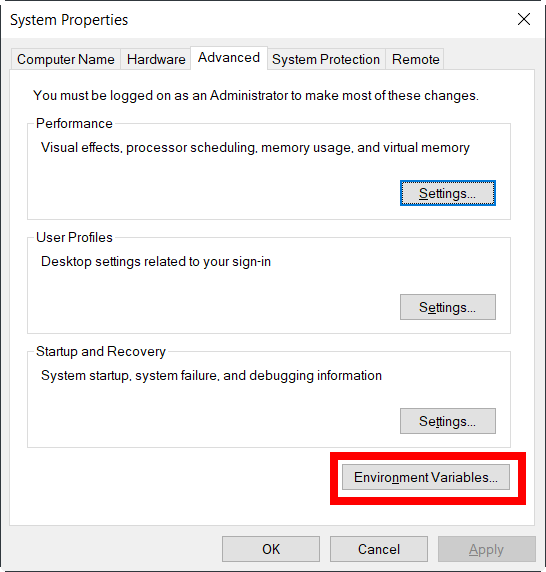
After download, double click the file and follow the installation wizard

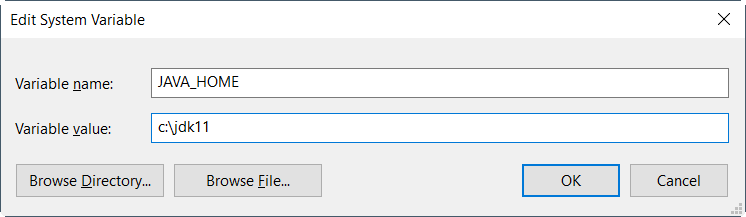
Recommend to change the default installation path to **c:\jdk11.**

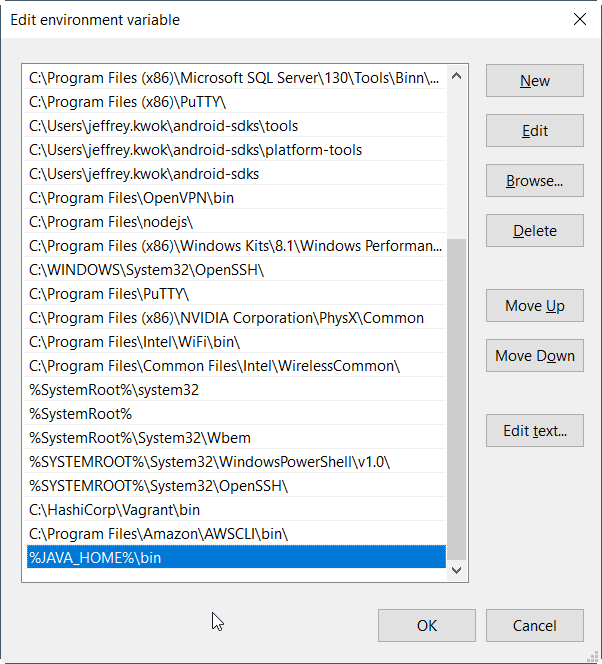


#### Environment Path Setup

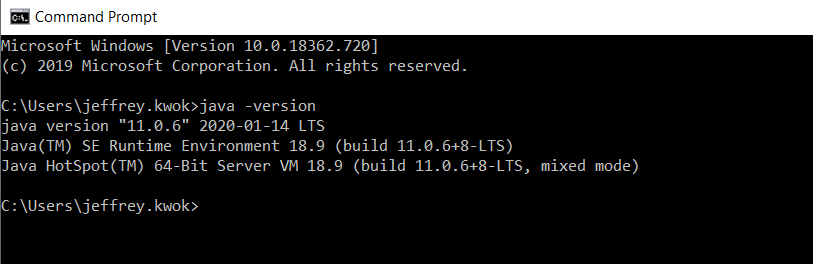
Add JAVA\_HOME and JAVA\_HOME/bin to your environment path

Create a new system path JAVA\_HOME variable to the JDK folder you just installed

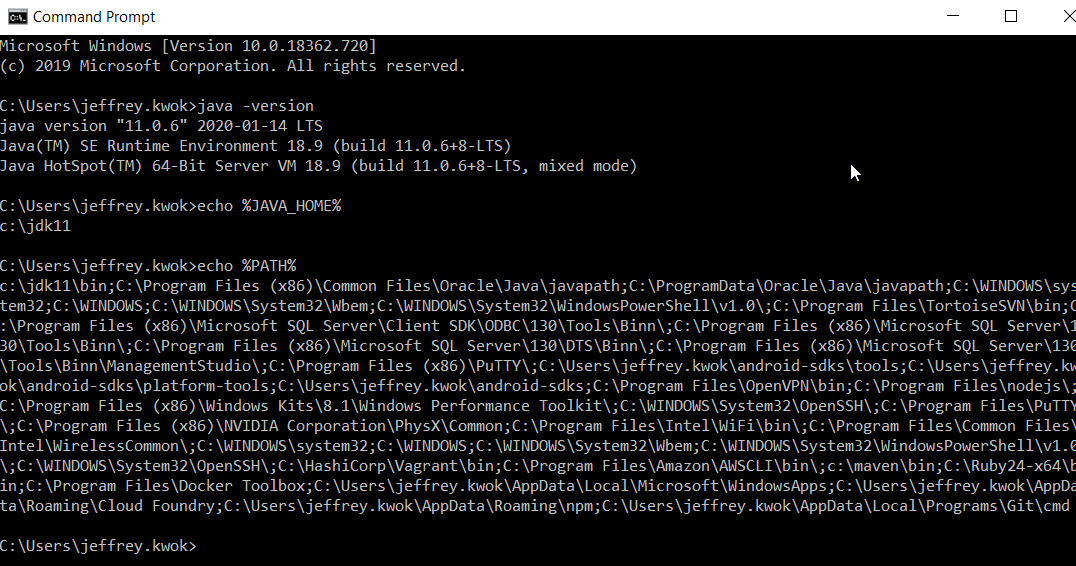


Update System **Path** variable and append %JAVA\_HOME%\bin

You may need to “Move Up” the %JAVA\_HOME%\bin to the toppest row if multiple java path is configured before

After completed, new a “Command window” and enter “java -version”. Java should be executed succesfully and return JDK 11 version

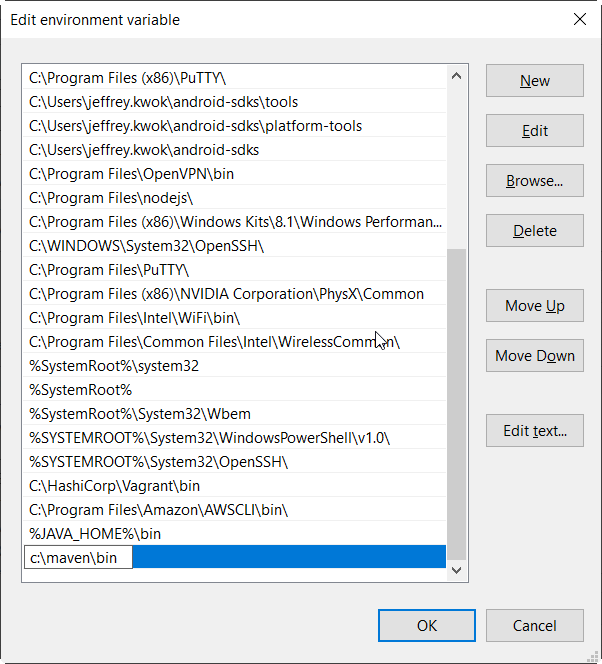
If does not work, you can echo the %JAVA\_HOME% and %PATH% to verify the setting. Sometime, restart windows may help



### Maven

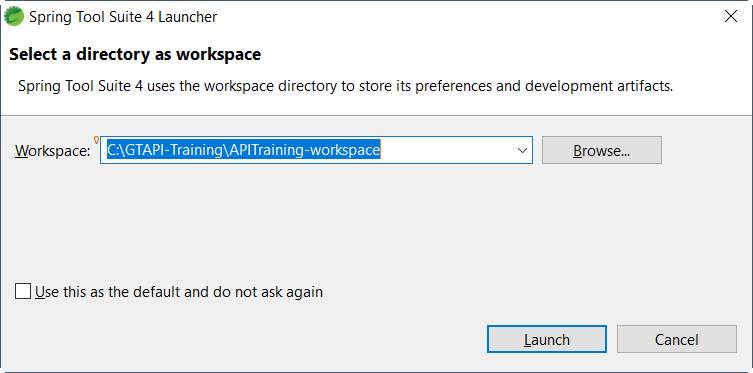
Download Mave from [**https://maven.apache.org/download.cgi**](https://maven.apache.org/download.cgi)and unzip the file. E.g c:\maven

Append the Maven **bin** path to **PATH** environment variable



### Spring Tool Suite

Download Spring Tool Suite from <https://spring.io/tools> and unzip it

Double click on bin\SpringToolSuite4.exe and and ask the workspace. You can enter a new workpace name and click launch. STS will creat a folder for you.

### Postman

Download Postman from <https://www.postman.com/> and install it according to Wizard

## TDD Exercise

### Generate sample pom.xml

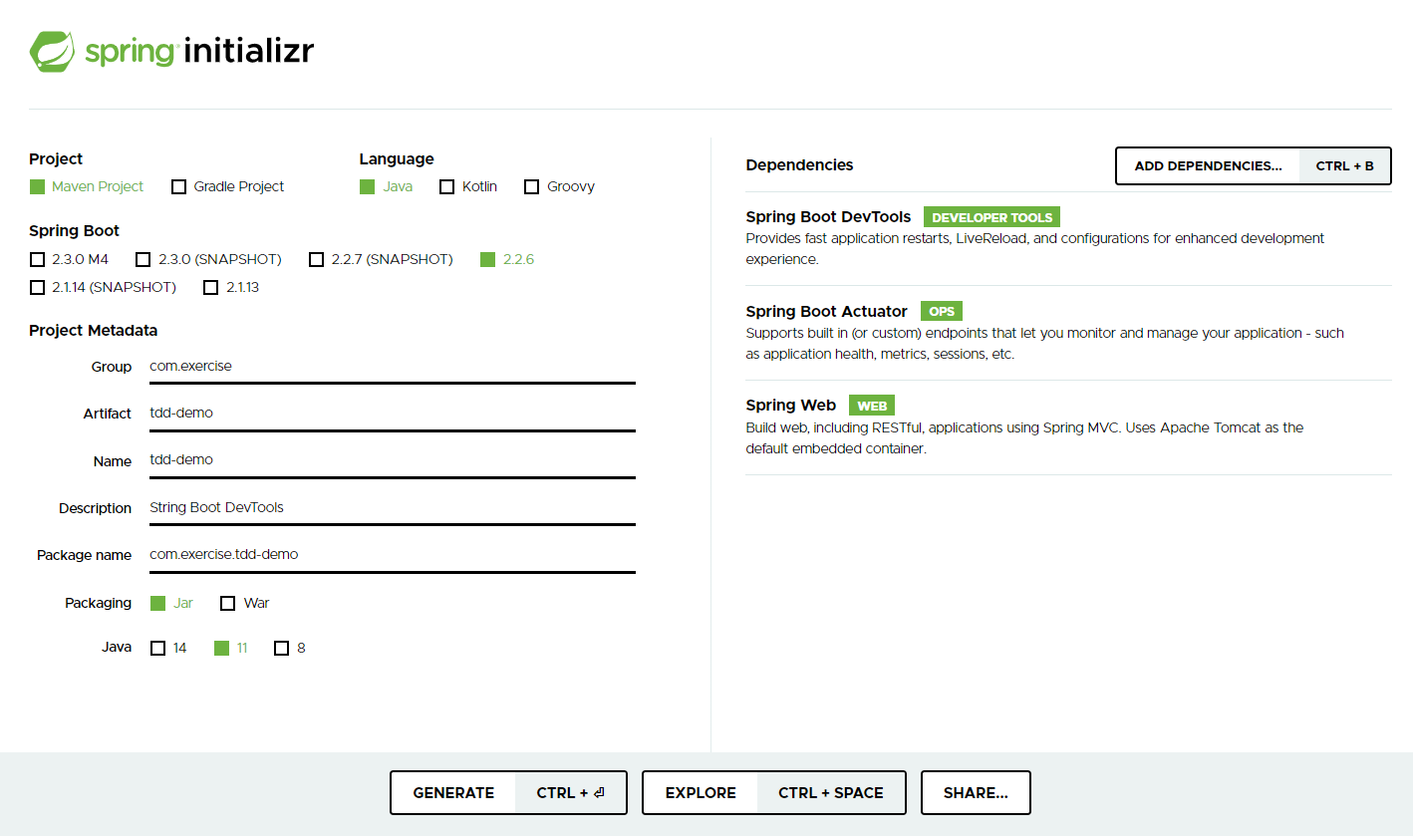
Goto <https://start.spring.io> and input the following fields:

Group: com.exercise

Artifact: tdd-demo

Dependencies: Spring Boot DevTools, Spring Web and Spring Boot Actuator

2. Click “Generate” and extract the zipped file to your local directory, e.g. C:\java-ex:



Or you may use this one:



C:\JAVA-EX\TDD-DEMO

│ .gitignore

│ HELP.md

│ mvnw

│ mvnw.cmd

│ pom.xml

│

├───.mvn

│ └───wrapper

│ maven-wrapper.jar

│ maven-wrapper.properties

│ MavenWrapperDownloader.java

│

└───src

├───main

│ ├───java

│ │ └───com

│ │ └───exercise

│ │ └───tdddemo

│ │ TddDemoApplication.java

│ │

│ └───resources

│ │ application.properties

│ │

│ ├───static

│ └───templates

└───test

└───java

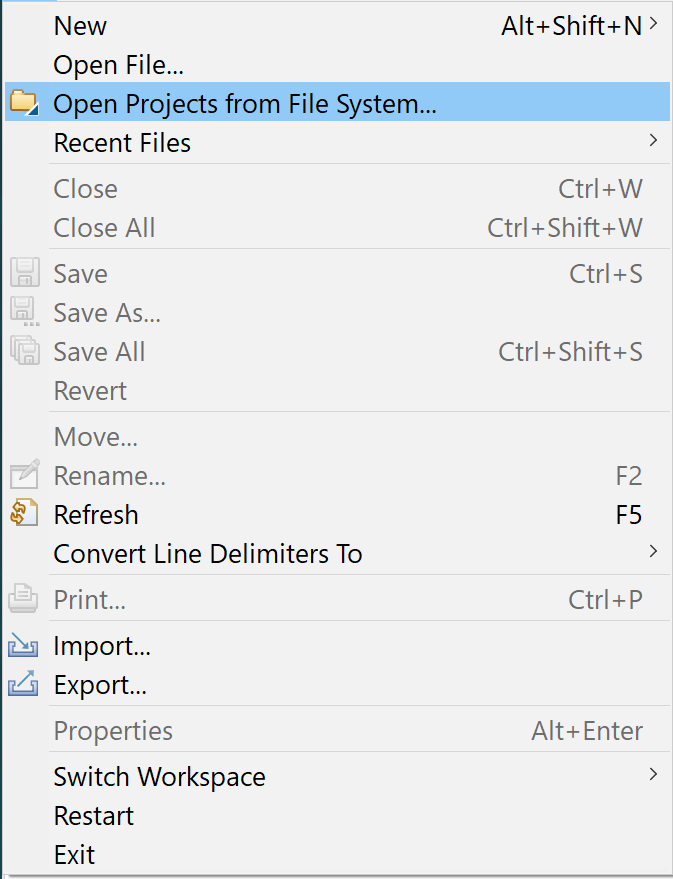
└───com

└───exercise

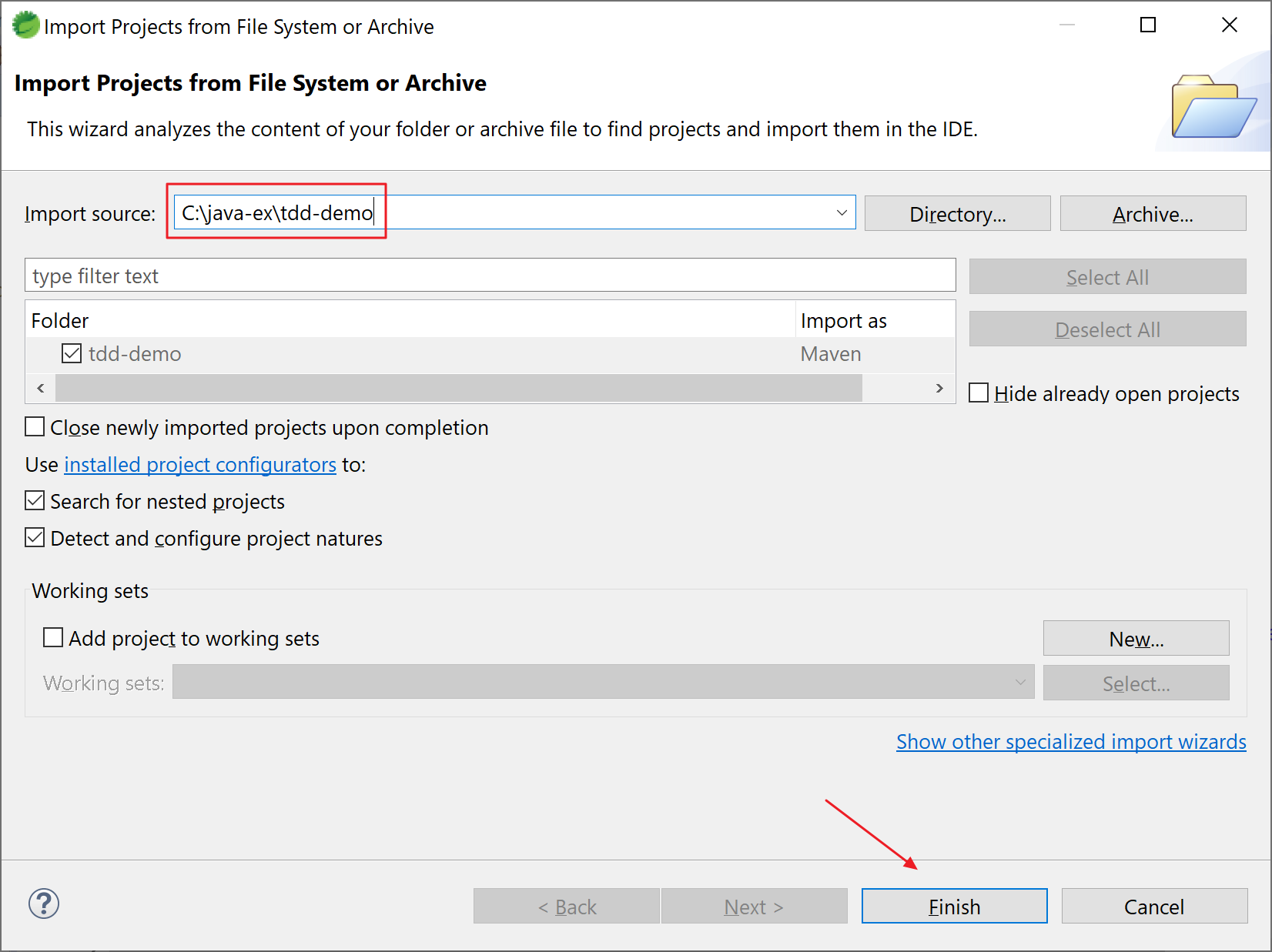
└───tdddemo

TddDemoApplicationTests.java

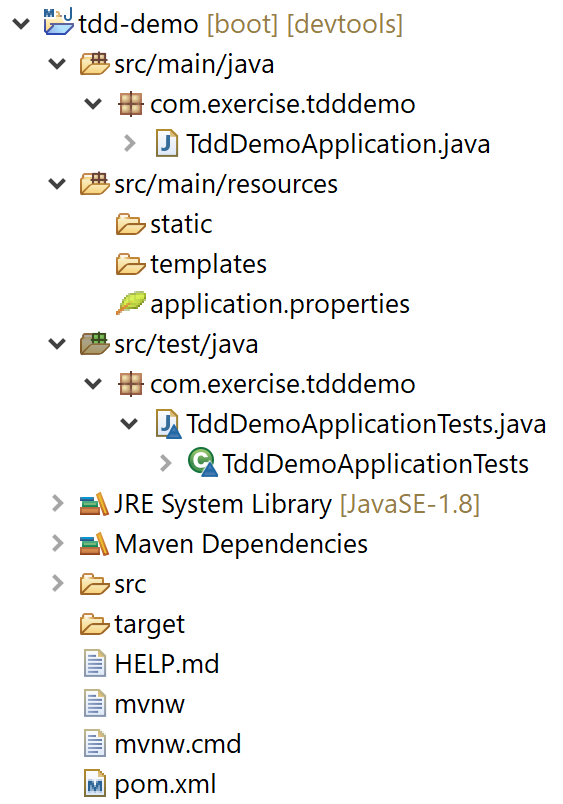
3. Start up STS, click “File” and select “Open Projects from File System…”:



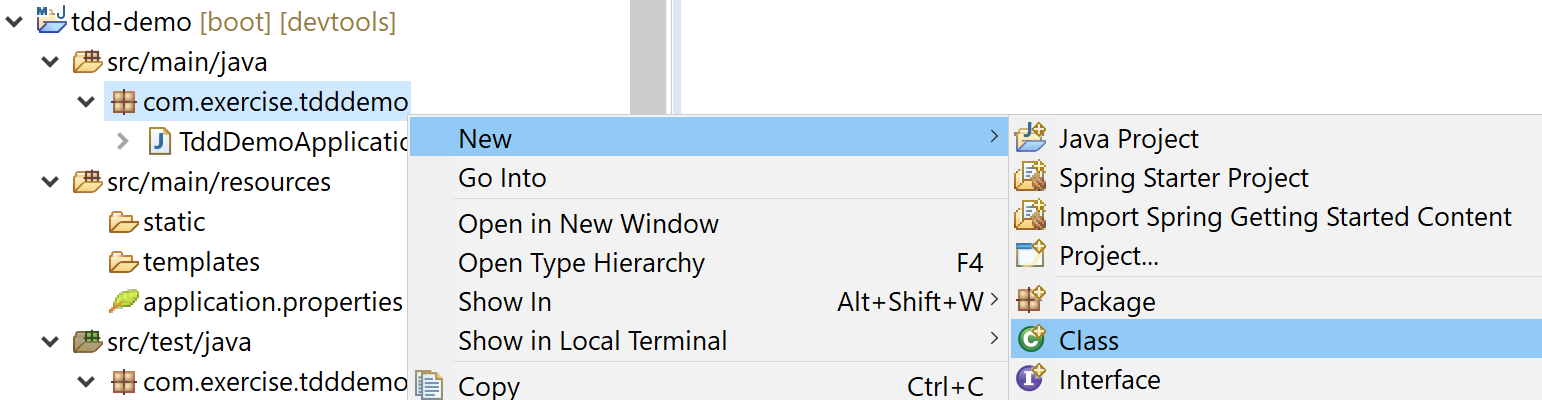
4. Type or search your project folder “tdd-demo”:

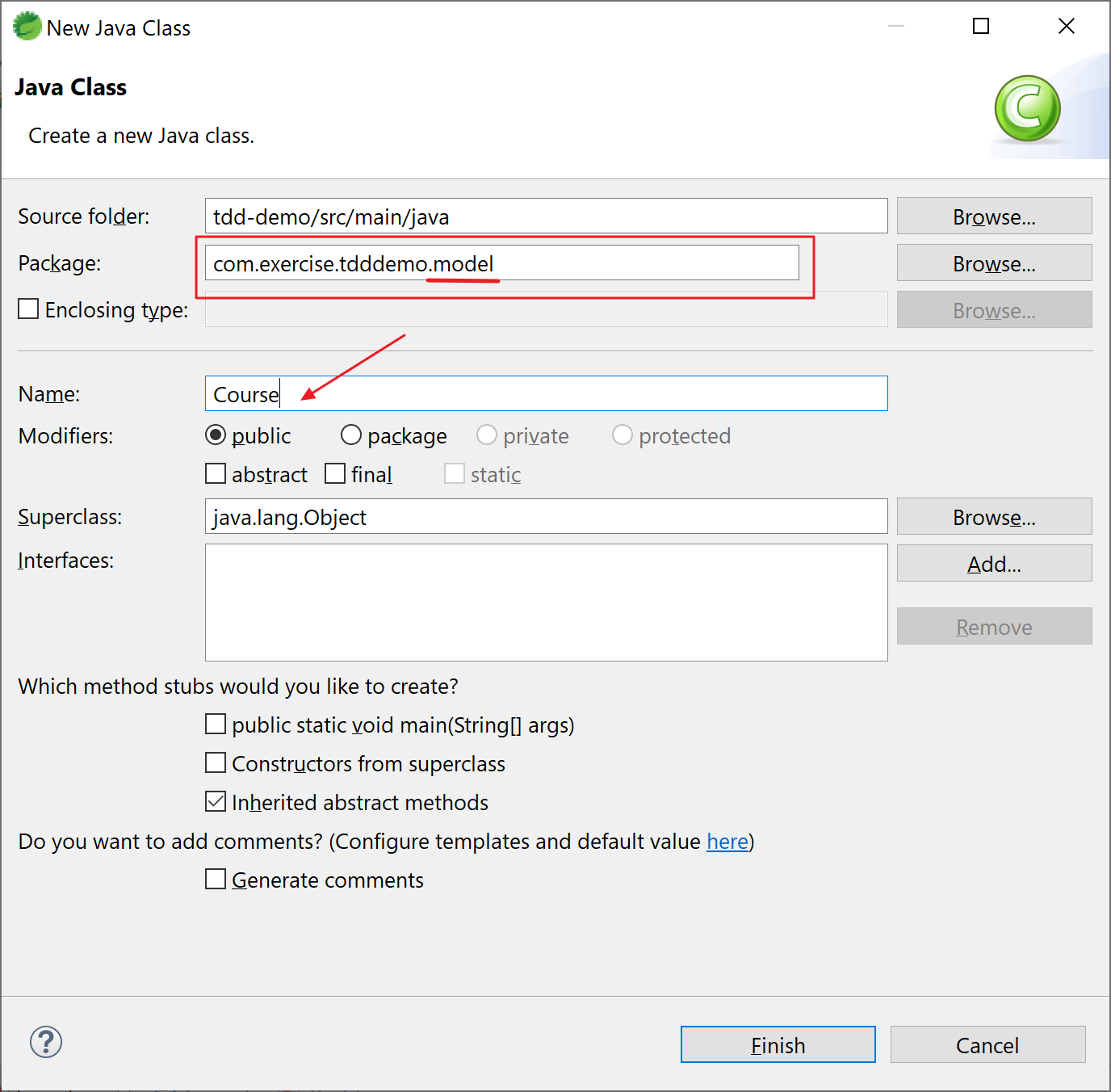


5. The project structure should be shown:



6. Create the model classes “Course” and “Student”.

6a. Create class “Course” under package “com.exercise.tdddemo.model”:



Here’s the content of “Course.java”: You can copy it to your Course.java file

package com.exercise.tdddemo.model;

import java.util.List;

public class Course {

private String id;

private String name;

private String description;

private List<String> steps;

public Course() {

super();

}

public Course(String id, String name, String description, List<String> steps) {

super();

this.id = id;

this.name = name;

this.description = description;

this.steps = steps;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getDescription() {

return description;

}

public String getName() {

return name;

}

public List<String> getSteps() {

return steps;

}

@Override

public String toString() {

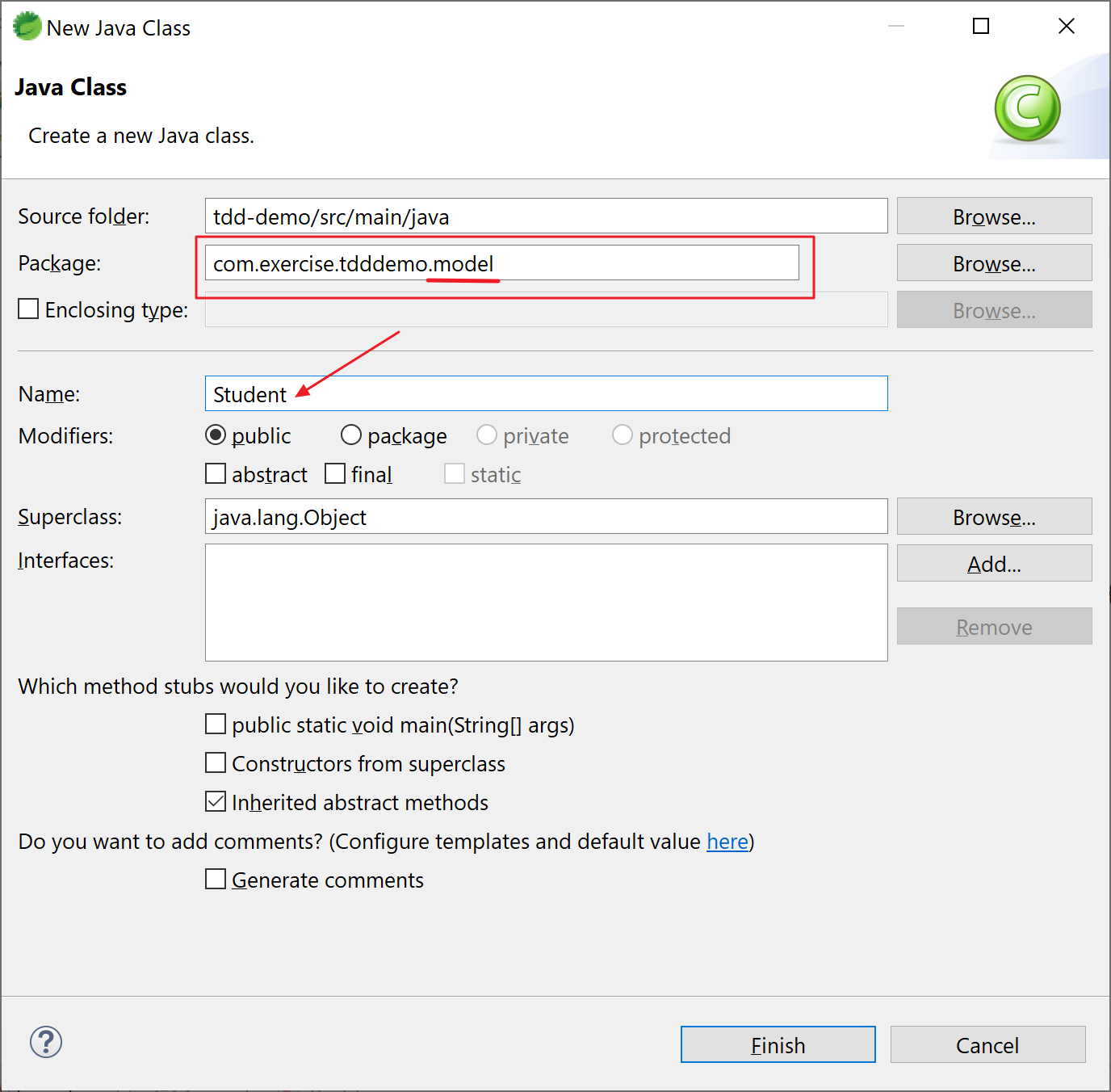
return String.format("Course [id=%s, name=%s, description=%s, steps=%s]",

id, name, description, steps);

}

}

6a. Create class “Student” under package “com.exercise.tdddemo.model”:



Here’s the content of “Student.java”:

package com.exercise.tdddemo.model;

import java.util.List;

public class Student {

private String id;

private String name;

private String description;

private List<Course> courses;

public Student() {

super();

}

public Student(String id, String name, String description, List<Course> courses) {

super();

this.id = id;

this.name = name;

this.description = description;

this.courses = courses;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public List<Course> getCourses() {

return courses;

}

public void setCourses(List<Course> courses) {

this.courses = courses;

}

@Override

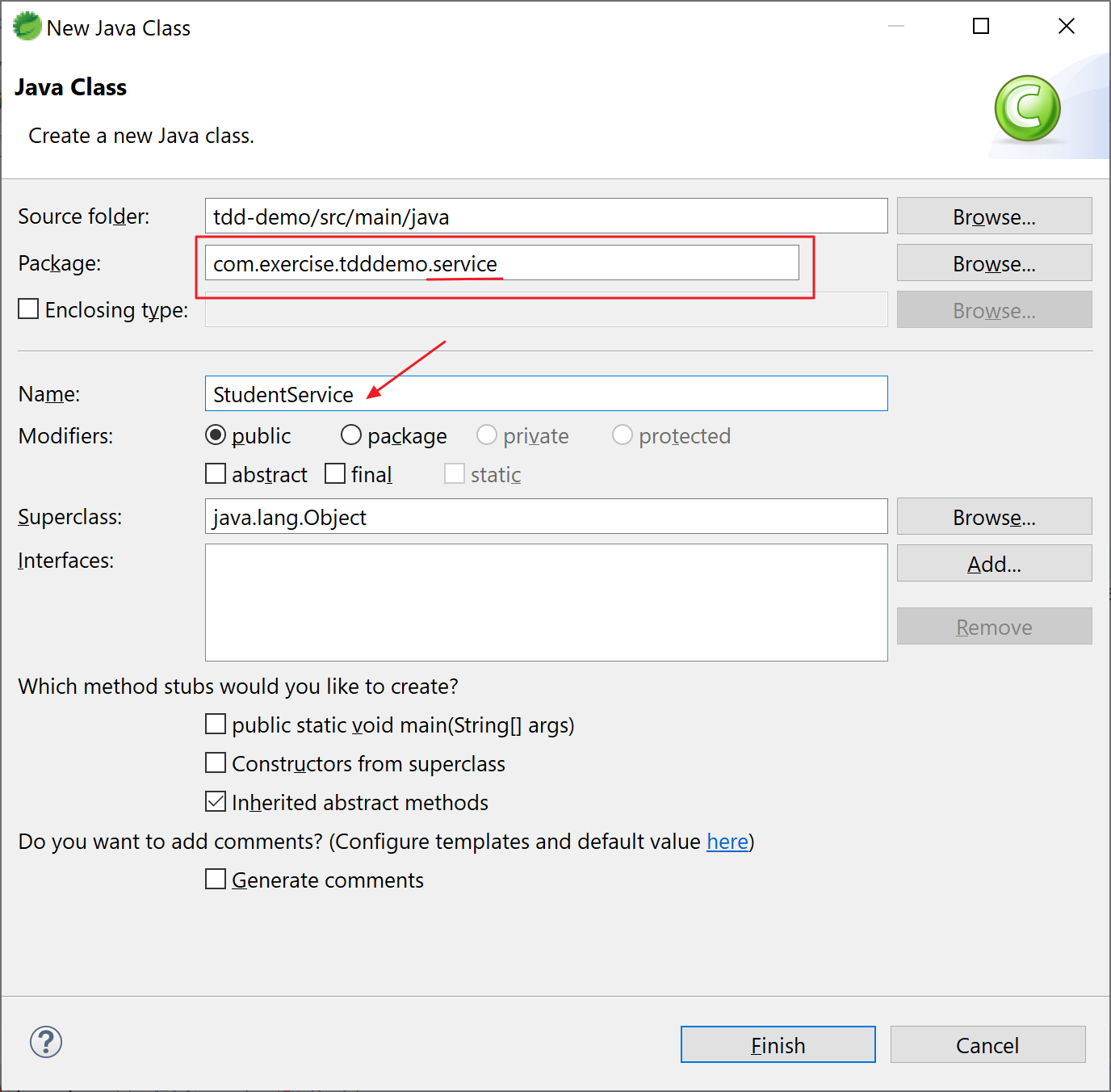
public String toString() {

return String.format("Student [id=%s, name=%s, description=%s, courses=%s]", id, name, description, courses);

}

}

7. Create the service class “StudentService” under package “com.exercise.tdddemo.service”:

Here’s the content of “StudentService.java”:

package com.exercise.tdddemo.service;

import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import org.springframework.stereotype.Component;

import com.exercise.tdddemo.model.Course;

import com.exercise.tdddemo.model.Student;

@Component

public class StudentService {

private static List<Student> students = new ArrayList<>();

static {

// Initialize Data

Course course1 = new Course("Course1", "Spring", "10 Steps",

Arrays.asList("Learn Maven", "Import Project", "First Example", "Second Example"));

Course course2 = new Course("Course2", "Spring MVC", "10 Examples",

Arrays.asList("Learn Maven", "Import Project", "First Example", "Second Example"));

Course course3 = new Course("Course3", "Spring Boot", "6K Students",

Arrays.asList("Learn Maven", "Learn Spring", "Learn Spring MVC", "First Example", "Second Example"));

Course course4 = new Course("Course4", "Maven", "Most popular maven course on internet!",

Arrays.asList("Pom.xml", "Build Life Cycle", "Parent POM", "Importing into Eclipse"));

Student alan = new Student("Student1", "Alan", "Hello, Alan",

new ArrayList<>(Arrays.asList(course1, course2)));

Student bonnie = new Student("Student2", "Bonnie", "Hi, Bonnie",

new ArrayList<>(Arrays.asList(course1, course3)));

students.add(alan);

students.add(bonnie);

}

public List<Student> retrieveAllStudents() {

return students;

}

public Student retrieveStudent(String studentId) {

for (Student student : students) {

if (student.getId().equals(studentId)) {

return student;

}

}

return null;

}

public List<Course> retrieveCourses(String studentId) {

Student student = retrieveStudent(studentId);

if (studentId.equalsIgnoreCase("Student1")) {

throw new RuntimeException("Something went wrong");

}

if (student == null) {

return null;

}

return student.getCourses();

}

public Course retrieveCourse(String studentId, String courseId) {

Student student = retrieveStudent(studentId);

if (student == null) {

return null;

}

for (Course course : student.getCourses()) {

if (course.getId().equals(courseId)) {

return course;

}

}

return null;

}

private SecureRandom random = new SecureRandom();

public Course addCourse(String studentId, Course course) {

Student student = retrieveStudent(studentId);

if (student == null) {

return null;

}

String randomId = new BigInteger(130, random).toString(32);

course.setId(randomId);

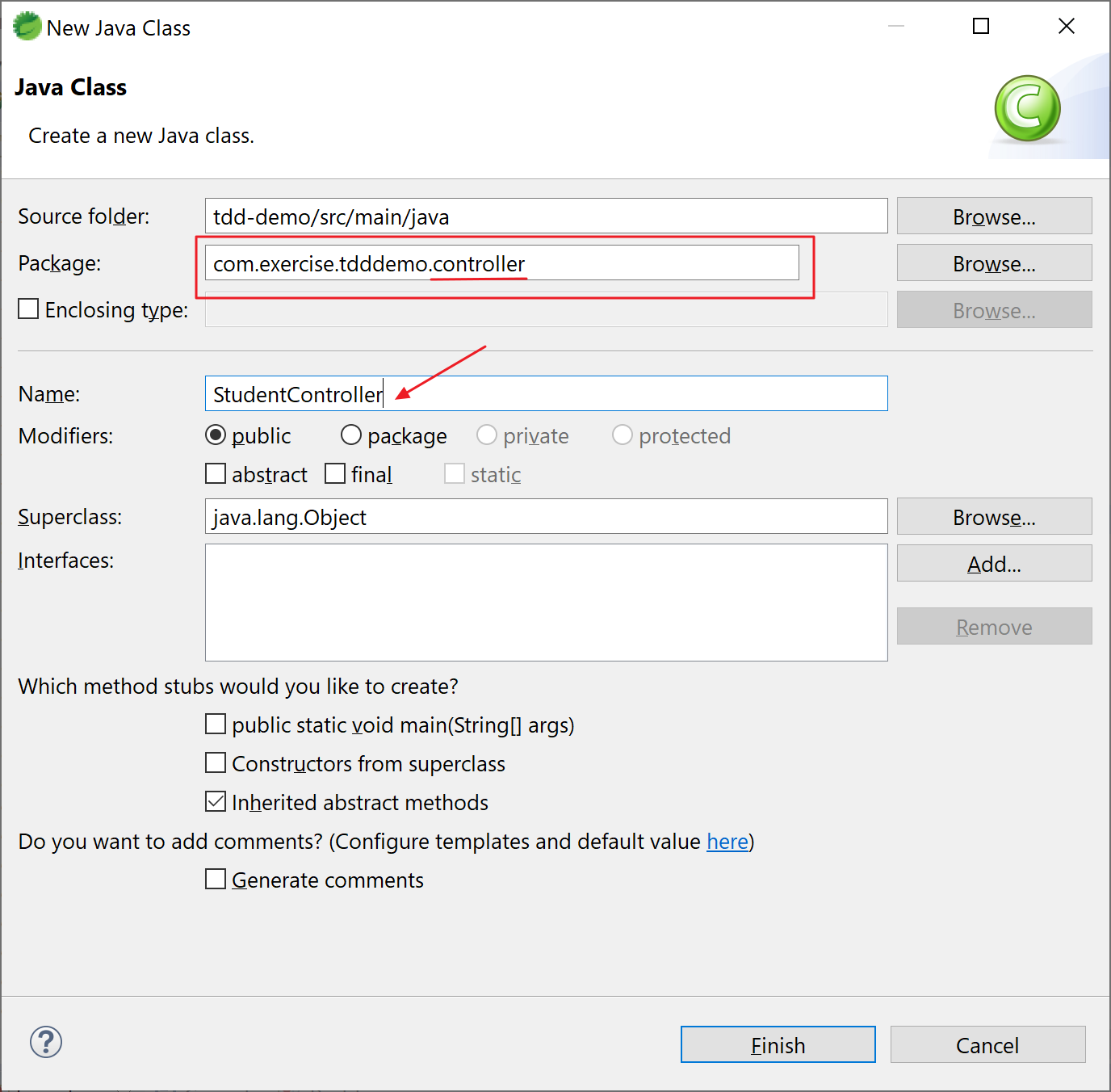
student.getCourses().add(course);

return course;

}

}

8. Create the service class “StudentController” under package “com.exercise.tdddemo.controller”:



Here’s the content of “StudentController.java”:

package com.exercise.tdddemo.controller;

import java.net.URI;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.exercise.tdddemo.model.Course;

import com.exercise.tdddemo.service.StudentService;

@RestController

public class StudentController {

@Autowired

private StudentService studentService;

@GetMapping("/students/{studentId}/courses")

public List<Course> retrieveCoursesForStudent(@PathVariable String studentId) {

return studentService.retrieveCourses(studentId);

}

@PostMapping("/students/{studentId}/courses")

public ResponseEntity<Void> registerStudentForCourse(

@PathVariable String studentId,

@RequestBody Course newCourse) {

Course course = studentService.addCourse(studentId, newCourse);

if (course == null)

return ResponseEntity.noContent().build();

URI location = ServletUriComponentsBuilder

.fromCurrentRequest()

.path("/{id}")

.buildAndExpand(course.getId())

.toUri();

return ResponseEntity.created(location).build();

}

@GetMapping("/students/{studentId}/courses/{courseId}")

public Course retrieveDetailsForCourse(

@PathVariable String studentId,

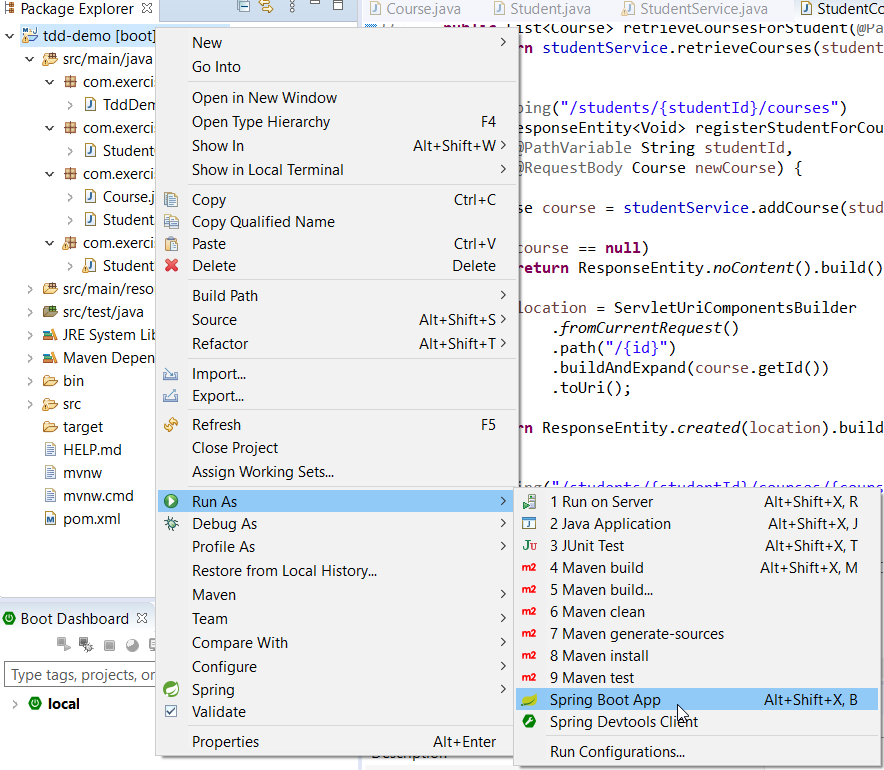
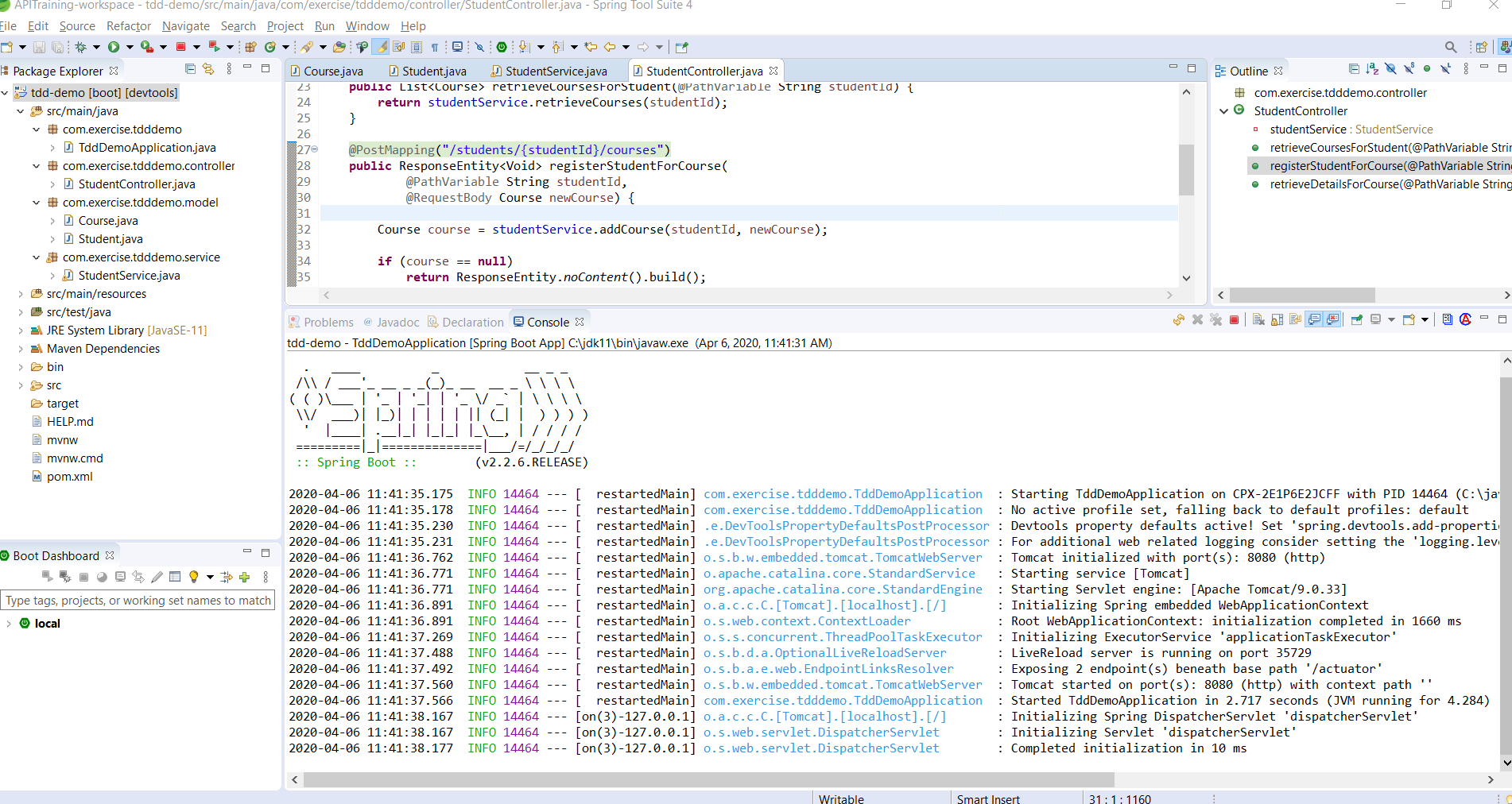
@PathVariable String courseId) {

return studentService.retrieveCourse(studentId, courseId);

}

}

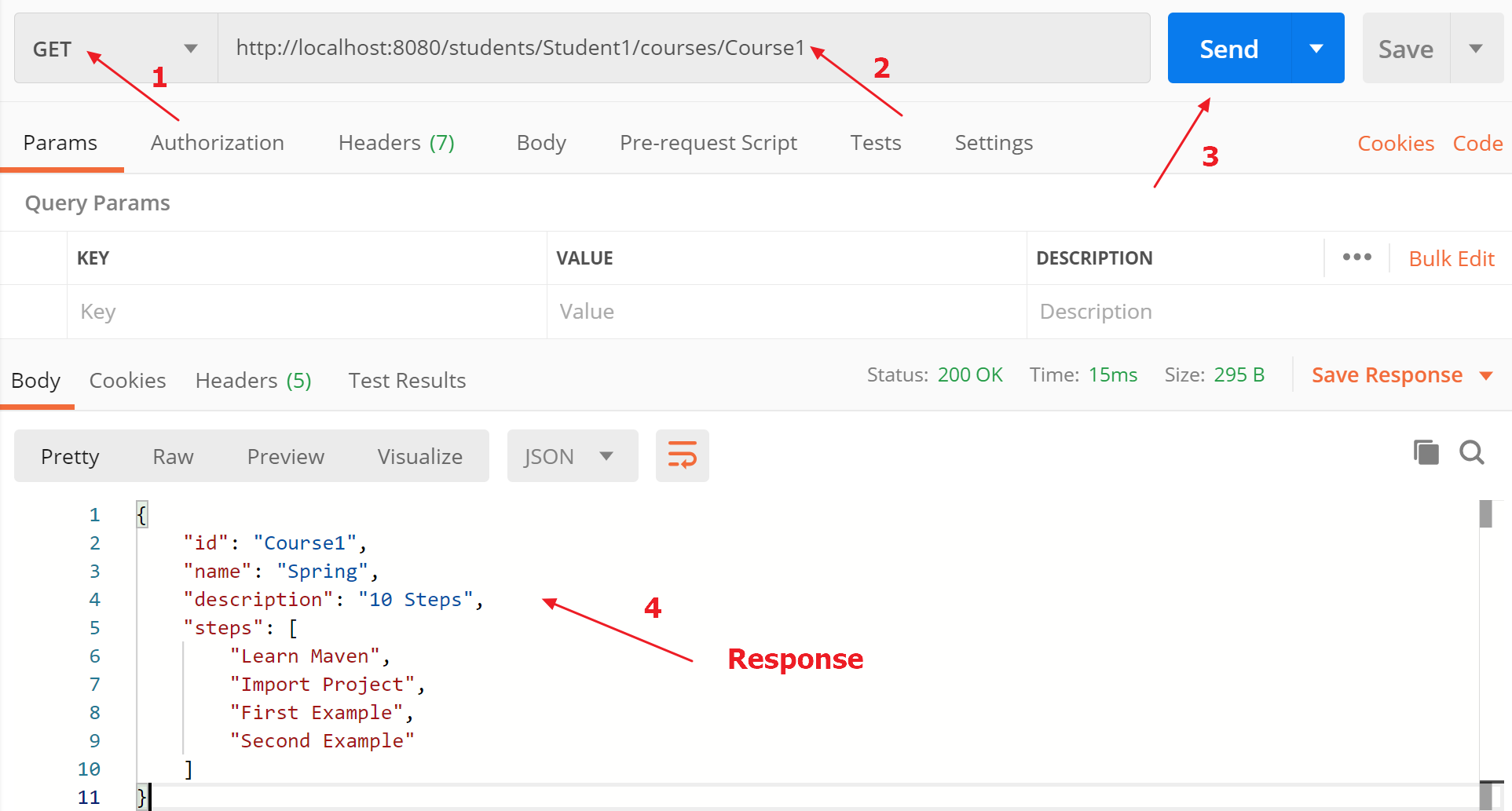
Run the Application. Right click on the project tdd-demo -> Run As -> Spring Boot App



9. Examine the APIs with Postman. Open Postman and input the following data:

GET http://localhost:8080/students/Student1/courses/Course1

Click the Send button and you should get the response:



10. After making sure that the APIs work. We’re going to write the unit tests.

Add Junit Dependency to your pom.xml

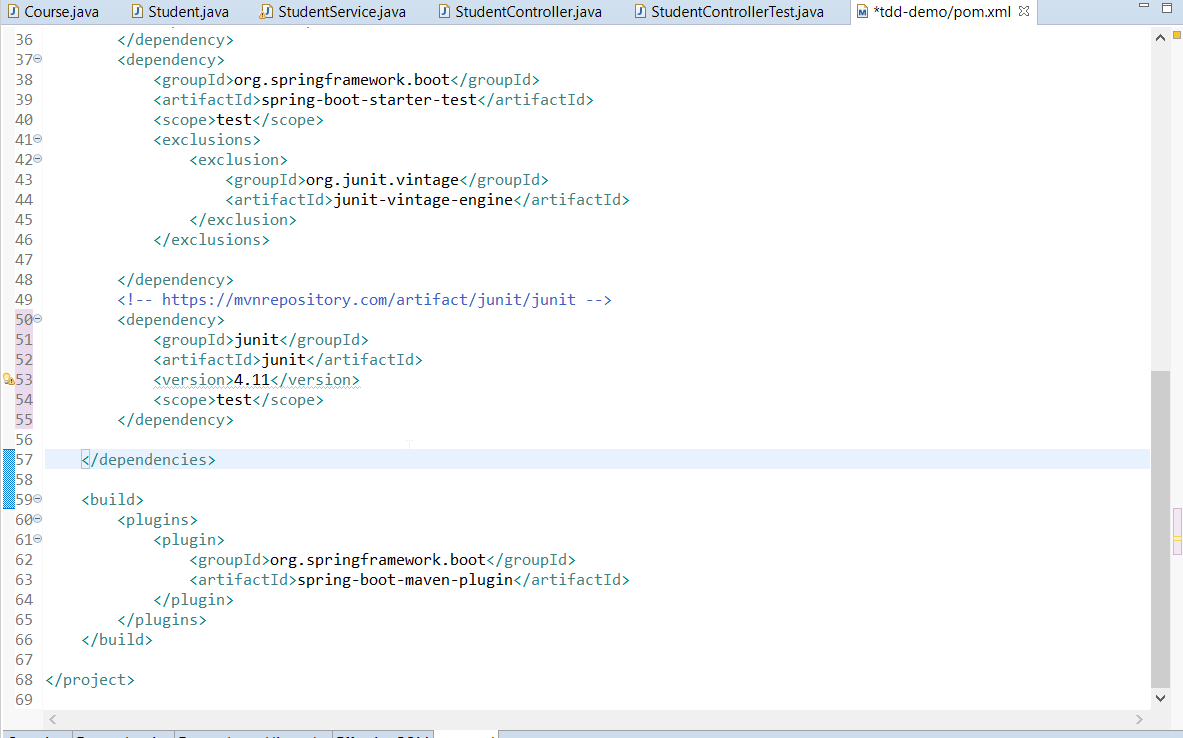
<dependency>

<groupId>junit</groupId>

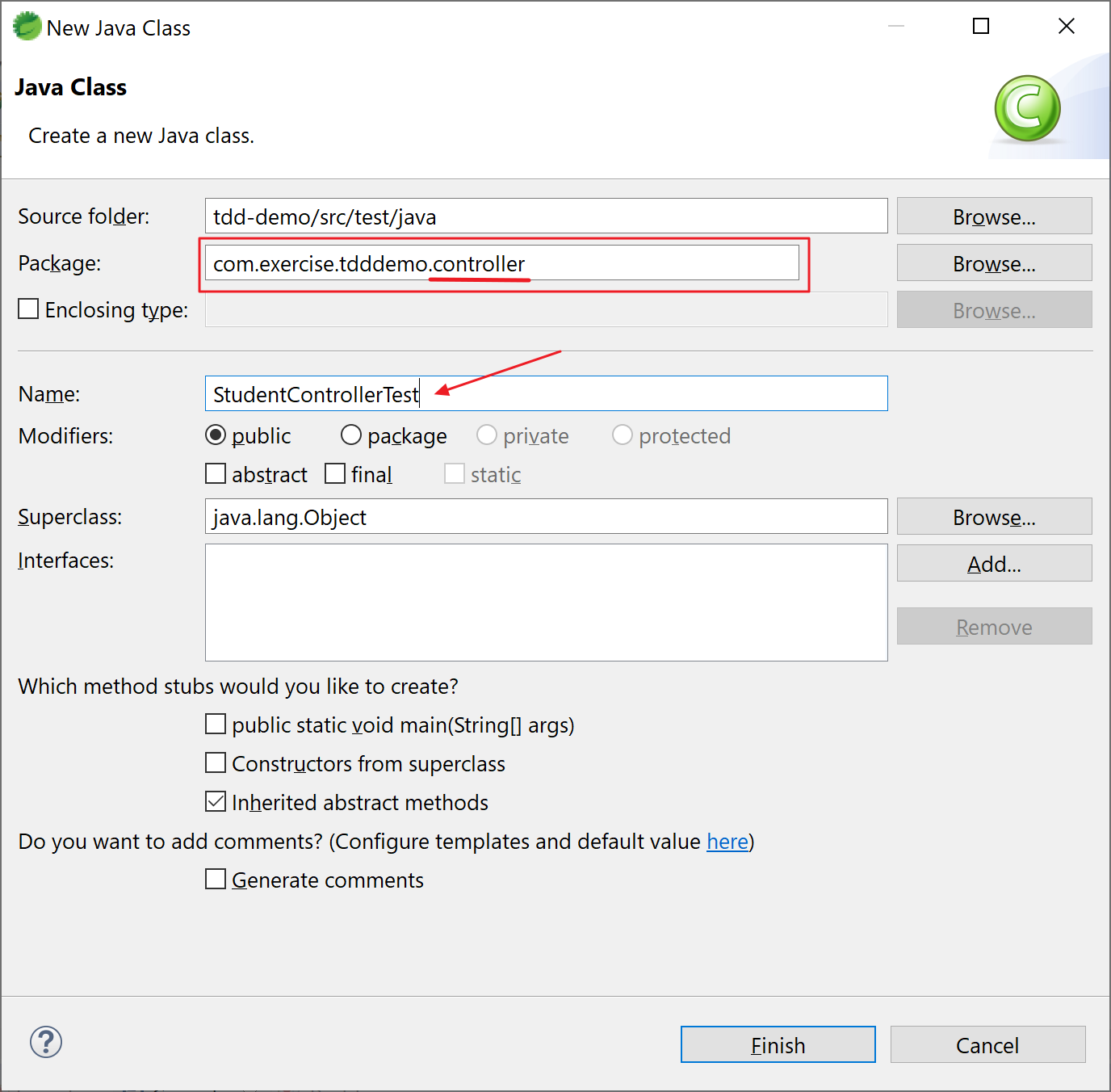
<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

11. Under “**src/test/java**”, create a new class “StudentControllerTest” under package “com.exercise.tdddemo.controller”:



Open the file and input the following content:

Here’s the content of “StudentControllerTest.java”:

package com.exercise.tdddemo.controller;

import static org.junit.Assert.assertEquals;

import java.util.Arrays;

import org.junit.jupiter.api.Test;

import org.junit.runner.RunWith;

import org.mockito.Mockito;

import org.skyscreamer.jsonassert.JSONAssert;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.mock.web.MockHttpServletResponse;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.MvcResult;

import org.springframework.test.web.servlet.RequestBuilder;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import com.exercise.tdddemo.model.Course;

import com.exercise.tdddemo.service.StudentService;

**@RunWith(SpringRunner.class)**

**@WebMvcTest(value = StudentController.class)**

public class StudentControllerTest {

@Autowired

private MockMvc mockMvc;

**@MockBean**

**private StudentService studentService;**

Course mockCourse = new Course("Course1", "Spring", "10 Steps",

Arrays.asList("Learn Maven", "Import Project", "First Example", "Second Example"));

String exampleCourseJson = "{\"name\":\"Spring\",\"description\":\"10 Steps\",\"steps\":[\"Learn Maven\",\"Import Project\",\"First Example\",\"Second Example\"]}";

@Test

public void whenStudent1Exist\_thenReturnCourses() throws Exception {

**Mockito.when(studentService.retrieveCourse(Mockito.anyString(), Mockito.anyString())).thenReturn(mockCourse);**

RequestBuilder requestBuilder = MockMvcRequestBuilders.get("/students/Student1/courses/Course1")

.accept(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

System.out.println(result.getResponse());

String expected = "{\"id\":\"Course1\",\"name\":\"Spring\",\"description\":\"10 Steps\",\"steps\":[\"Learn Maven\",\"Import Project\",\"First Example\",\"Second Example\"]}";

**JSONAssert.assertEquals(expected, result.getResponse().getContentAsString(), false);**

}

@Test

public void whenStudent1Exist\_thenNewCourseCreate() throws Exception {

Course mockCourse = new Course("1", "Smallest Number", "1", Arrays.asList("1", "2", "3", "4"));

// studentService.addCourse to respond back with mockCourse

Mockito.when(studentService.addCourse(Mockito.anyString(), Mockito.any(Course.class))).thenReturn(mockCourse);

// Send course as body to /students/Student1/courses

RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson).contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus());

assertEquals("http://localhost/students/Student1/courses/1", response.getHeader(HttpHeaders.LOCATION));

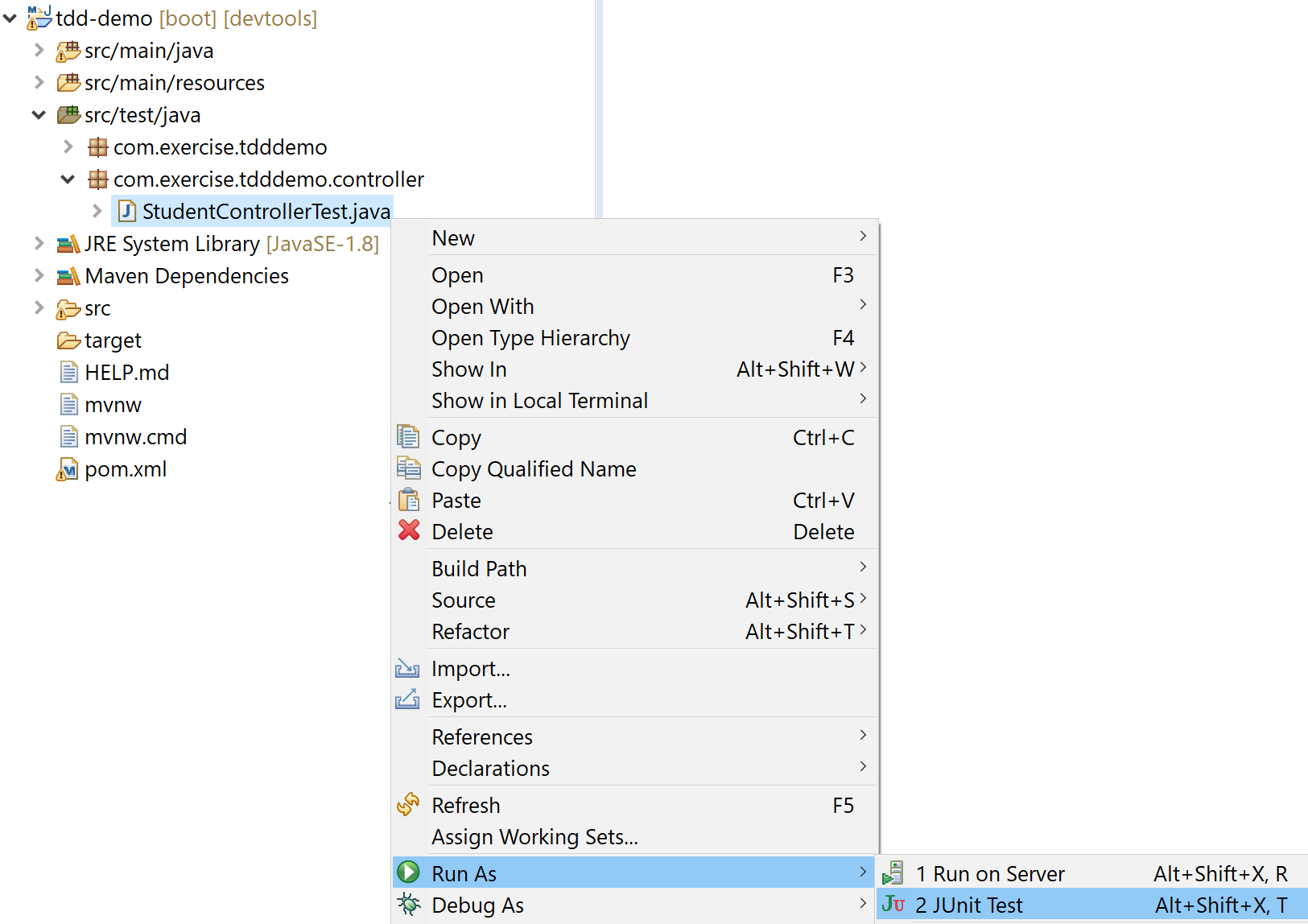
}

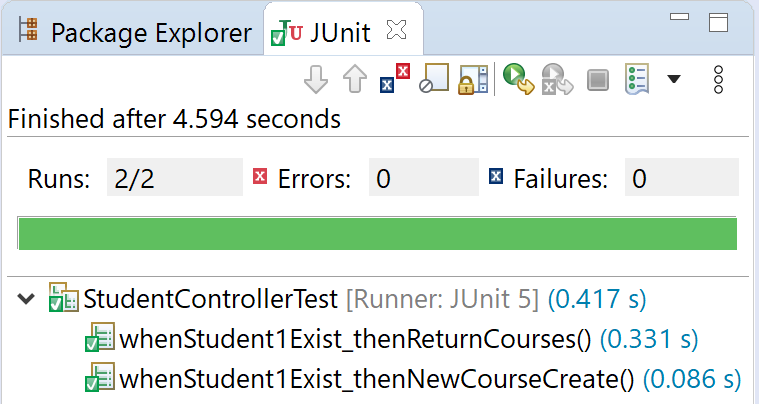
}

Note:

1. We annotate the tests by using @RunWith (SpringRunner is an alias for the SpringJUnit4ClassRunner)
2. Annotation @WebMvcTest is used to fire up an application context that contains only the beans needed for testing a web controller
3. Annotation @MockBean is used to mock away the business logic. It automatically replaces the bean of the same type in the application context with a Mockito mock.
4. Mockito allows you to create and configure mock objects. Using Mockito greatly simplifies the development of tests for classes with external dependencies.
5. Function assertEquals is for validating the results.

12. Run your test cases.

12a. In STS, right click your test class, select “Run As” and click “Junit Test”:

After the tests running, you should get the results:

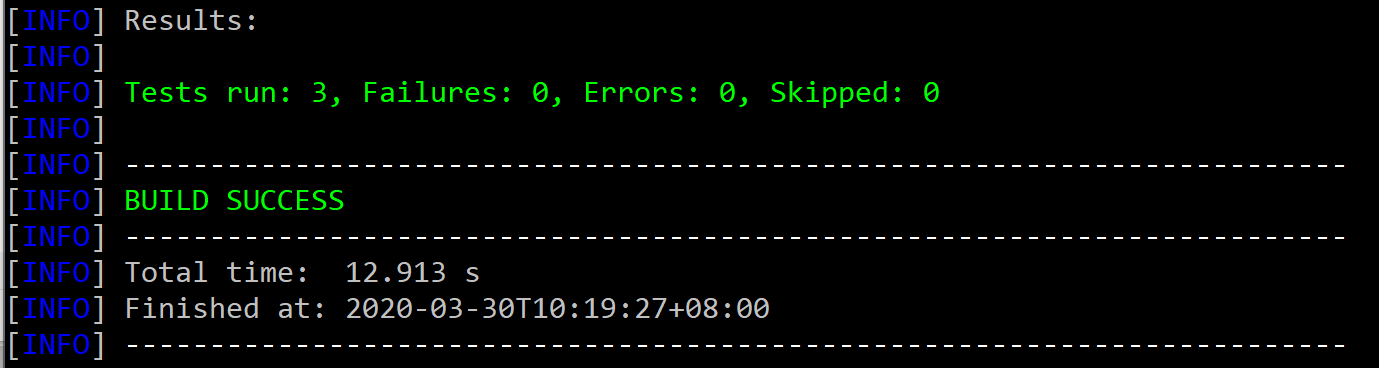
12b. In command prompt, change directory to your project root, e.g.:

C:\>cd C:\java-ex\tdd-demo

Type and run the following command:

C:\java-ex\tdd-demo>mvn test

And you should get the following results:

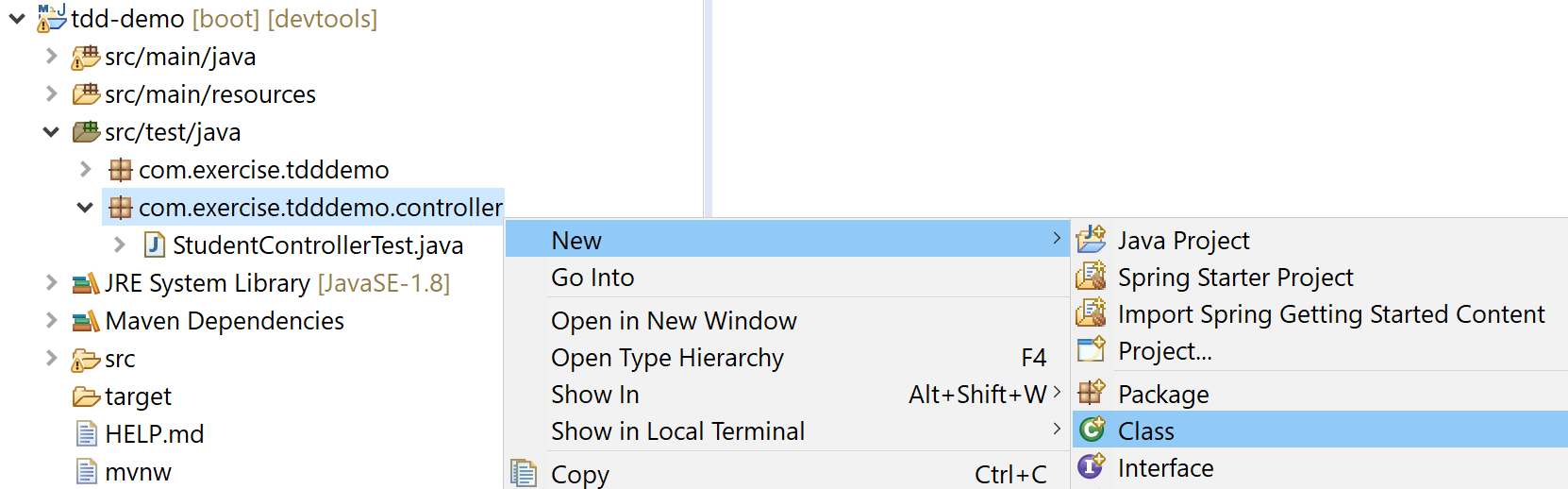
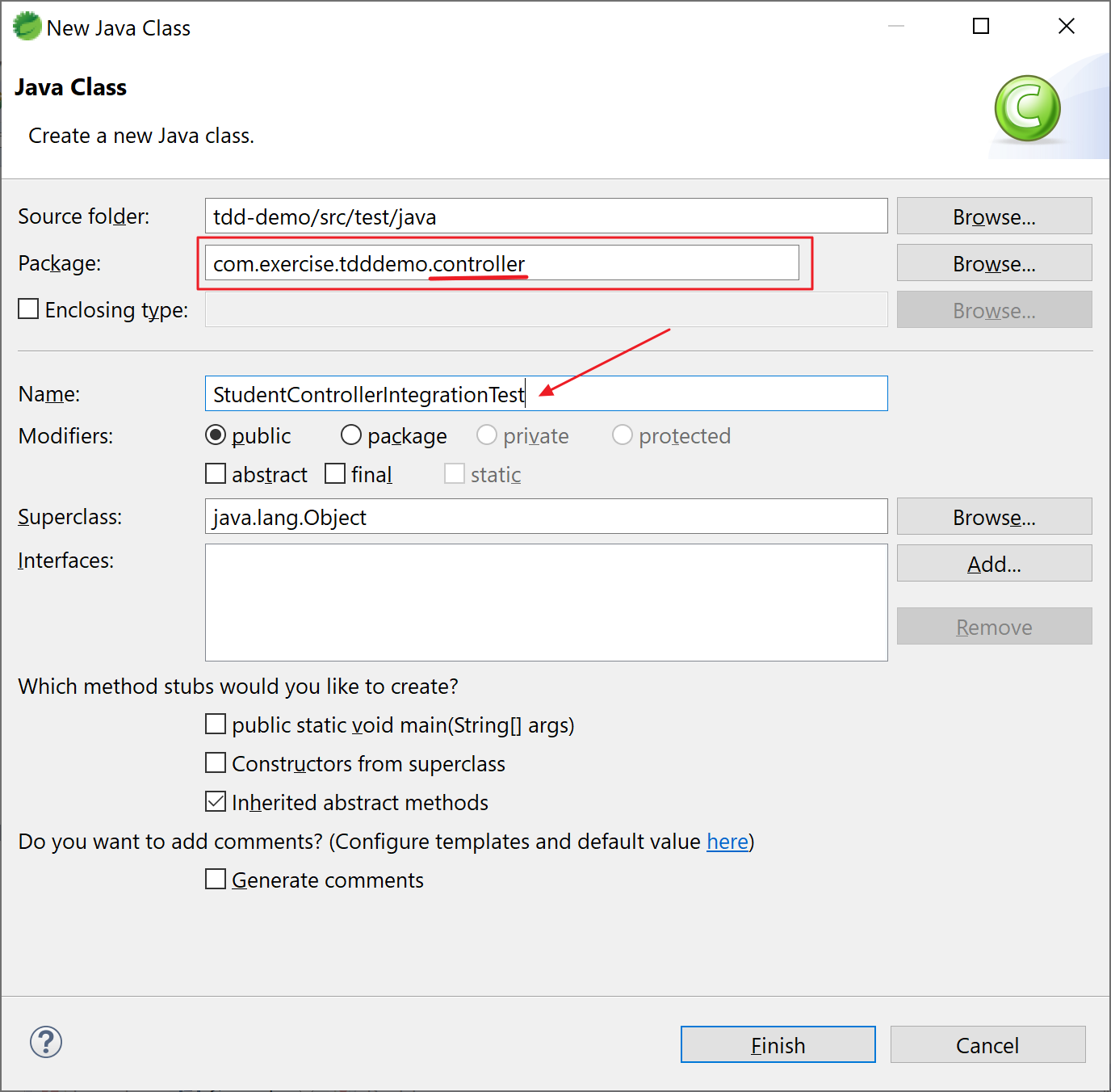


Note: we have one empty test in “com.exercise.tdddemo.TddDemoApplicationTests.java” plus the two tests added in the above steps. So there are 3 tests in total.

13. In the previous step, we did the unit tests (making sure each function/unit of code works as expected). Now we’re going to write the integration test.

Note: Integration testing basically means having automatic tests that run against your entire app. So you're not simply testing individual functions or classes (as you would in unit tests) but (in the context of a REST API) your tests make HTTP requests against a running web server, look at the response to your request.

13a. Under “**src/test/java**”, create a new class “StudentControllerIntegrationTest” under package “com.exercise.tdddemo.controller”:



Open the file and input the following content:

Here’s the content of “StudentControllerIntegrationTest.java”:

package com.exercise.tdddemo.controller;

import static org.junit.Assert.assertTrue;

import java.util.Arrays;

import org.junit.jupiter.api.Test;

import org.junit.runner.RunWith;

import org.skyscreamer.jsonassert.JSONAssert;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.web.client.TestRestTemplate;

import org.springframework.boot.web.server.LocalServerPort;

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

import org.springframework.http.ResponseEntity;

import org.springframework.test.context.junit4.SpringRunner;

import com.exercise.tdddemo.TddDemoApplication;

import com.exercise.tdddemo.model.Course;

@RunWith(SpringRunner.class)

**@SpringBootTest(classes = TddDemoApplication.class, webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)**

public class StudentControllerIntegrationTest {

@LocalServerPort

private int port;

TestRestTemplate restTemplate = new TestRestTemplate();

HttpHeaders headers = new HttpHeaders();

@Test

public void retrieveStudentCourseTest() throws Exception {

HttpEntity<String> entity = new HttpEntity<String>(null, headers);

**ResponseEntity<String> response = restTemplate.exchange(**

**createURLWithPort("/students/Student1/courses/Course1"),**

**HttpMethod.GET, entity, String.class);**

String expected = "{\"id\":\"Course1\",\"name\":\"Spring\",\"description\":\"10 Steps\",\"steps\":[\"Learn Maven\",\"Import Project\",\"First Example\",\"Second Example\"]}";

**JSONAssert.assertEquals(expected, response.getBody(), true);**

}

@Test

public void addCourseTest() {

Course course = new Course("Course1", "Spring", "10 Steps", Arrays

.asList("Learn Maven", "Import Project", "First Example", "Second Example"));

HttpEntity<Course> entity = new HttpEntity<Course>(course, headers);

ResponseEntity<String> response = restTemplate.exchange(createURLWithPort("/students/Student1/courses"),

HttpMethod.POST, entity, String.class);

String actual = response.getHeaders().get(HttpHeaders.LOCATION).get(0);

assertTrue(actual.contains("/students/Student1/courses/"));

}

private String createURLWithPort(String uri) {

return "http://localhost:" + port + uri;

}

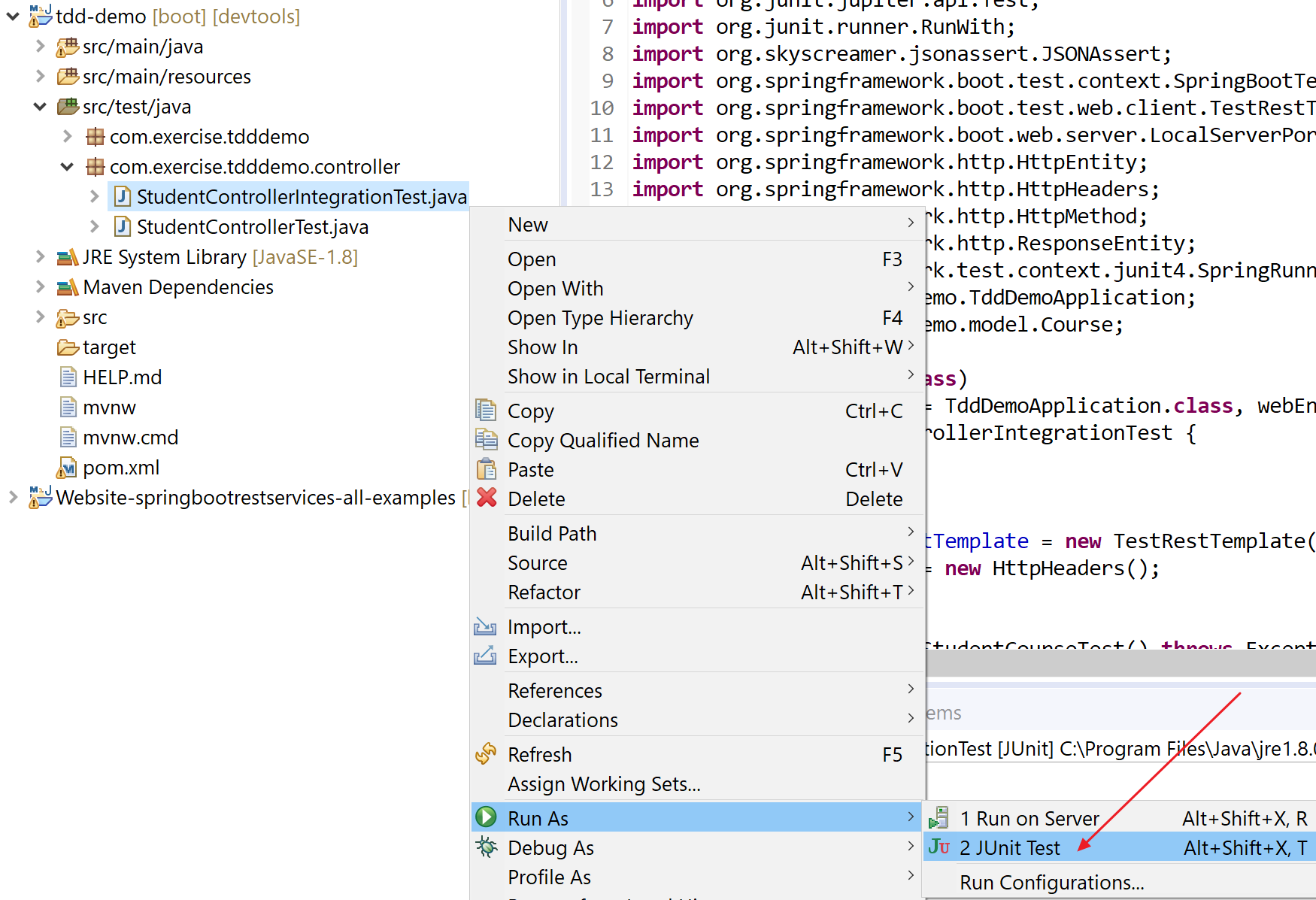
}

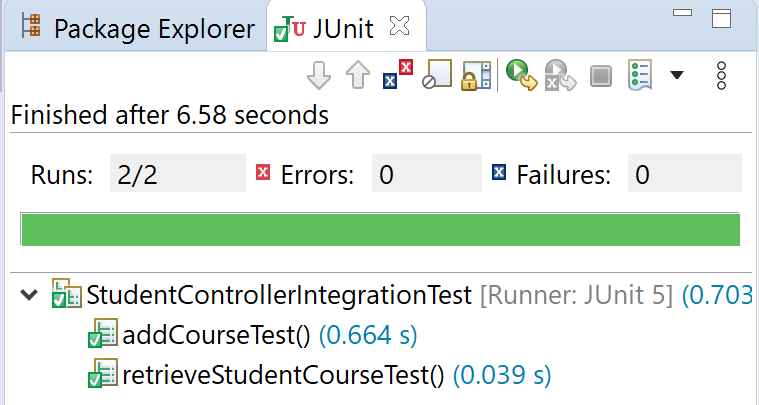
Note:

1. RANDOM\_PORT is for restTemplate to use port randomly
2. restTemplate is used to send request to and retrieve the response from the APIs
3. assertEquals is used to validate the results

14. Run your test cases.

14a. In STS, right click your test class, select “Run As” and click “Junit Test”:



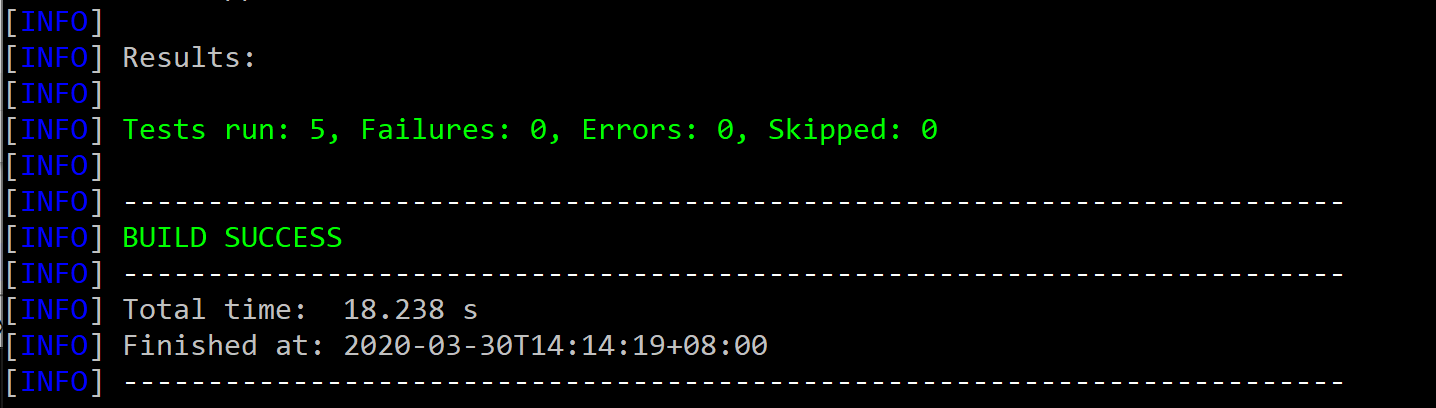
After the tests running, you should get the results:

14b. In command prompt, change directory to your project root, e.g.:

C:\>cd C:\java-ex\tdd-demo

Type and run the following command:

C:\java-ex\tdd-demo>mvn test

And you should get the following results:

Note: we have 3 tests from previous steps plus the two tests from integration tests. There are 5 tests in total.

Congratulations! Try this one in case you have problem with this exercise:

