Steps

Steps:

1. Follow the instructions in **bootcamp-install-environments-v2.0.pdf** to install the environments
    - Some of the installation items are not needed for this Library Project but are needed for the class

2. Follow the instructions in this document's section, *Using Eclipse (STS), Create Spring Boot Project for the Library Application*

3. Using the requirements in this document's section, *Library Project Requirements*, create the library project

4. Reach out to wleonar@us.ibm.com if any issues

5. Create a zip file for the project in following format: library-lastname-firstname-ibmid.zip (e.g. library-smith-john-js123.zip)

6. Add the project to an IBM box folder or as an email attachment

7. Send email notifying of completion and location of project to wleonar@us.ibm.com, cewhalen@us.ibm.com

8. A meeting will be scheduled with you to review the project

Library Project Requirements

Requirements

1. Create the start of a Library project developed in Spring and Java

2. It needs to be a Spring Boot application
   - You don't have to know Spring Boot; the instructions in the following sections provide all that you need:
     - *Using Eclipse (STS), Create Spring Boot Project for the Library Application*
     - *Running the Library Project Application*

3. It will read books from a text file and store them in memory

4. It will have a single REST API method that will return all the books that were read in from the text file

5. The Book information that is returned from the REST API will have some additional information added based on the requirements specified in **Service / business logic class page/slide**

6. When done, make sure it works using the instructions in Running the Library Project Application to run the application and ... use a REST client (e.g. Postman) to send the http request to it

Model Classes

1. Create the Java package, com.ibm.library.model

2. Create the following Java classes in this package:
   a. Book class - abstract class
   b. 2 child classes of Book with each calculating the late fee differently:
      i. BookFiction
         - Add late fee method that does the following:
           - if numDaysLate is negative, throw the user-defined exception, BadValue
           - late fee = (# of days late / 2) * 75 cents
      ii. BookNonFiction
         - Add late fee method that does the following:
           - if numDaysLate is negative, throw the user-defined exception, BadValue
           - late fee = (# of days late * 1.5) * 80 cents
   c. All books have the same set of fields: isbn (unique value), title, author, notes

Controller, Service and Database

1. Create a **Spring RestController class** in its appropriate java package (software architecture layer) that does the following:
    a. It receives a REST request to retrieve all books and returns a JSON-formatted list of books
    b. It uses the Service/business logic class

2. Create a **Service / business logic class** in its appropriate java package (software architecture layer) that does the following:
    a. Uses a database/repo class (see step 3 below) to get the books from the 'database'
    b. Iterates through the list of books, processing each Book as follows:
        i. If the author is "Tom Smith", it appends " - CHECKED" to the author
        ii. Call each book's calculateLateFee(numberOfDaysLate) method, passing in the 'number of days late' value based on the following:
            - if the number of characters in the book's title is an odd number then set 'number of days late' to: -1 * the number of characters in the book's title
            - else set 'number of days late' to: number of characters in the book's title
        iii. If book's calculateLateFee(numberOfDaysLate) throws BadValue exception, it sets book's 'notes' field to BadValue's exception message
        iv. If book's calculateLateFee(numberOfDaysLate) does NOT throw BadValue exception, it sets book's 'notes' field to "Fee is: " + fee where 'fee' is the value returned from calculateLateFee(numberOfDaysLate)

3. Create a **Repository / Database class** in its appropriate java package (software architecture layer) that does the following:
    - *For now, we're not getting the Books data from a database but, instead, from a file*
    a. Each line in the file represents a single book with the following format: book type|isbn|title|author

        E.g.
        FICTION|12345|Some Book|Tom Smith
        FICTION|87887|Whatever|Aanand Agarwal
        NONFICTION|99445|Another Book|Lucy Chen

    b. Parse each of the lines of Book data, setting the appropriate Book object fields (isbn, author, or title) and creating the correct Book object (BookFiction or BookNonFiction) based on the type read in from the file

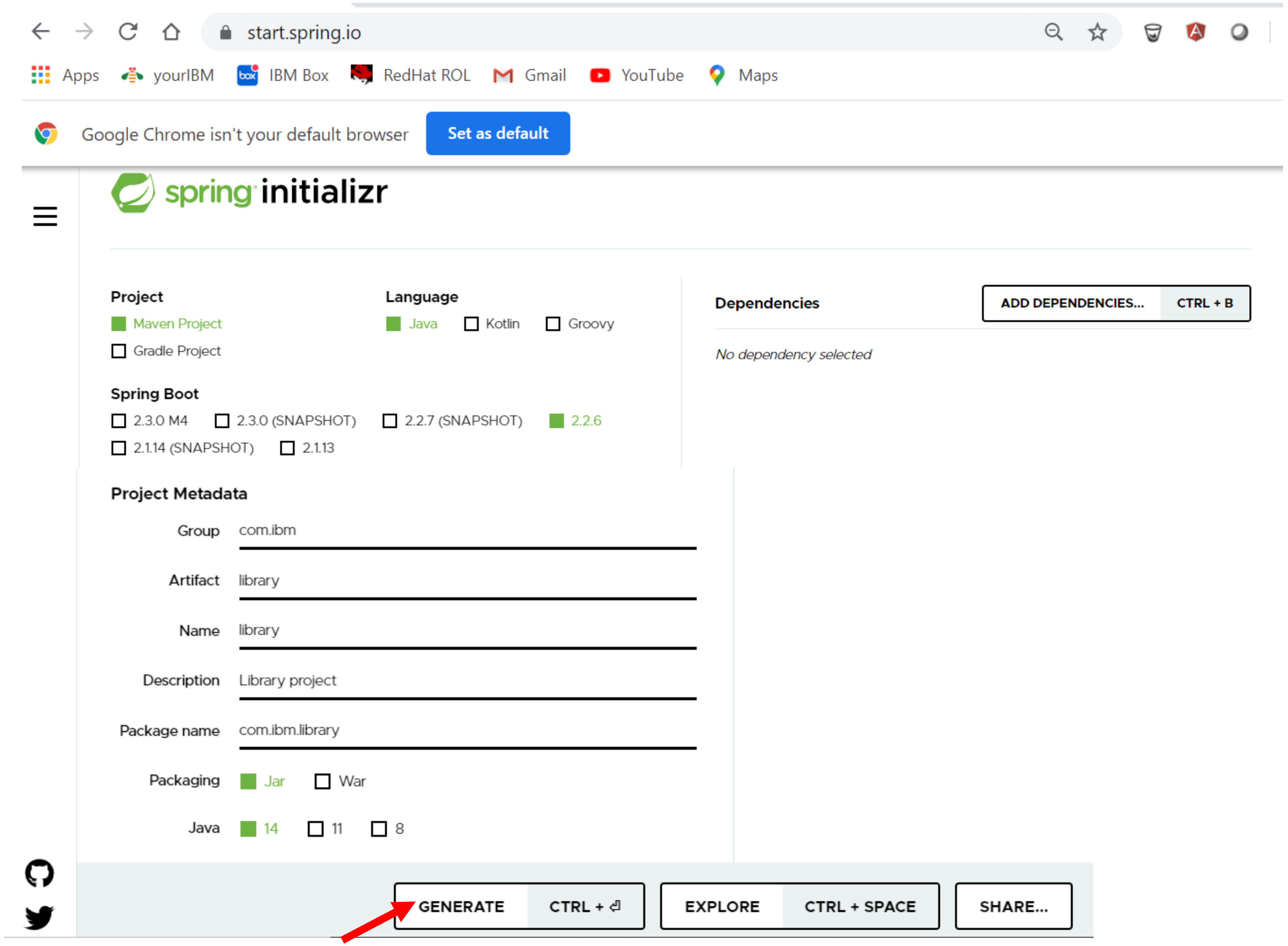
Using Eclipse (STS), Create Spring Boot Project for the Library Application

Create Spring Boot Java Project Using Spring Initializr

1. Open a browser and go to https://start.spring.io/

2. Fill out the fields:
   - Project: Select 'Maven Project'
   - Language: Select 'Java'
   - Spring Boot: Accept the default (mine is 2.26)
   - Project Metadata:
     - Group: *com.ibm*
     - Artifact: *library*
     - Name: *library*
     - Description: *Library Project*
     - Package name: *com.ibm.library*
     - Packaging: Jar
     - Java: *14*

3. Click on **GENERATE**



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

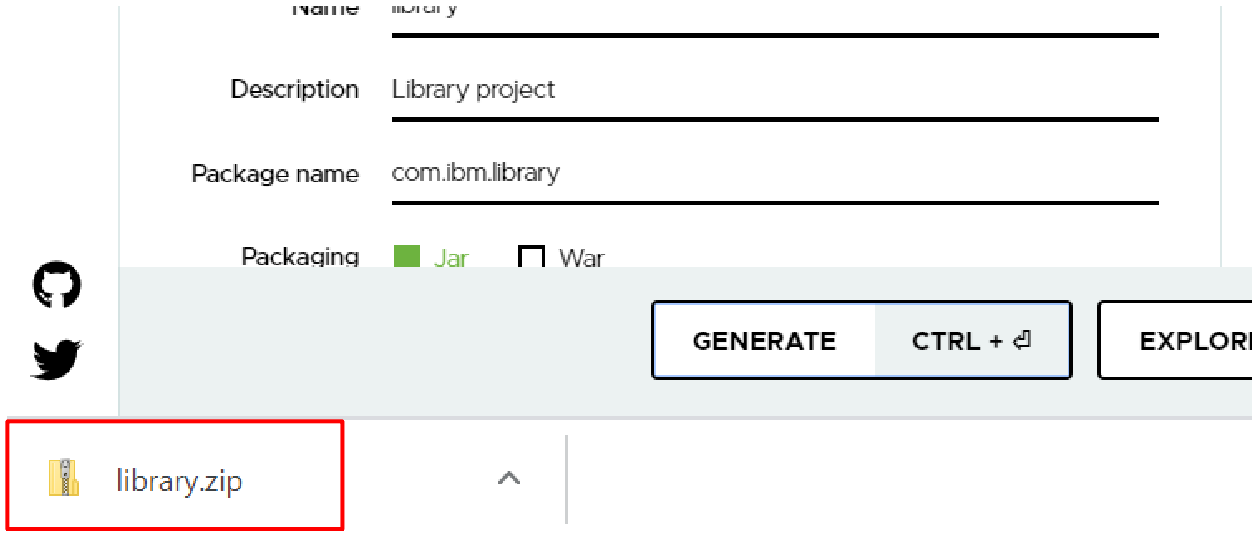
Dependencies
No dependency selected

Project Metadata

Group	com.ibm
Artifact	library
Name	library
Description	Library project
Package name	com.ibm.library
Packaging	🟩 Jar ☐ War
Java	🟩 14 ☐ 11 ☐ 8

Spring Initializr Generates the Spring Boot Java Project and creates a zip file for it

After clicking on GENERATE, spring initializr will create your Spring Boot Java project and automatically download it as library.zip



Name library

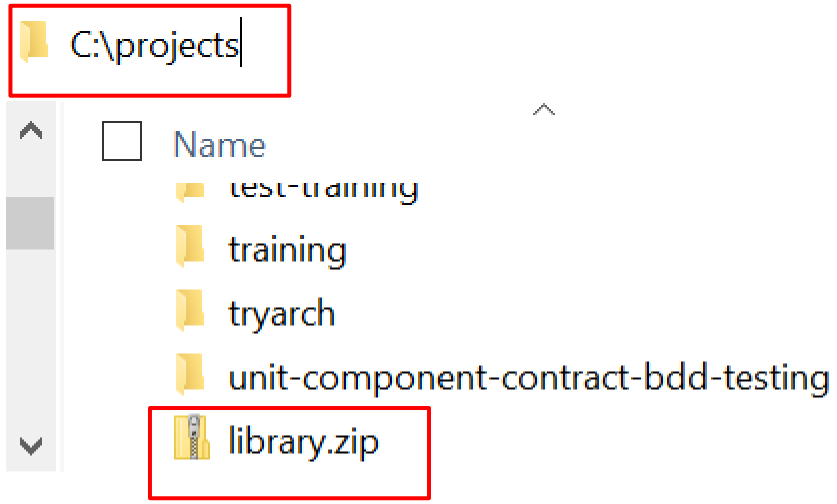
Description Library project

Package name com.ibm.library

Packaging 🟩 Jar ☐ War

Move the Spring Boot Java project (zip file) to another folder

Move library.zip to C:\projects folder;
result: C:\projects\library.zip

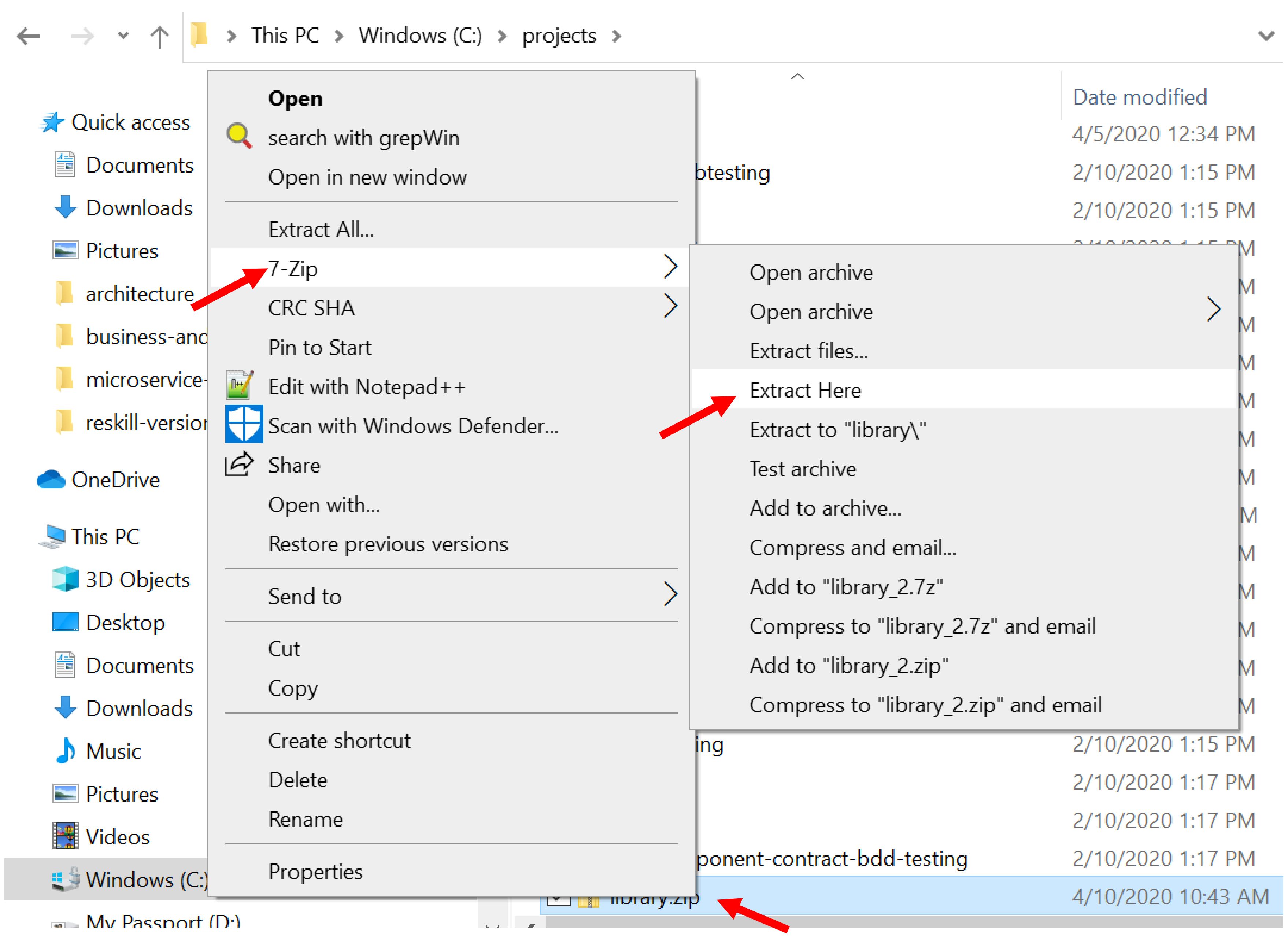

C:\projects

Name

test-training
training
tryarch
unit-component-contract-bdd-testing
library.zip

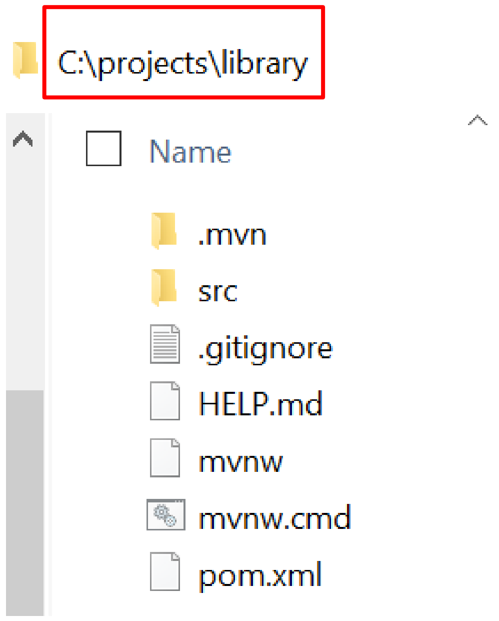
Unzip the Spring Boot Java project

- Unzip library.zip in the C:\projects folder (for windows users, use 7zip; don't use windows provided zip utility)
- On Windows:
    - right-click library.zip
    - Select '7-Zip' then 'Extract Here'



After unzipping the zip file, you'll see the folder, library

Result after unzipping library.zip: C:\projects\library folder with the contents as shown


C:\projects\library

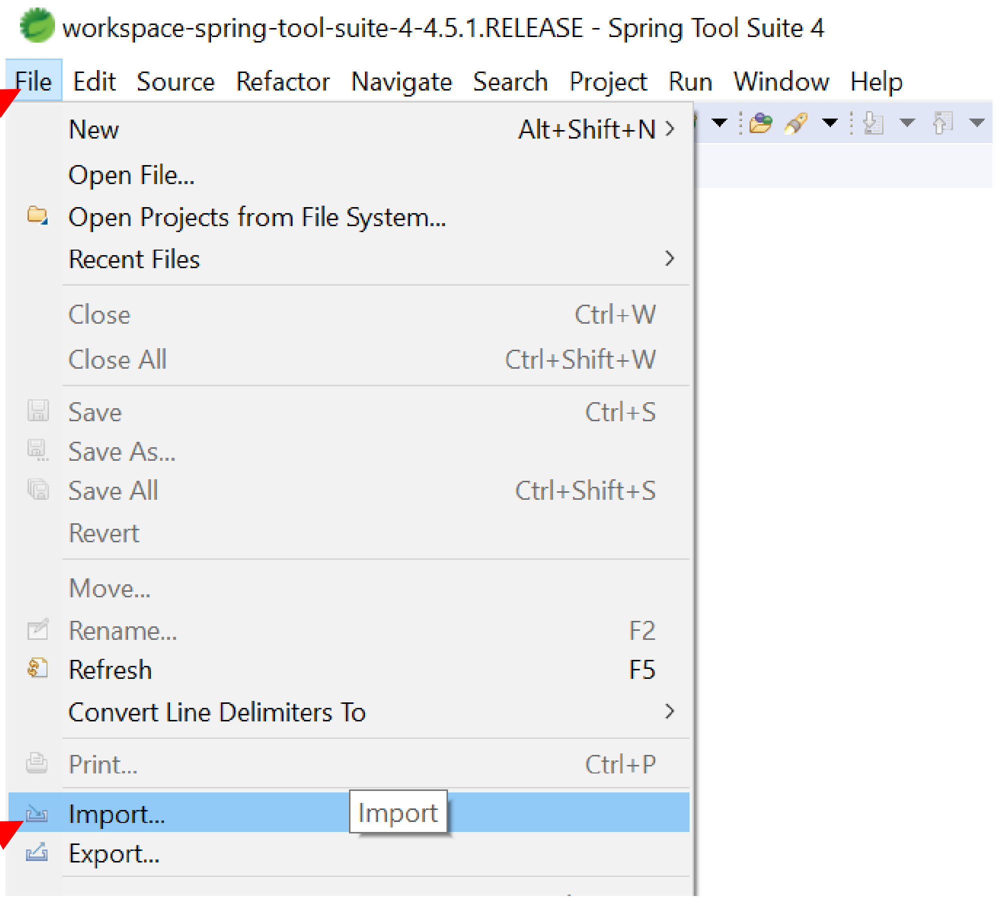
Name

.mvn
src
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml

Import the library project (using its maven project, pom.xml) into Spring Tool Suite (STS) - Slide 1 of 8

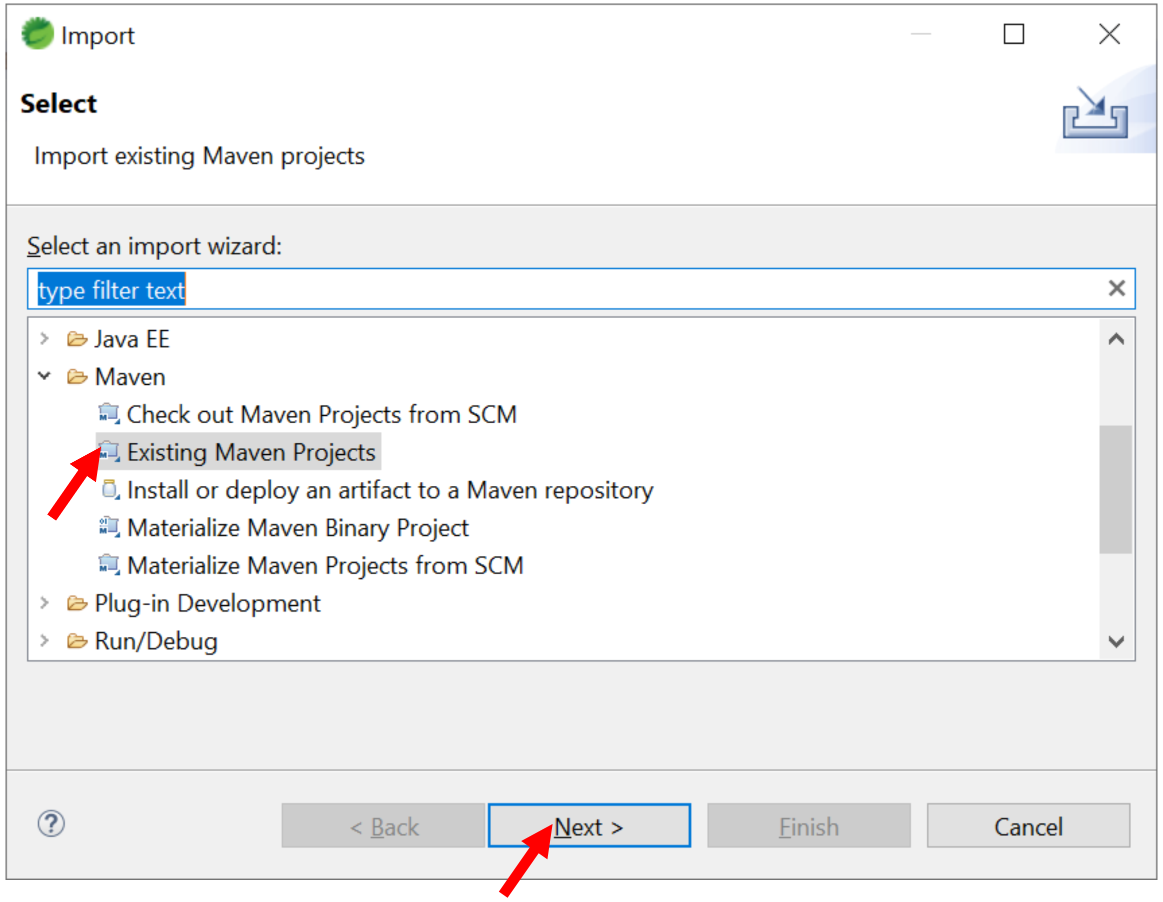
Start STS (Spring Tool Suite)

In STS, click on 'File' then 'Import'


workspace-spring-tool-suite-4-4.5.1.RELEASE - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Import
Export...

Import the library project (using its maven project, pom.xml) into Spring Tool Suite (STS) - Slide 2 of 8

- Under 'Maven', select 'Existing Maven Projects'
- Click on 'Next'


Import

Select

Import existing Maven projects

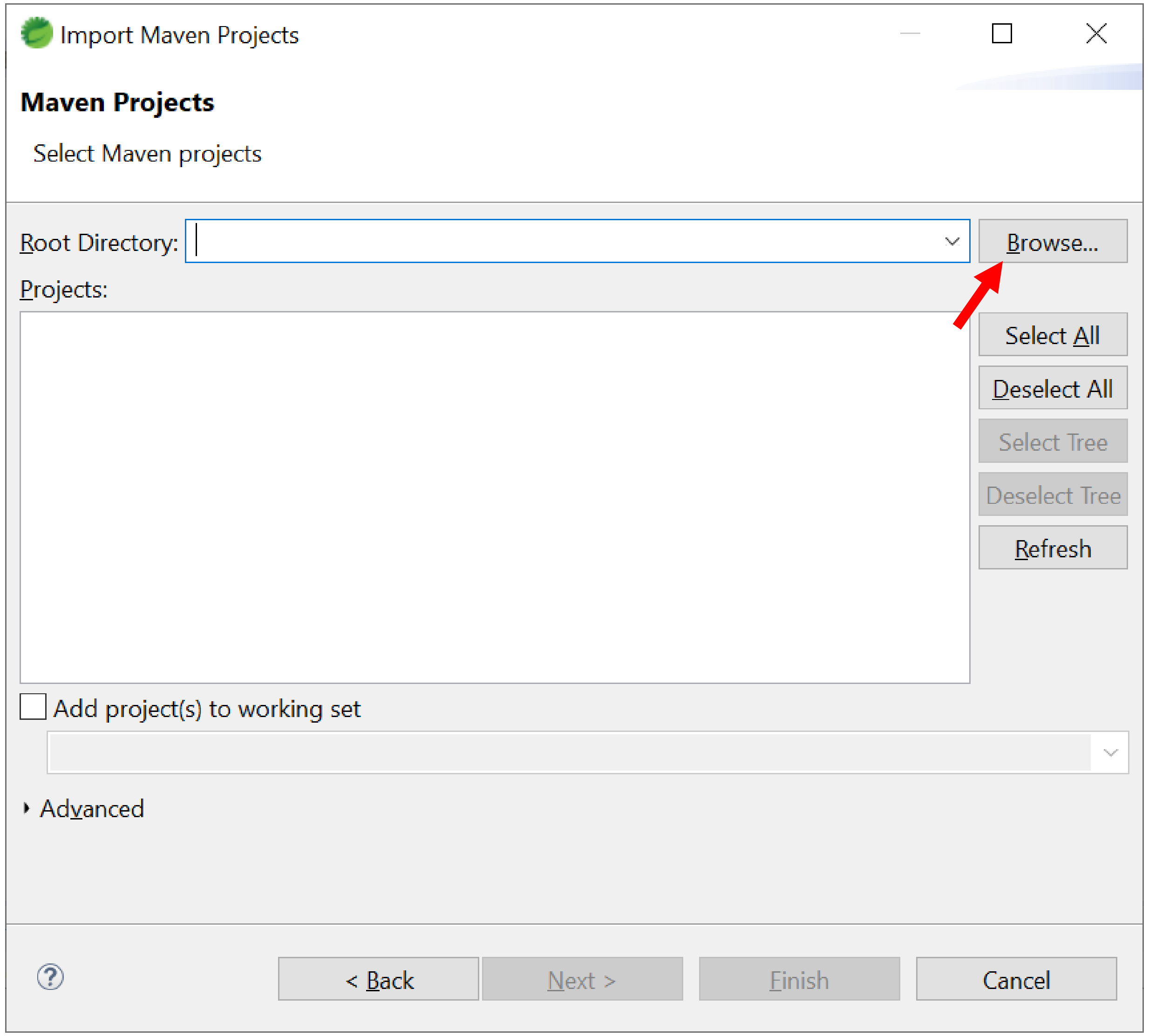
Select an import wizard:

type filter text

- Java EE
- Maven
  - Check out Maven Projects from SCM
  - Existing Maven Projects
  - Install or deploy an artifact to a Maven repository
  - Materialize Maven Binary Project
  - Materialize Maven Projects from SCM
- Plug-in Development
- Run/Debug

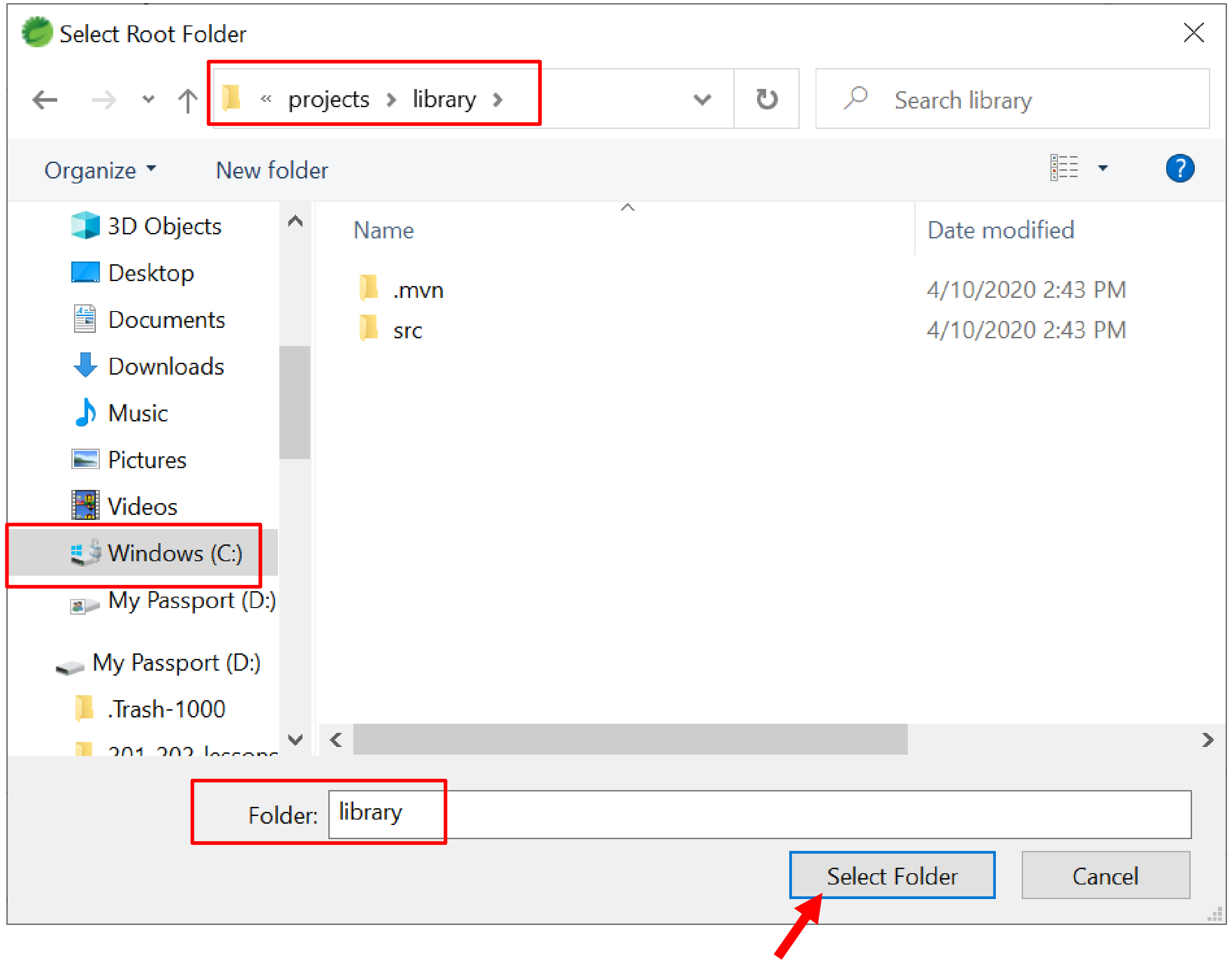
Import the library project (using its maven project, pom.xml) into Spring Tool Suite (STS) - Slide 3 of 8

- Click on Browse



Import the library project (using its maven project, pom.xml) into Spring Tool Suite (STS) - Slide 4 of 8

- Browse to C:\projects\libary folder
- Click on 'Select Folder'

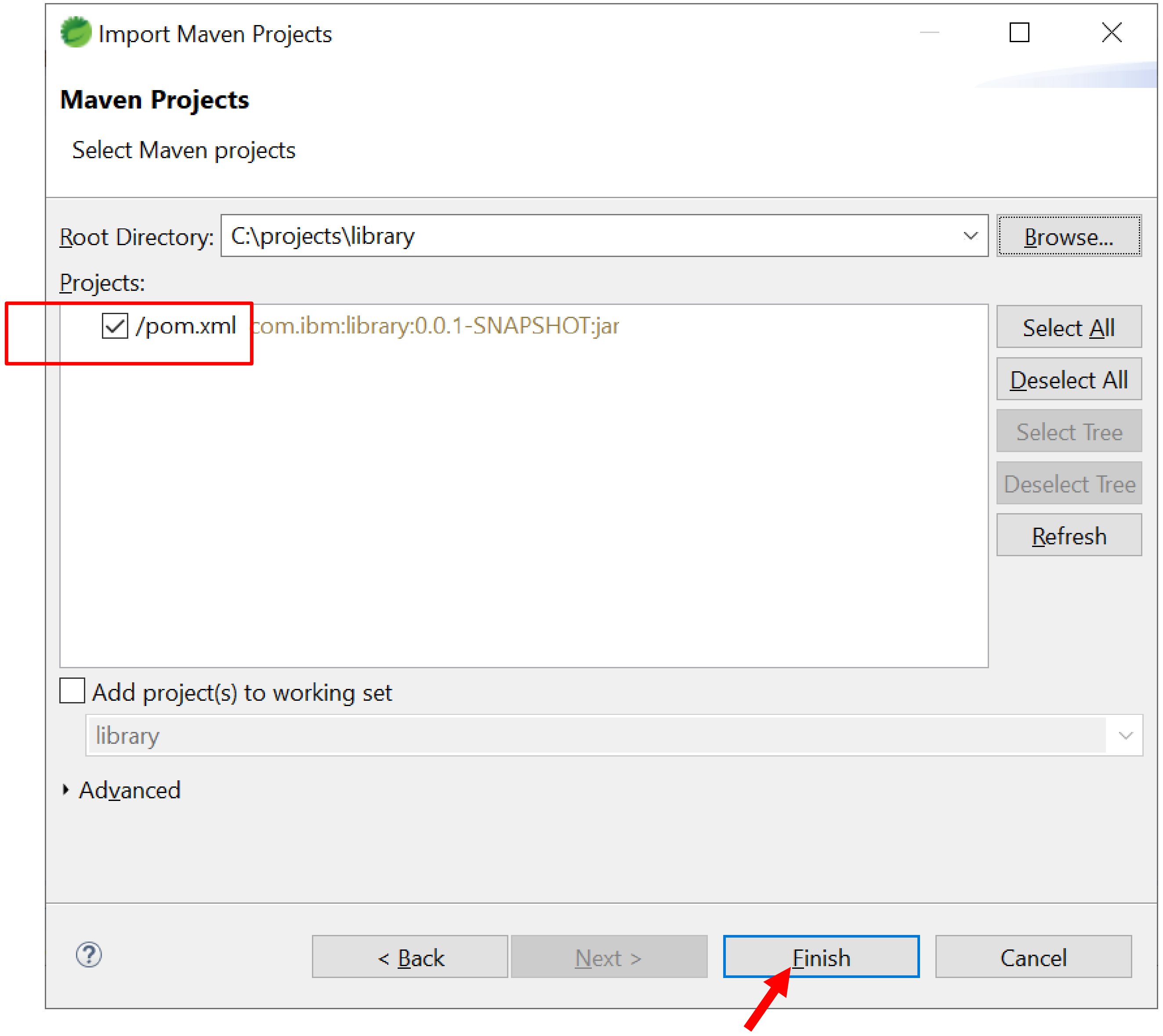


Name	Date modified
.mvn	4/10/2020 2:43 PM
src	4/10/2020 2:43 PM

Folder: library

Import the library project (using its maven project, pom.xml) into Spring Tool Suite (STS) - Slide 5 of 8

- Make sure that you see the pom.xml check box is checked
- Click 'Finish'


Import Maven Projects

Maven Projects

Select Maven projects

Root Directory: C:\projects\library

Projects:

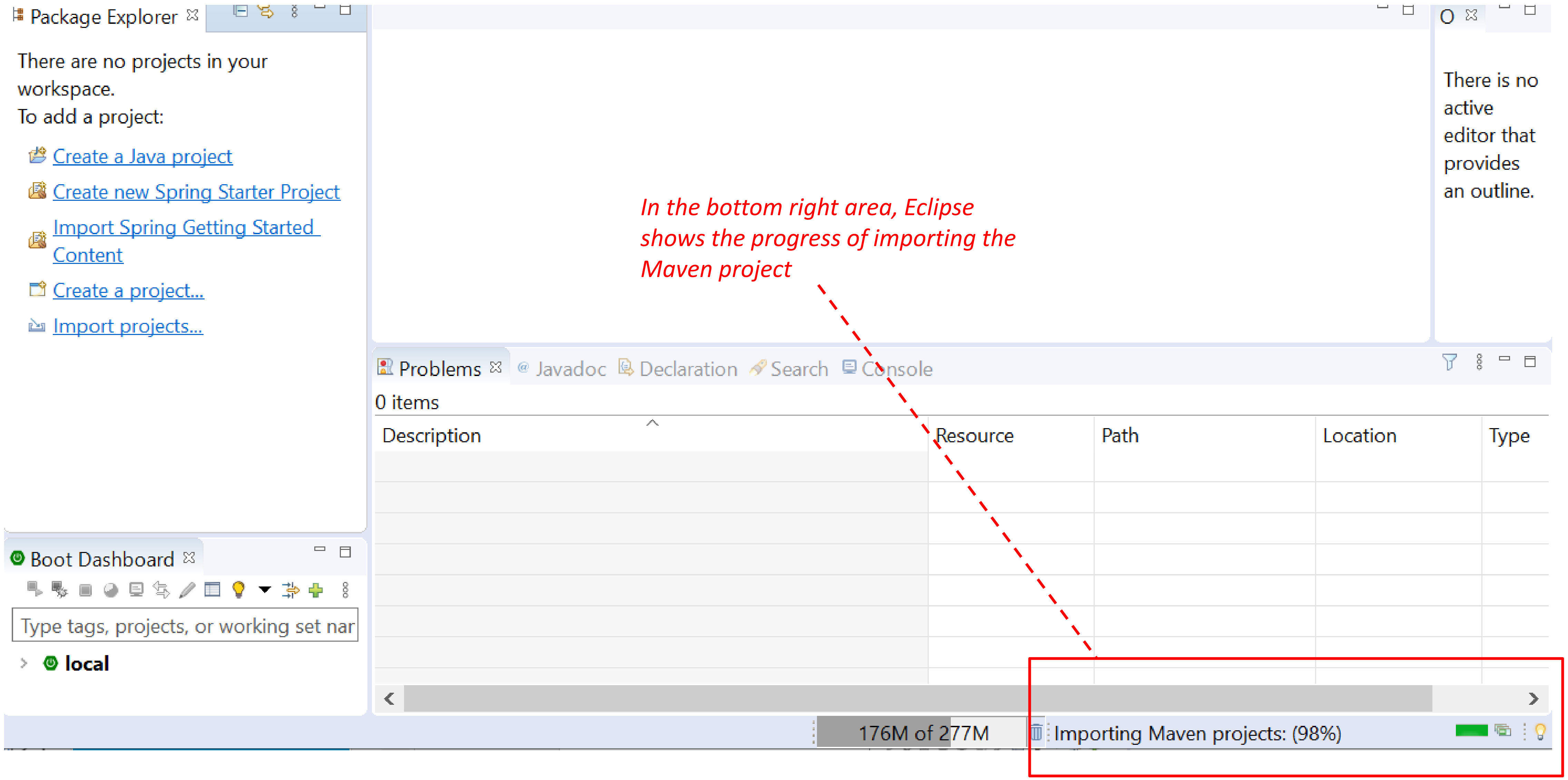
[✓] /pom.xml com.ibm:library:0.0.1-SNAPSHOT:jar

[ ] Add project(s) to working set

library

Notes:
- pom.xml is the Maven project for our Java project
- Spring Initialzr generated this Maven pom.xml for us

Import the library project (using its maven project, pom.xml) into Spring Tool Suite (STS) - Slide 6 of 8



In the bottom right area, Eclipse shows the progress of importing the Maven project

Package Explorer

There are no projects in your workspace.
To add a project:

- Create a Java project
- Create new Spring Starter Project
- Import Spring Getting Started Content
- Create a project...
- Import projects...

There is no active editor that provides an outline.

Problems | Javadoc | Declaration | Search | Console

0 items

Description	Resource	Path	Location	Type

Boot Dashboard

Type tags, projects, or working set nar

local

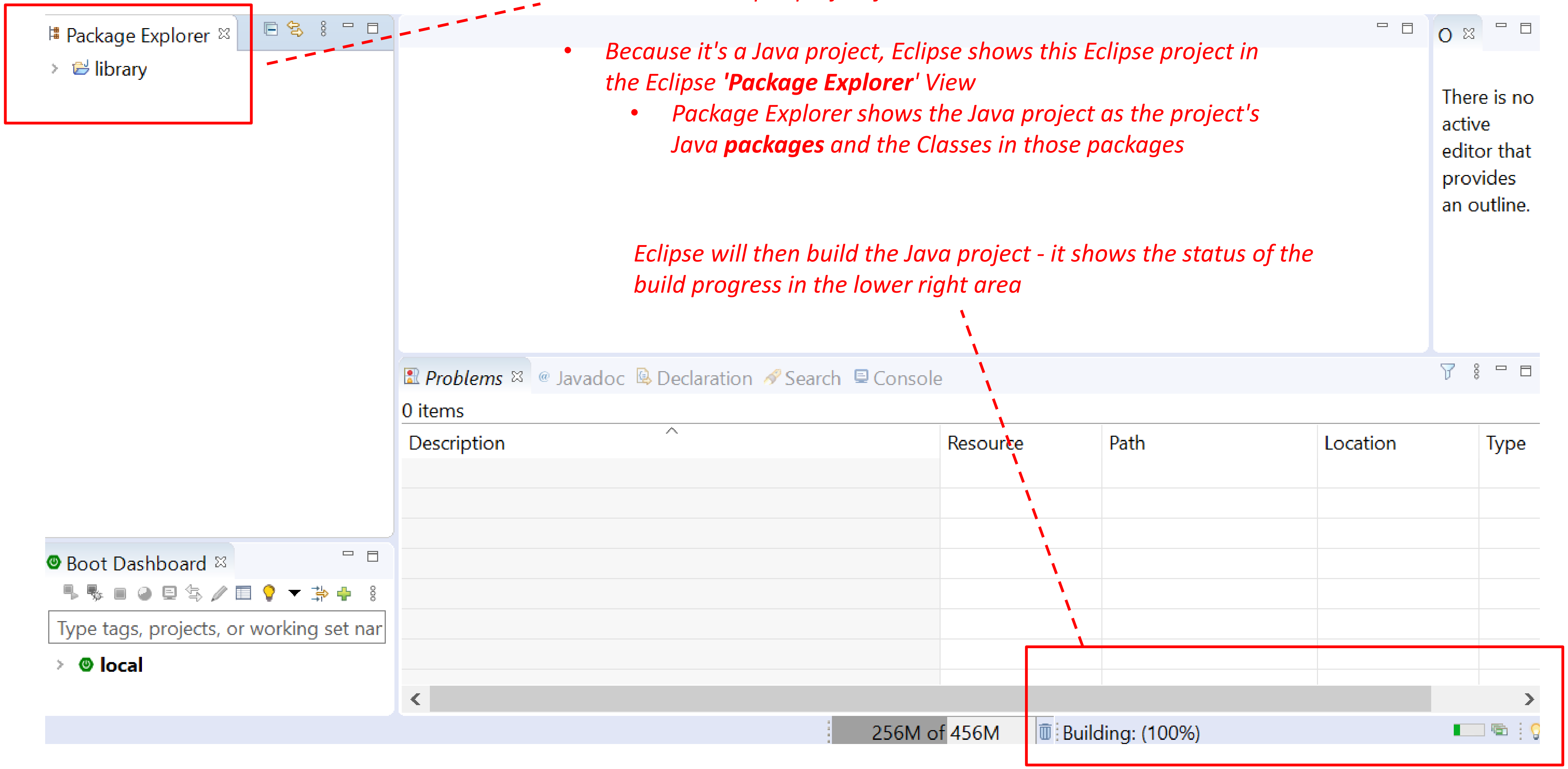
176M of 277M

Importing Maven projects: (98%)

Import the library project (using its maven project, pom.xml) into Spring Tool Suite (STS) - Slide 7 of 8

- *After the maven project is imported into Eclipse, Eclipse creates an Eclipse project for it*
- *Because it's a Java project, Eclipse shows this Eclipse project in the Eclipse* ***'Package Explorer'*** *View*
  - *Package Explorer shows the Java project as the project's Java* ***packages*** *and the Classes in those packages*

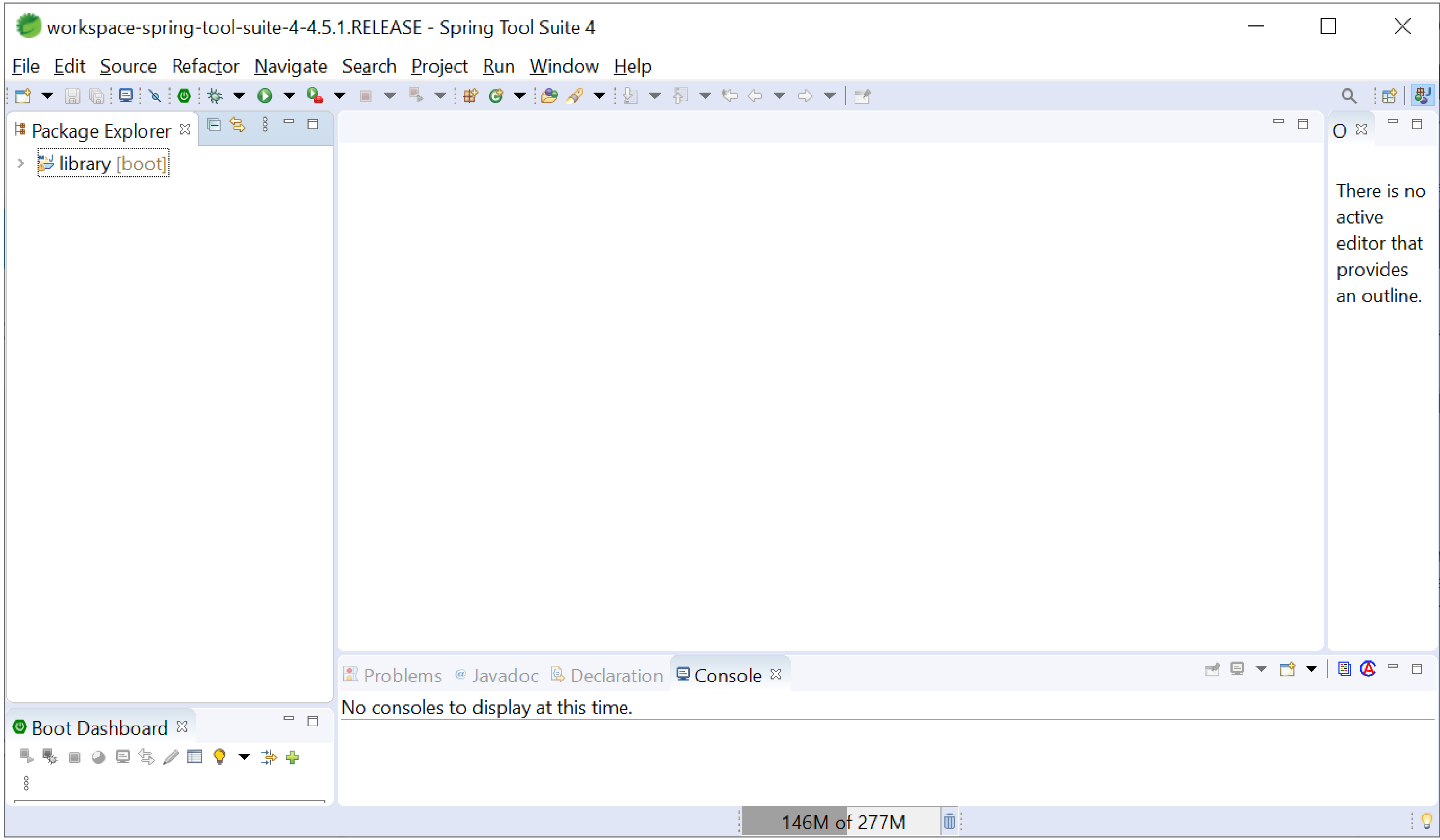
Eclipse will then build the Java project - it shows the status of the build progress in the lower right area



Import the library project (using its maven project, pom.xml) into Spring Tool Suite (STS) - Slide 8 of 8

Result: the library project in STS

Notes:
STS (version of Eclipse) created an Eclipse project (.project) from the maven project (pom.xml)


workspace-spring-tool-suite-4-4.5.1.RELEASE - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
library [boot]
O
Boot Dashboard
Problems
Javadoc
Declaration
Console
146M of 277M

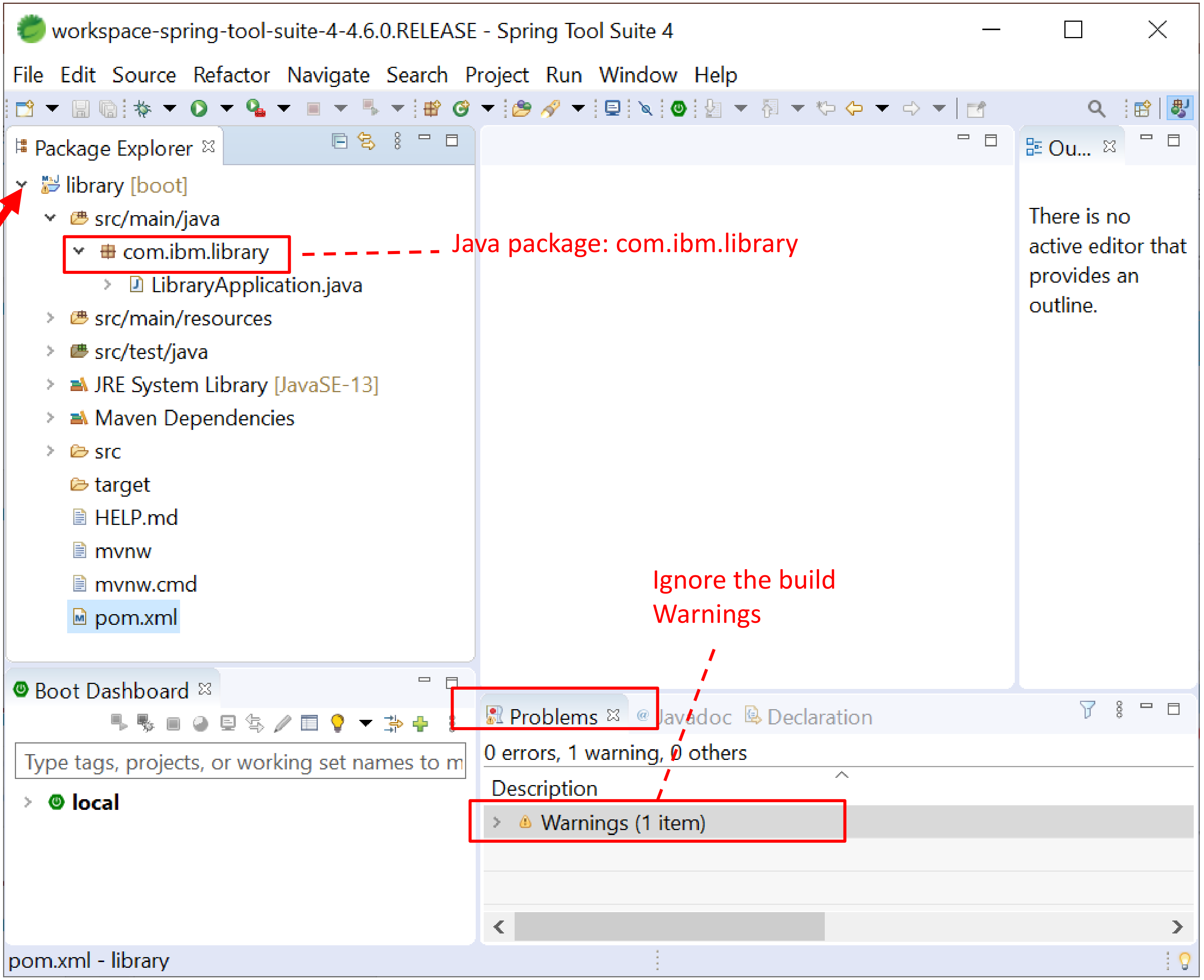
There is no active editor that provides an outline.

No consoles to display at this time.

Examine the Spring Boot Java project

Expand the library project

You'll see 1 Java package in it: **com.ibm.library** (in the source folder, src/main/java)


workspace-spring-tool-suite-4-4.6.0.RELEASE - Spring Tool Suite 4

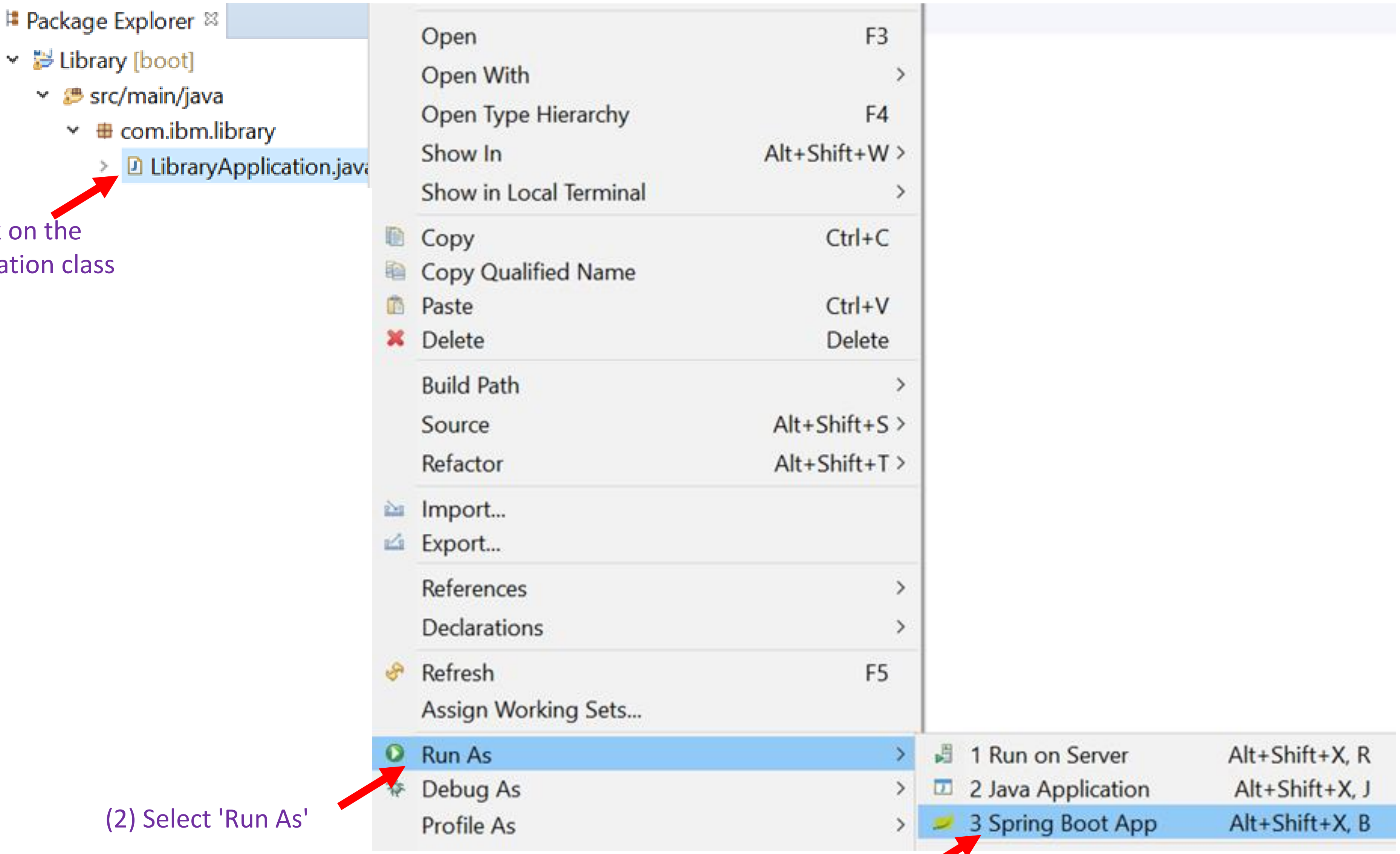
There is no active editor that provides an outline.

0 errors, 1 warning, 0 others

Running the Library Project Application

Running our Spring Boot Application - Slide 1 of 2

When you're ready to run our Library application, do the following:



(1) Right-click on the LibraryApplication class

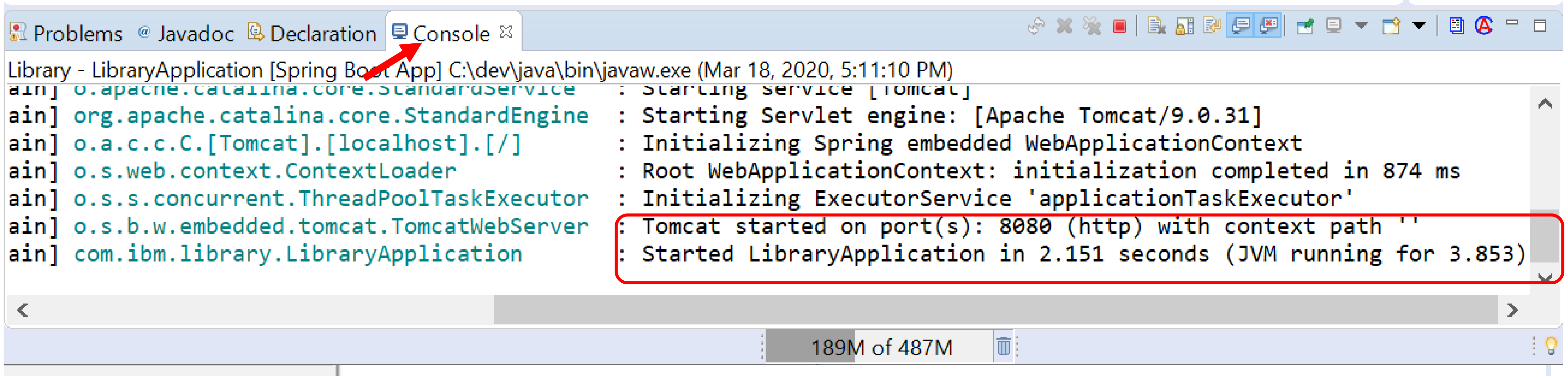
(2) Select 'Run As'

(3) Select 'Spring Boot App'

Running our Spring Boot Application - Slide 2 of 2

After running our Spring Boot Application, make sure that you see in the Console trace the following 2 items:

- Spring Boot started the Tomcat server
- Spring Boot started LibraryApplication



```
Library - LibraryApplication [Spring Boot App] C:\dev\java\bin\javaw.exe (Mar 18, 2020, 5:11:10 PM)
ain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
ain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.31]
ain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
ain] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 874 ms
ain] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
ain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
ain] com.ibm.library.LibraryApplication       : Started LibraryApplication in 2.151 seconds (JVM running for 3.853)
```

Appendix

Appendix

Architectural Layers and Package by Layer vs Package by Feature

- The inbound, service, and repo packages represent **architectural layers**

- They're namespaces for us to organize our Java classes, and they define the kind of classes and code that belong in that layer.

- We'll add the Java class that has to do with the database access into the repo package, Java class that does 'business logic' into the 'service' package, and the Java class that processes the REST request into the 'inbound' package

- The code that we write should not have code that belongs in another architectural layer's class. E.g. we shouldn't any of the code that accesses the database in the Service class's method. Instead, the Service class's method should call a method on the Repo class to get the data from the database

- There's nothing preventing us from putting all our java classes into a single package or from adding code that 'should' be database access code in our classes that are in the 'inbound' or 'service' packages. We're responsible for ensuring that we follow the guidelines of what belongs in an architectural layer (package).

- The architectural layers for this project use the *Package by Layer* concept vs *Package by Feature* concept. We'll revisit this in the class.

- Following is optional reading and is best to read after the project is done. It may be too much conceptual info that is best left to discussion in the class.
- If interested prior to the class, following presents discussions (the following is easier to digest if one has prior experience on Java projects):
  - https://dzone.com/articles/package-by-feature-is-demanded
  - https://dzone.com/articles/package-by-layer-for-spring-projects-is-obsolete