

API TRAINING EXERCISE

TDD Ver. 1.0

Apr 2020

[generali.com](https://www.generali.com)



1 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.....	8

2 Exercise 1

The exercise will demonstrate 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)

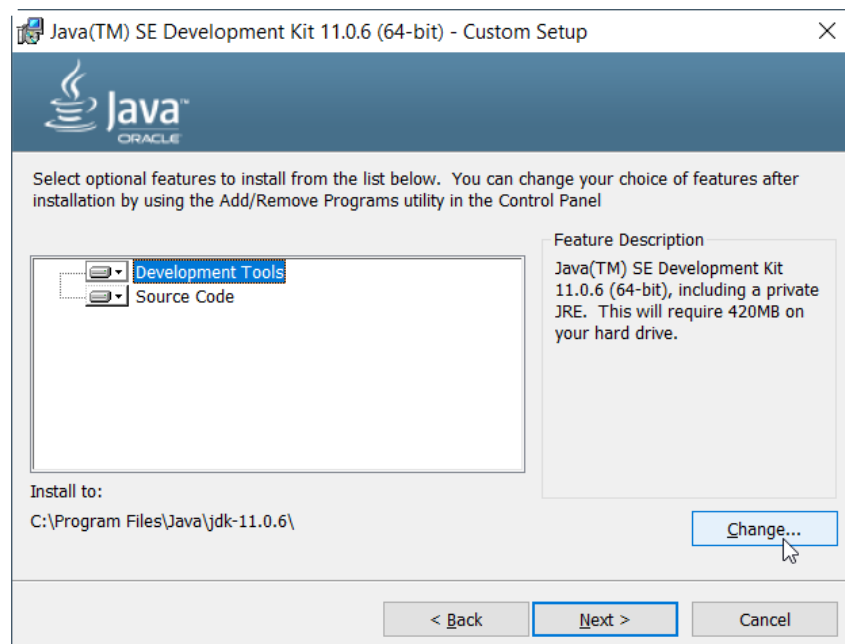
2.1 PREREQUISITE INSTALLATION

2.1.1 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.

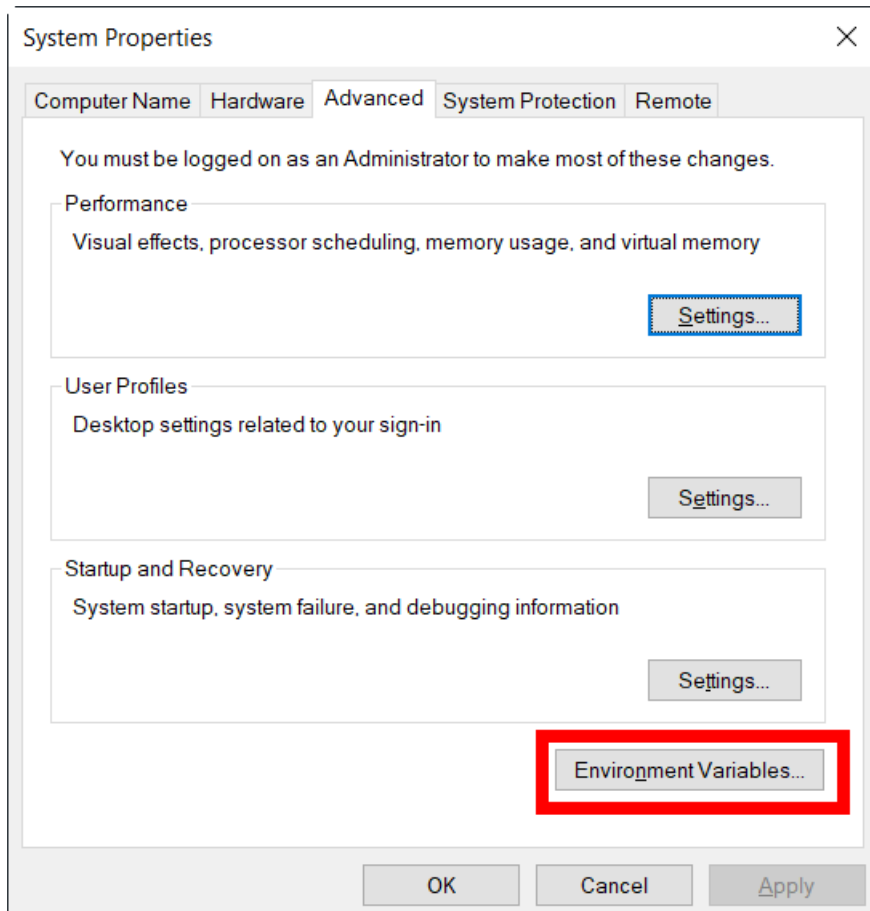
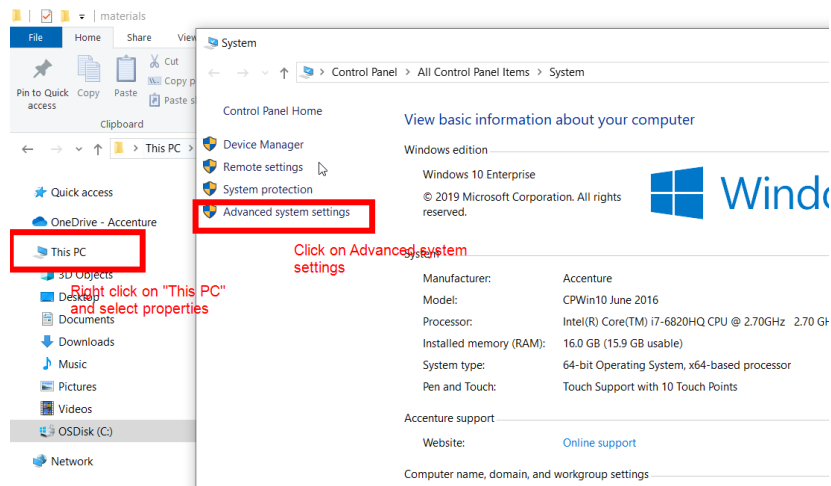
After download, double click the file and follow the installation wizard

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

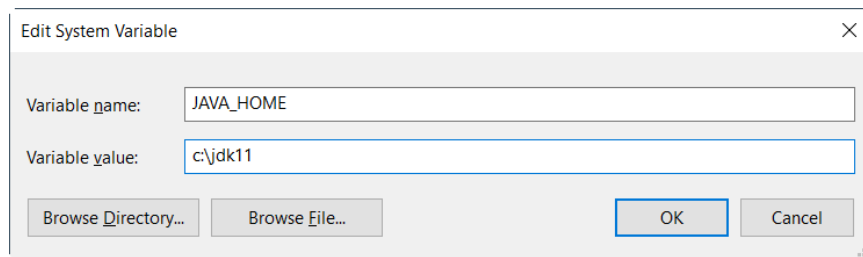


2.1.1.1 Environment Path Setup

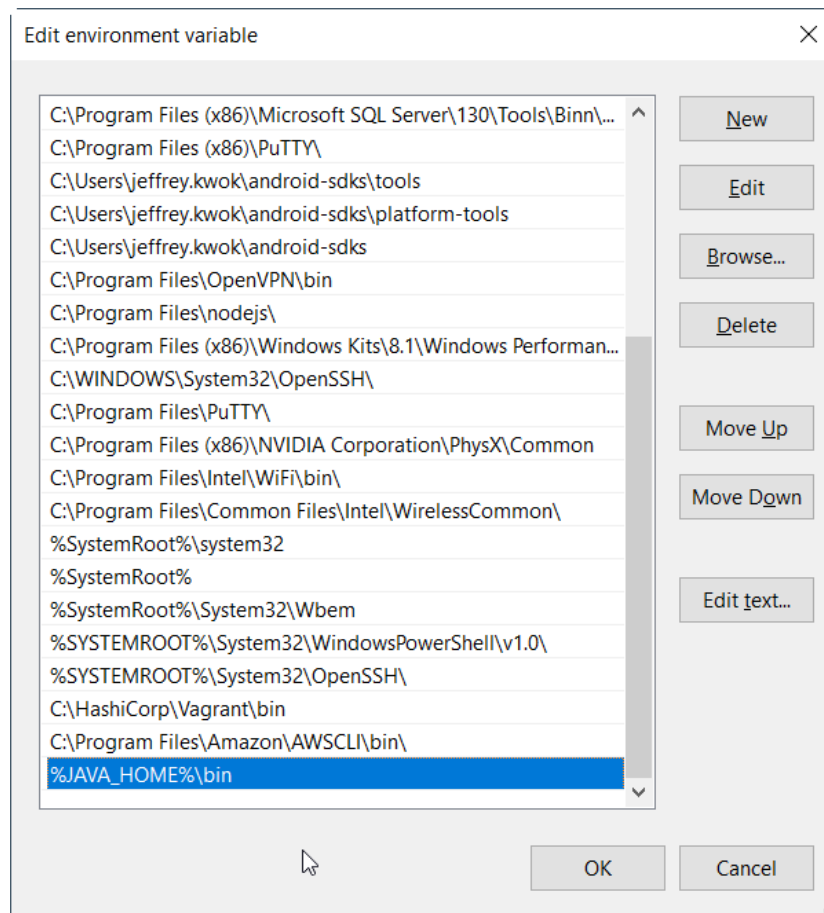
Add JAVA_HOME and JAVA_HOME/bin to your environment path



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

```
Command Prompt
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\jeffrey.kwok>java -version
java version "11.0.6" 2020-01-14 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.6+8-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.6+8-LTS, mixed mode)

C:\Users\jeffrey.kwok>
```

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

```

Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\jeffrey.kwok>java -version
java version "11.0.6" 2020-01-14 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.6+8-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.6+8-LTS, mixed mode)

C:\Users\jeffrey.kwok>echo %JAVA_HOME%
c:\jdk11

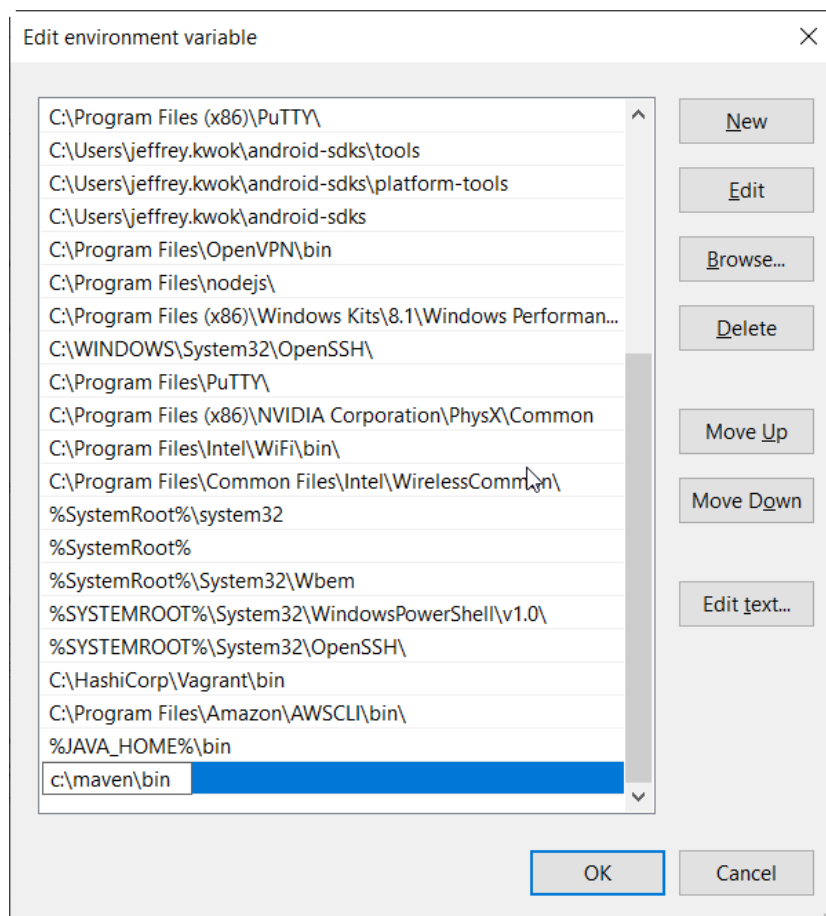
C:\Users\jeffrey.kwok>echo %PATH%
c:\jdk11\bin;C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\TortoiseSVN\bin;C:\Program Files (x86)\Microsoft SQL Server\Client SDK\ODBC\130\Tools\Binn\;C:\Program Files (x86)\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x86)\Microsoft SQL Server\130\Tools\Binn\ManagementStudio\;C:\Program Files (x86)\PuTTY\;C:\Users\jeffrey.kwok\android-sdks\tools;C:\Users\jeffrey.kwok\android-sdks\platform-tools;C:\Users\jeffrey.kwok\android-sdks;C:\Program Files\OpenVPN\bin;C:\Program Files\nodejs\;C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Toolkit\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files\PuTTY\;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;C:\HashiCorp\Vagrant\bin;C:\Program Files\Amazon\AWSCLI\bin\;c:\maven\bin;C:\Ruby24-x64\bin;C:\Program Files\Docker\Toolbox\;C:\Users\jeffrey.kwok\AppData\Local\Microsoft\WindowsApps\;C:\Users\jeffrey.kwok\AppData\Local\Roaming\Cloud Foundry\;C:\Users\jeffrey.kwok\AppData\Roaming\npm\;C:\Users\jeffrey.kwok\AppData\Local\Programs\Git\cmd
C:\Users\jeffrey.kwok>

```

2.1.2 Maven

Download Maven from <https://maven.apache.org/download.cgi> and unzip the file. E.g c:\maven

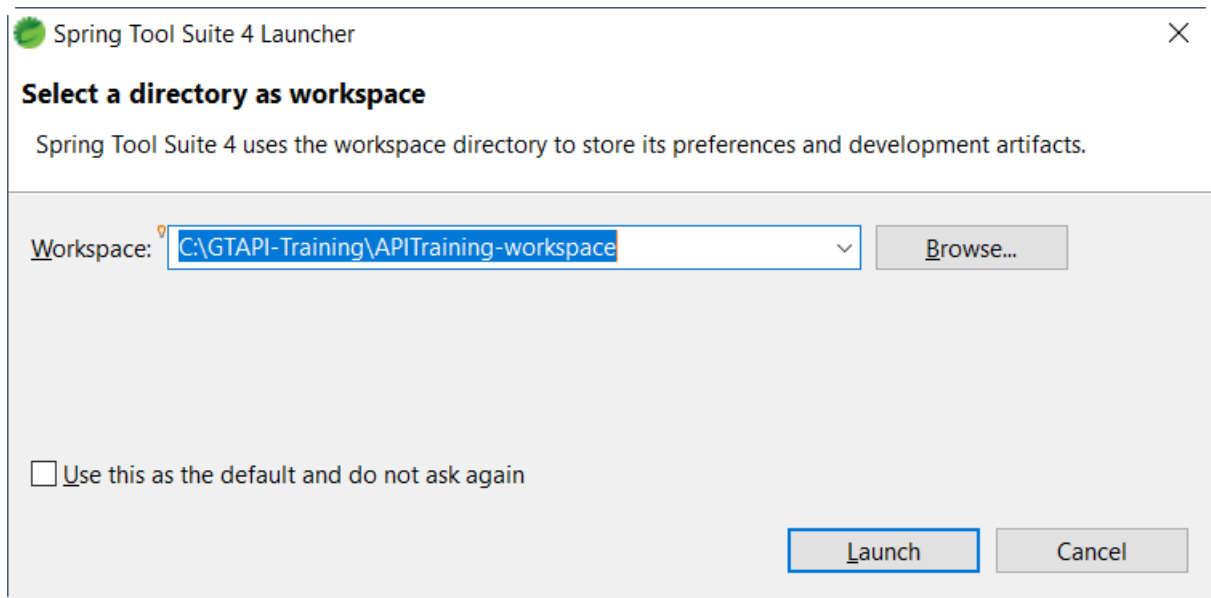
Append the Maven **bin** path to **PATH** environment variable



2.1.3 Spring Tool Suite

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

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



2.1.4 Postman

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

2.2 TDD EXERCISE

2.2.1 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:



Project

☒ Maven Project
 ☐ Gradle Project

Language

☒ Java
 ☐ Kotlin
 ☐ Groovy

Spring Boot

☐ 2.3.0 M4
 ☐ 2.3.0 (SNAPSHOT)
 ☐ 2.2.7 (SNAPSHOT)
 ☒ 2.2.6
 ☐ 2.1.14 (SNAPSHOT)
 ☐ 2.1.13

Project Metadata

Group

com.exercise

Artifact

tdd-demo

Name

tdd-demo

Description

String Boot DevTools

Package name

com.exercise.tdd-demo

Packaging

☒ Jar
 ☐ War

Java

☐ 14
 ☒ 11
 ☐ 8

Dependencies

ADD DEPENDENCIES...

CTRL + B

Spring Boot DevTools

DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Boot Actuator

OPS

Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

GENERATE

CTRL + G

EXPLORE

CTRL + SPACE

SHARE...

Or you may use this one:



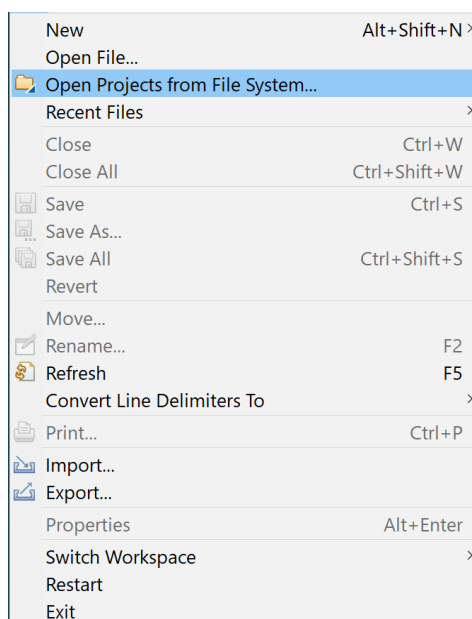
tdd-demo_empty.zip

```

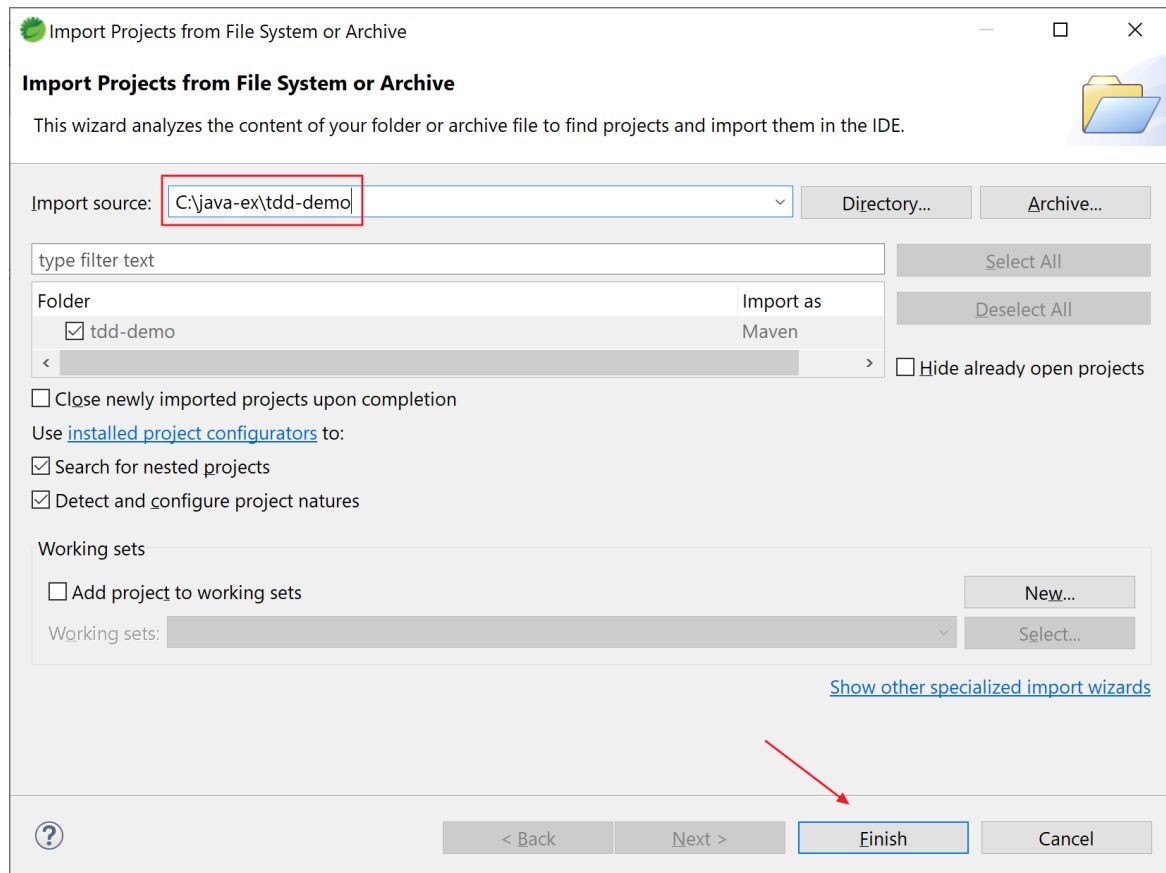
C:\JAVA-EX\TDD-DEMO
|
| .gitignore
| HELP.md
| mvnw
| mvnw.cmd
| pom.xml
|
|---.mvn
|   |---wrapper
|       maven-wrapper.jar
|       maven-wrapper.properties
|       MavenWrapperDownloader.java
|
|---src
|   |---main
|       |---java
|           |---com
|               |---exercise
|                   |---tdddemo
|                       TddDemoApplication.java
|
|       |---resources
|           application.properties
  
```

```
|
|_ test
|   |_ java
|       |_ com
|           |_ exercise
|               |_ tdddemo
|                   TddDemoApplicationTests.java
```

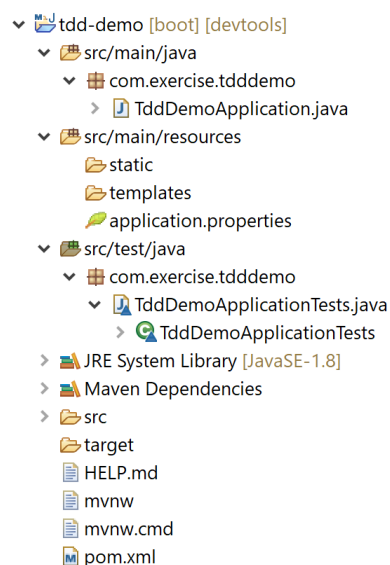
3. Start up STS, click “File” and select “Open Projects from File System...”:



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

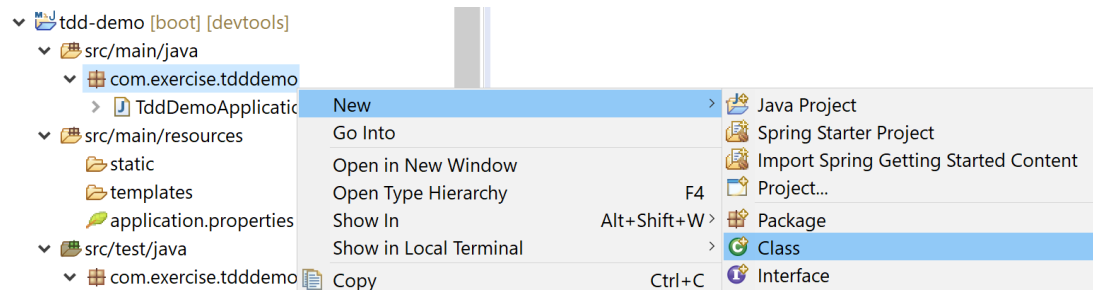
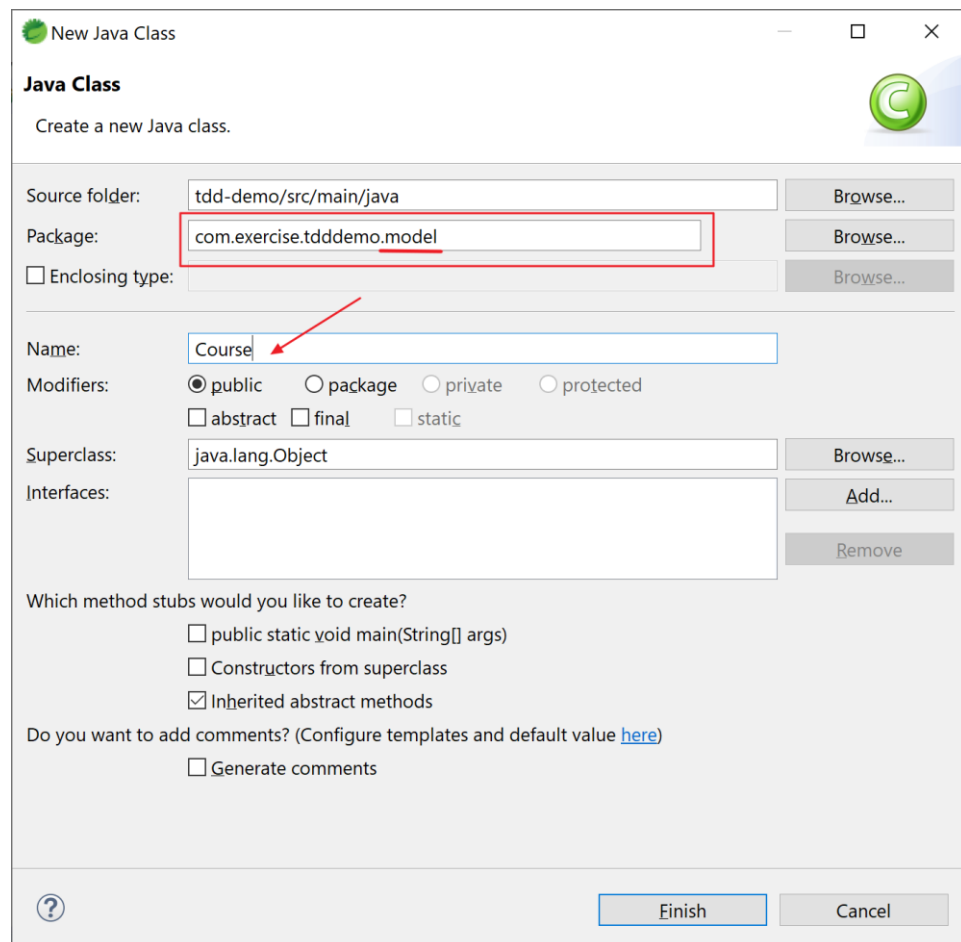


5. The project structure should be shown:



6. Create the model classes "Course" and "Student".

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

New Java Class

Create a new Java class.

Source folder: tdd-demo/src/main/java Browse...

Package: com.exercise.tdddemo.model Browse...

☐ Enclosing type: Browse...

Name: Course

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

? Finish Cancel

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;
    }

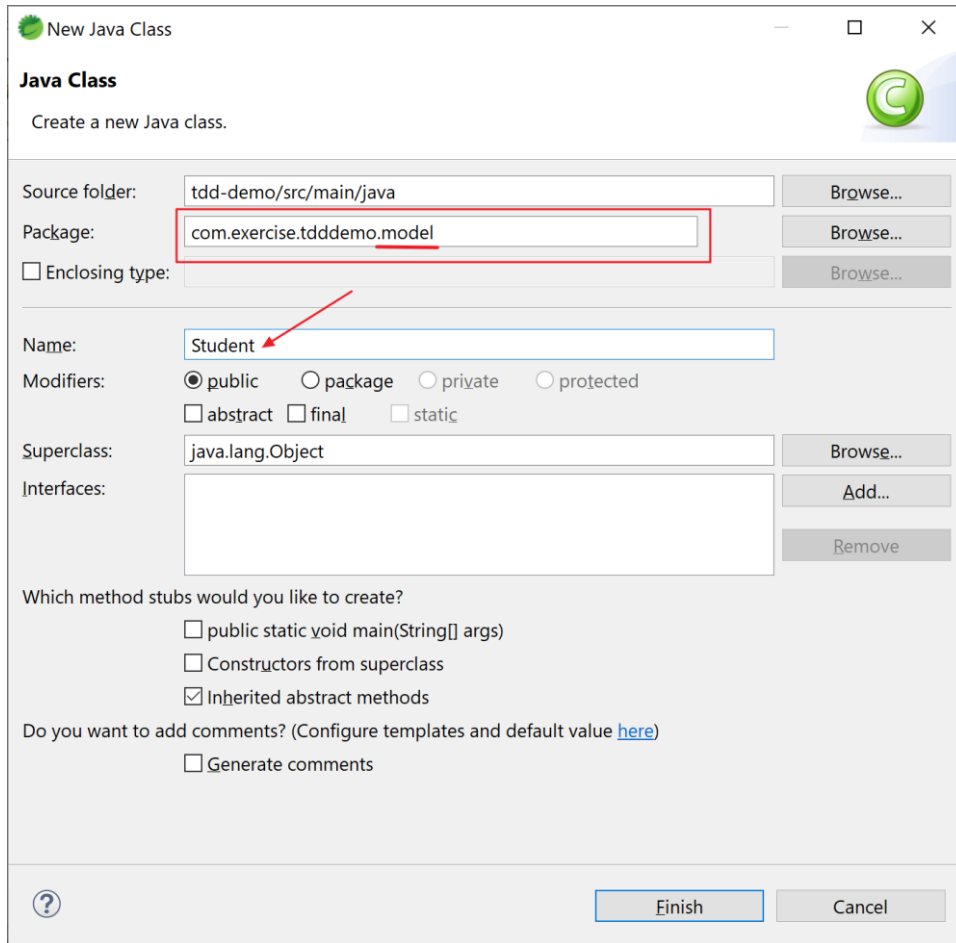
    public String getDescription() {
        return description;
    }

    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”:



New Java Class

Create a new Java class.

Source folder: tdd-demo/src/main/java Browse...

Package: com.exercise.tdddemo.model Browse...

☐ Enclosing type: Browse...

Name: Student

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

? Finish Cancel

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;
}

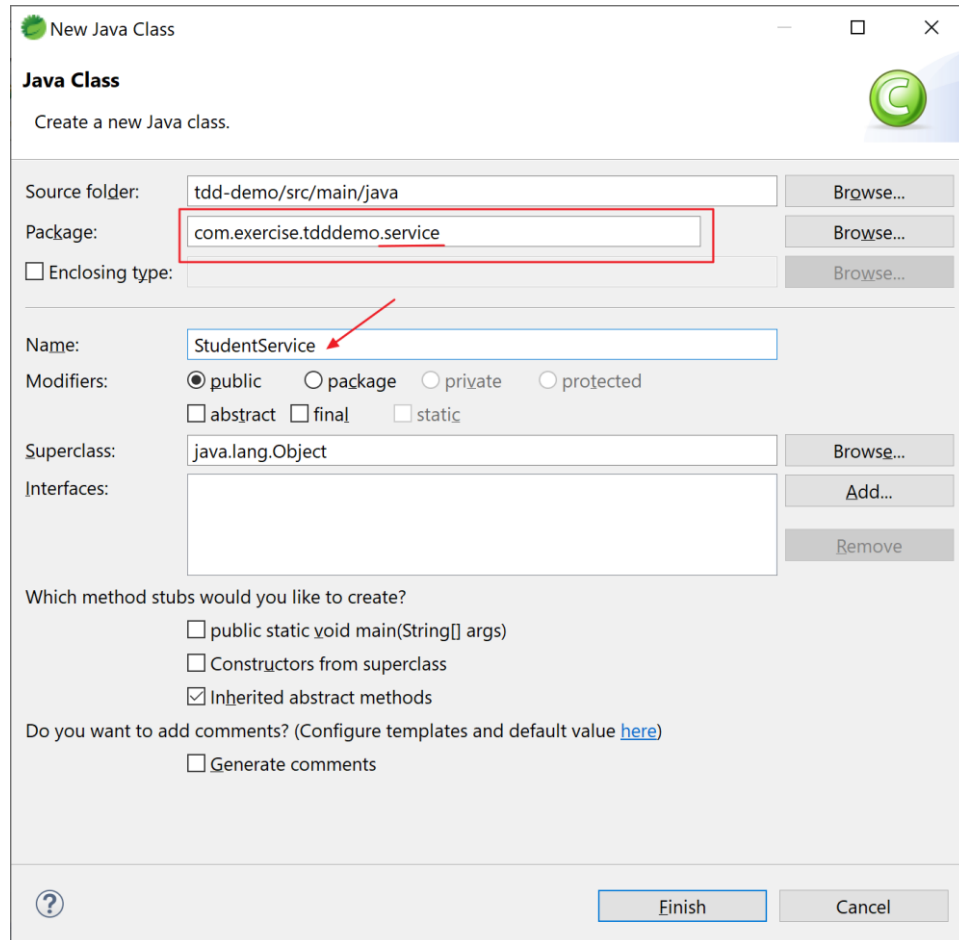
public List<Course> getCourses() {
    return courses;
}

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":



New Java Class

Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

```
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);

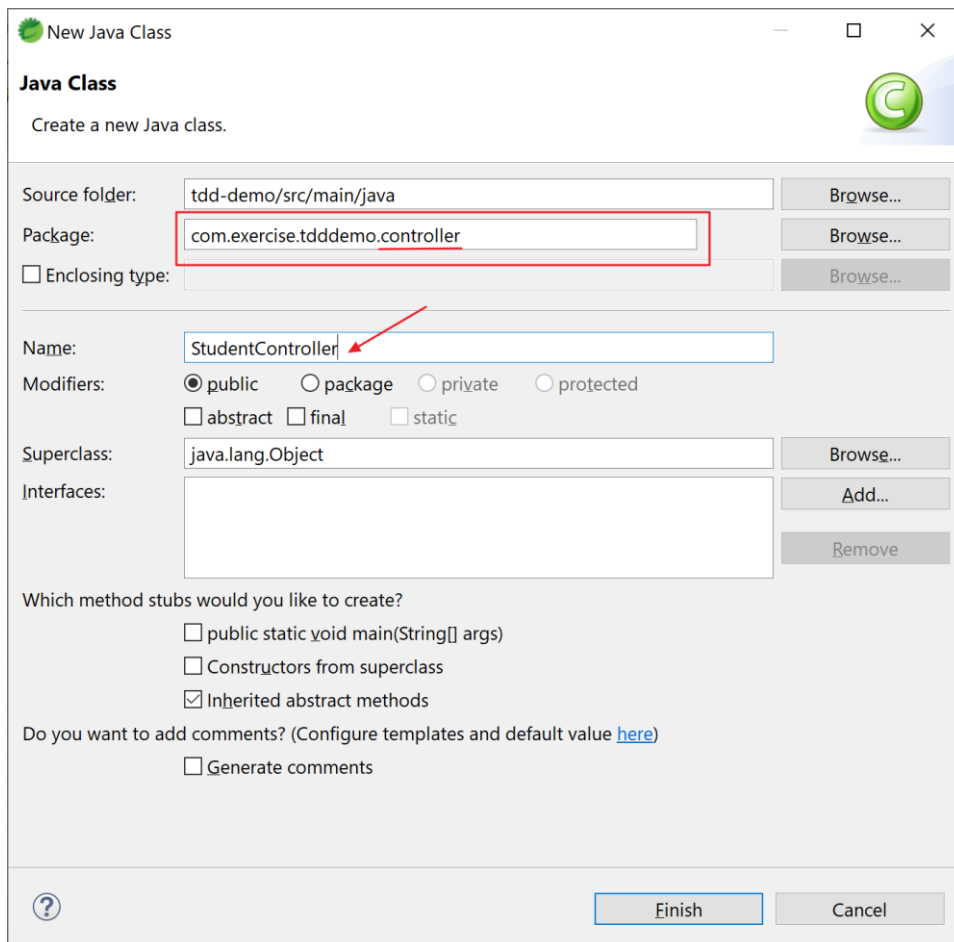
if (student == null) {
    return null;
}

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);

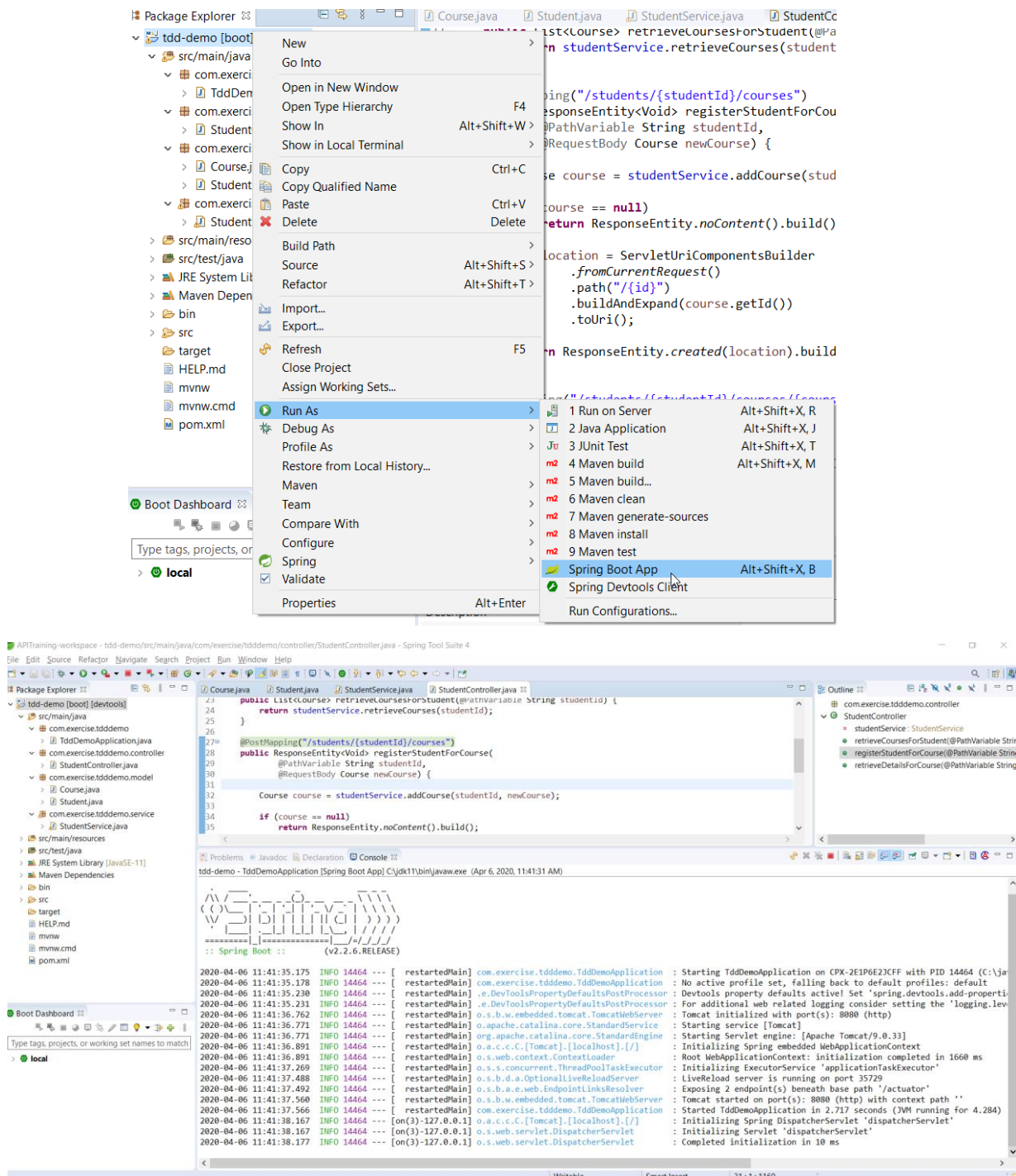
        if (course == null)
            return ResponseEntity.noContent().build();

        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);
    }
}
```

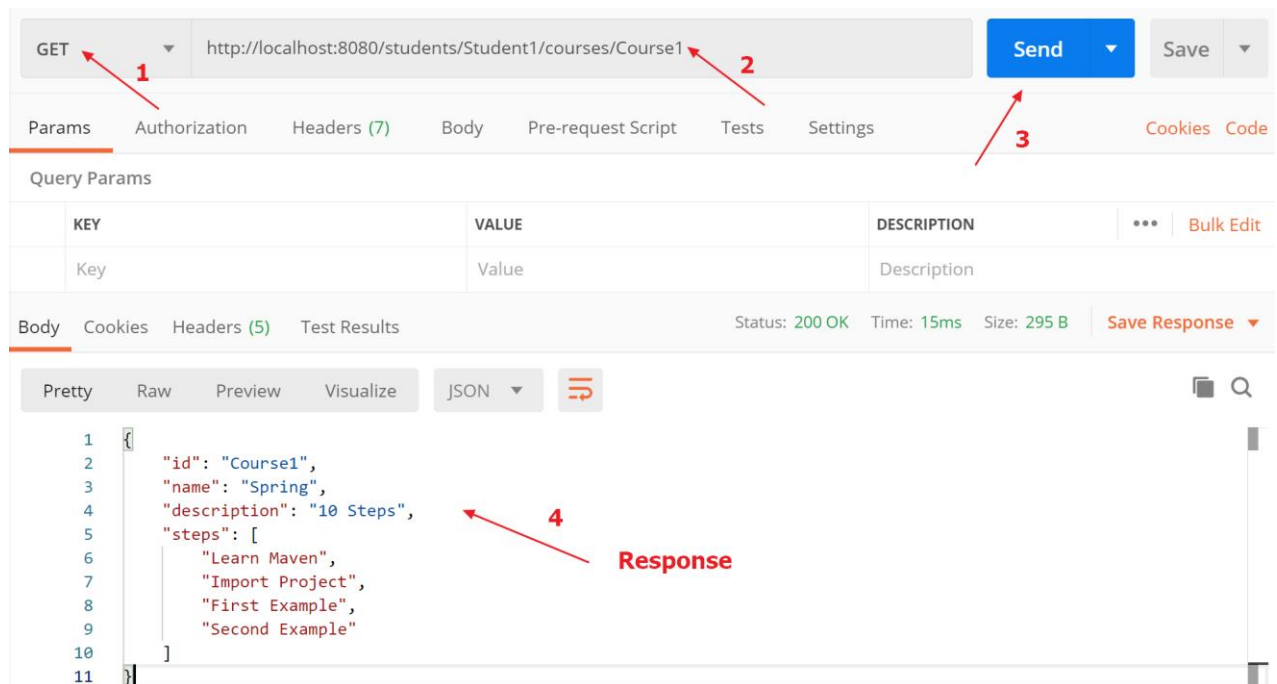
Run the Application. Right click on the project tdd-demo -> Run As -> Spring Boot App



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:



GET 1 http://localhost:8080/students/Student1/courses/Course1 2 Send 3 Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 15ms Size: 295 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": "Course1",
3   "name": "Spring",
4   "description": "10 Steps",
5   "steps": [
6     "Learn Maven",
7     "Import Project",
8     "First Example",
9     "Second Example"
10  ]
11 }

```

4 Response

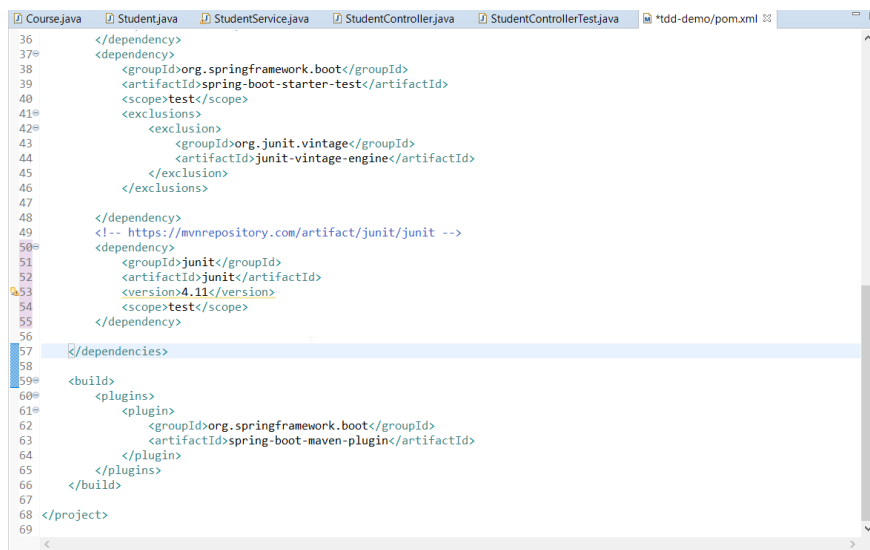
10. After making sure that the APIs work. We're going to write the unit tests.

Add Junit Dependency to your pom.xml

```

<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>

```

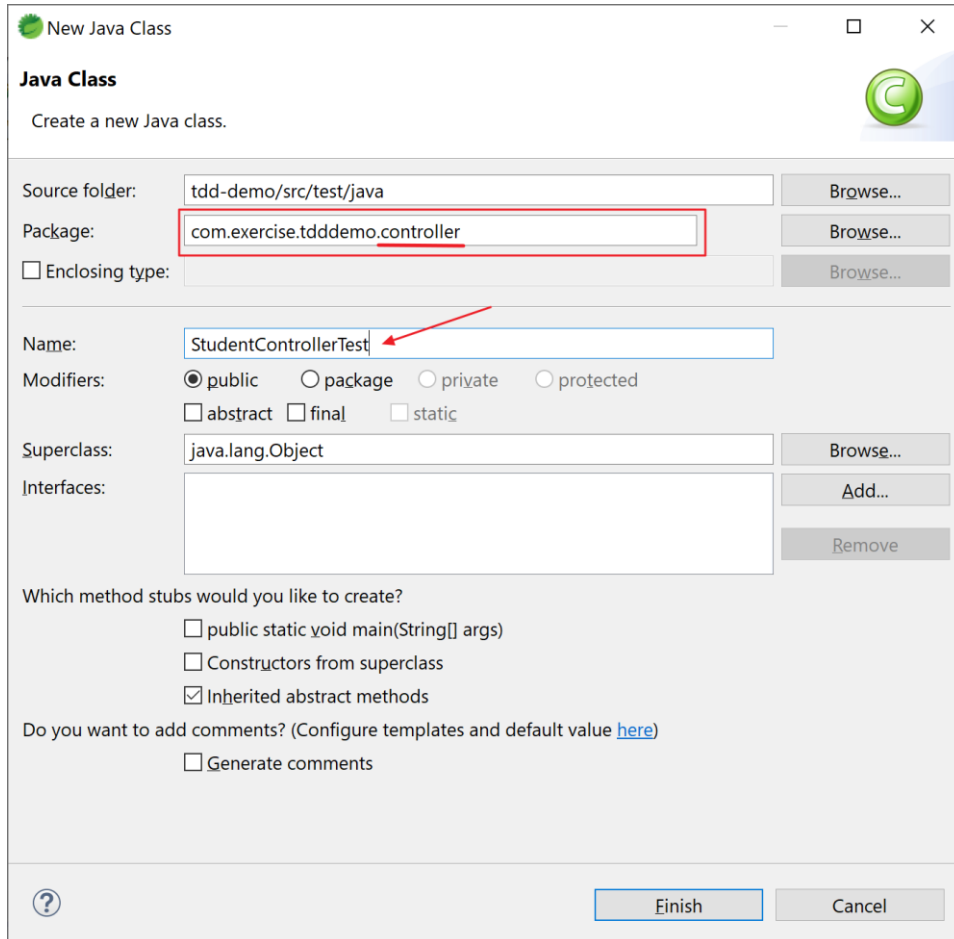


```

36 </dependency>
37 <dependency>
38   <groupId>org.springframework.boot</groupId>
39   <artifactId>spring-boot-starter-test</artifactId>
40   <scope>test</scope>
41   <exclusions>
42     <exclusion>
43       <groupId>org.junit.vintage</groupId>
44       <artifactId>junit-vintage-engine</artifactId>
45     </exclusion>
46   </exclusions>
47 </dependency>
48 <!-- https://mvnrepository.com/artifact/junit/junit -->
49 <dependency>
50   <groupId>junit</groupId>
51   <artifactId>junit</artifactId>
52   <version>4.11</version>
53   <scope>test</scope>
54 </dependency>
55 </dependencies>
56
57 <build>
58   <plugins>
59     <plugin>
60       <groupId>org.springframework.boot</groupId>
61       <artifactId>spring-boot-maven-plugin</artifactId>
62     </plugin>
63   </plugins>
64 </build>
65 </project>
66
67
68
69

```

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.AP
PLICATION_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));
    }
}

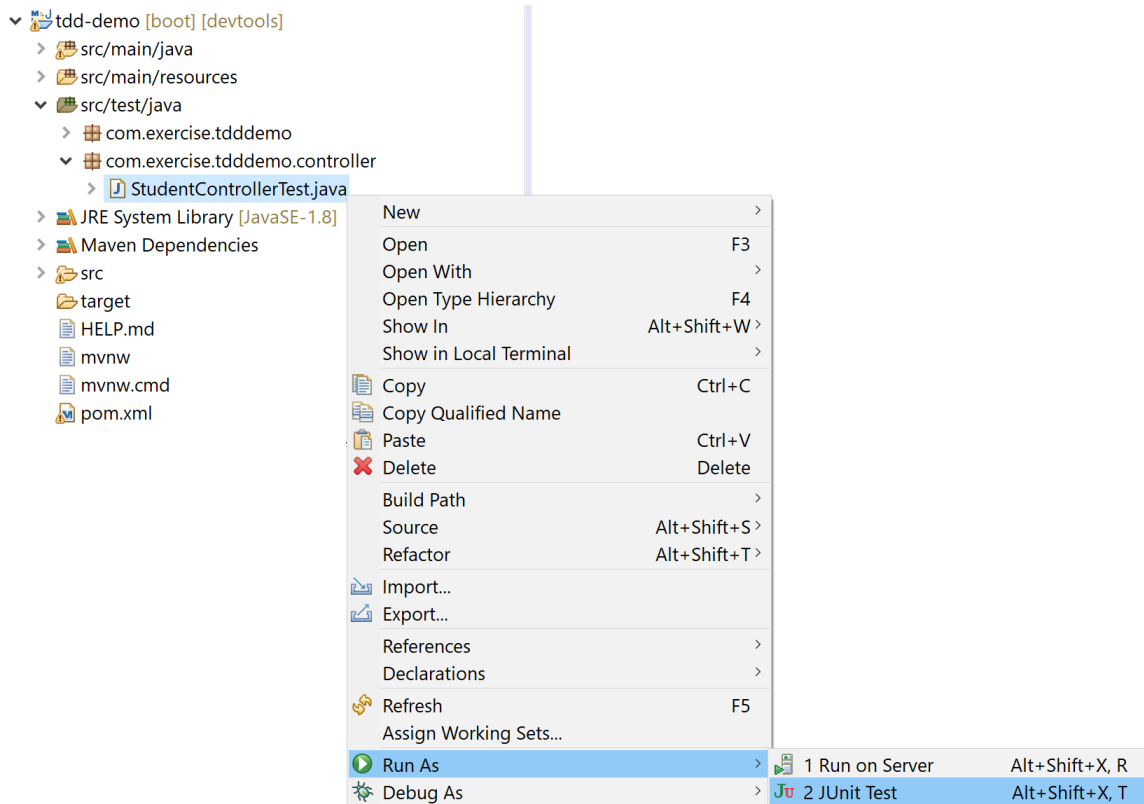
```

Note:

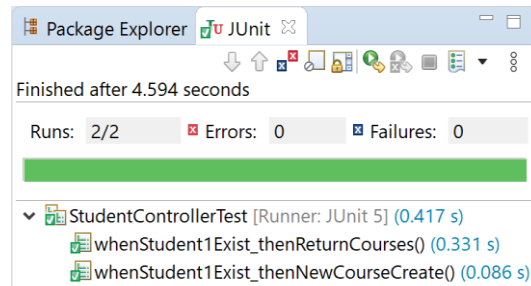
- i. We annotate the tests by using `@RunWith` (SpringRunner is an alias for the SpringJUnit4ClassRunner)
- ii. Annotation `@WebMvcTest` is used to fire up an application context that contains only the beans needed for testing a web controller
- iii. Annotation `@MockBean` is used to mock away the business logic. It automatically replaces the bean of the same type in the application context with a Mockito mock.
- iv. Mockito allows you to create and configure mock objects. Using Mockito greatly simplifies the development of tests for classes with external dependencies.
- v. 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:

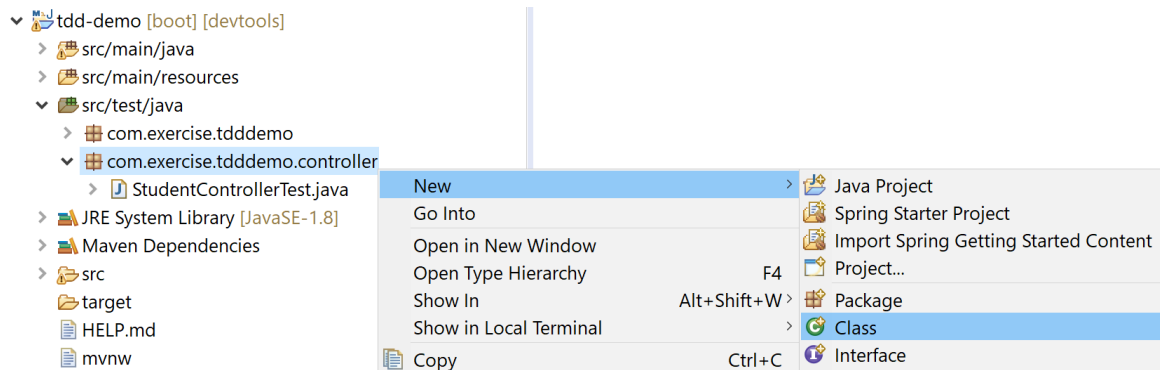
```
[INFO] Results:
[INFO]
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO]
[INFO] Total time: 12.913 s
[INFO] Finished at: 2020-03-30T10:19:27+08:00
[INFO] -----
```

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”:



New Java Class

Create a new Java class.

Source folder: tdd-demo/src/test/java Browse...

Package: com.exercise.tdddemo.controller Browse...

☐ Enclosing type: Browse...

Name: StudentControllerIntegrationTest

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

? Finish Cancel

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;
    }
}
```

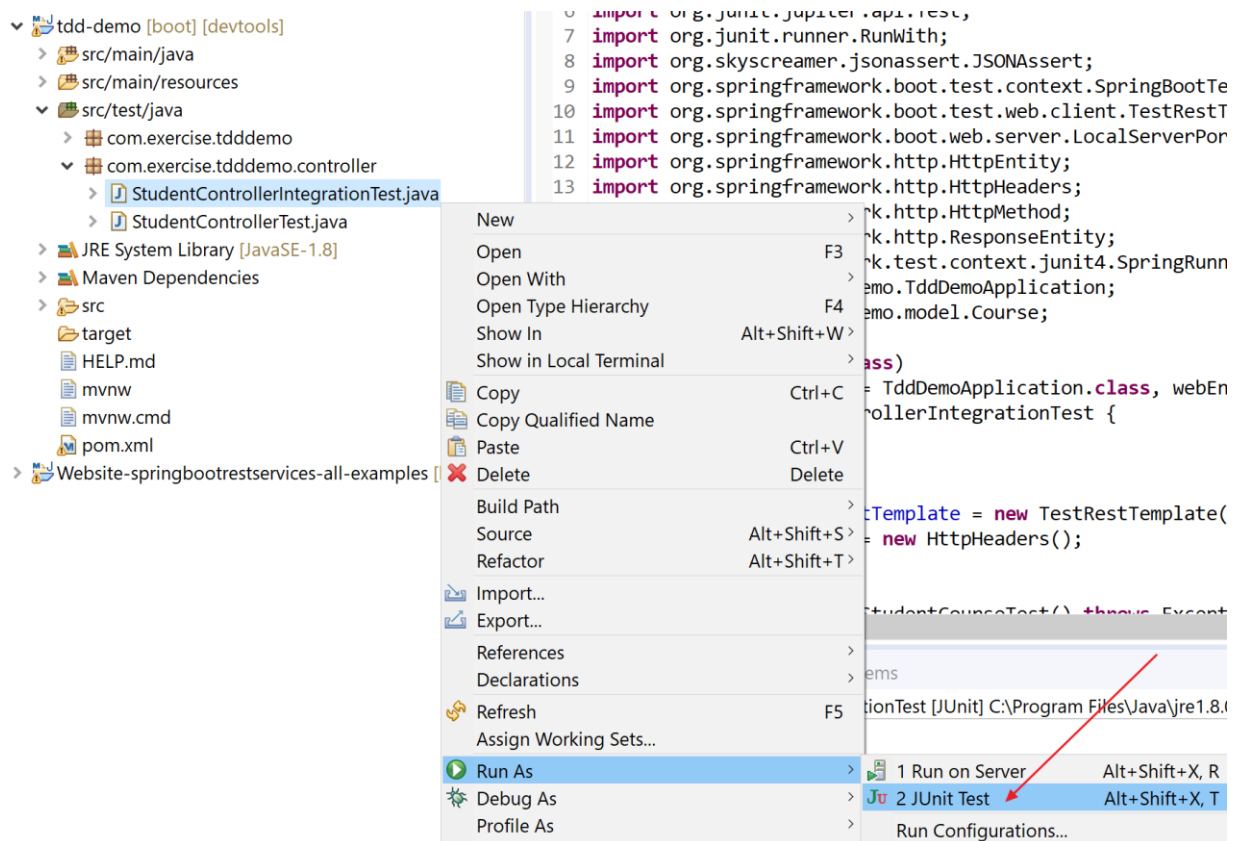
Note:

- i. RANDOM_PORT is for restTemplate to use port randomly

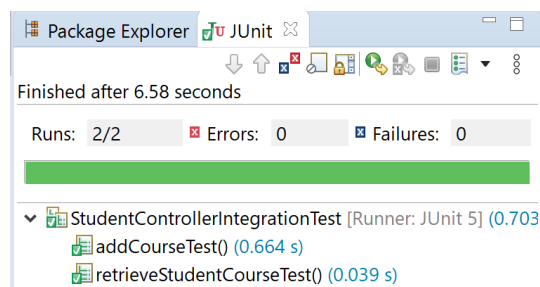
- ii. restTemplate is used to send request to and retrieve the response from the APIs
- iii. assertEquals is used to validate the results

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:

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.238 s
[INFO] Finished at: 2020-03-30T14:14:19+08:00
[INFO] -----
```

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:



tdd-demo_complete.zip