Module 3 Exercise Guide

Lab 3: Building Microservices

Advanced Java Programming

Prerequisites / Getting Started

In this exercise, you will create two microservices to expose a list of phone devices from two brands: Alpha and Beta.

Exercise 1:

Part 1: Create and Import a Spring Boot Application JSP

This lab is designed to give you an introduction to creating a Web application and incorporating some useful functionalities.

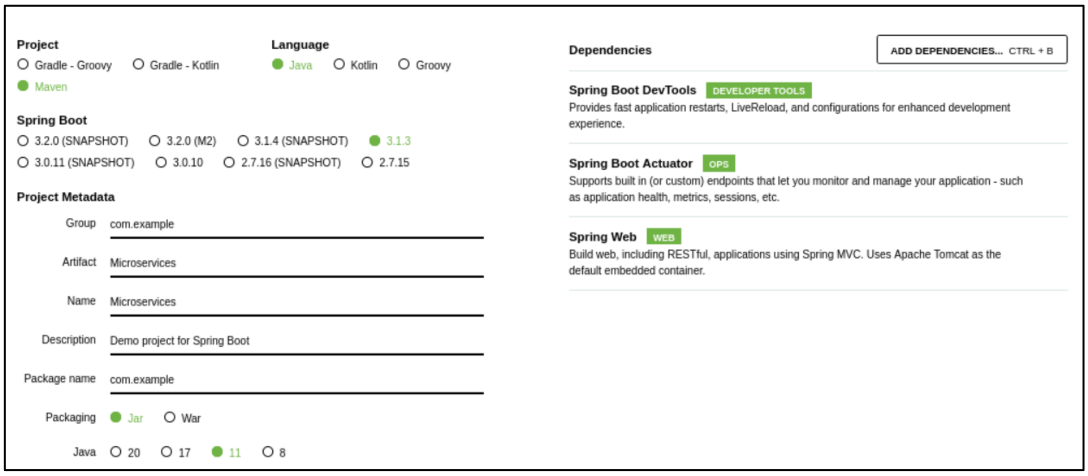
1. Open your lab.
2. On the desktop, double-click the **Firefox Web Browser** icon to open it.

3. When Firefox opens, type **https://start.spring.io/** in the URL and hit **Enter** to go to the Spring webpage.

4. Set the following details for your project:
   a. For Project, select **Maven.** These types are used to generate the project's package structure and naming conventions.
   b. For Group, leave the default of **com.example**.
   c. Change Artifact name to **Microservices.** (Name will automatically default to this).
   d. For Description, enter **microservices**.
   e. For Package name, use **com.example**.
   f. Select **Jar** file for Packaging.
   g. Select **Java** version **11**.
5. Click **Add Dependencies** at the upper right.

6. Select **Spring Web**, then click **Add Dependencies** again.
7. Type **Actuator** into the search bar at the top of the window to find and add **Spring Boot Actuator**.

8. Click **Add Dependencies** again. Type **Dev** in the search bar to find and add **Spring Boot DevTools** as a dependency.



Project
○ Gradle - Groovy ○ Gradle - Kotlin [✓ Maven]

Language
[✓ Java] ○ Kotlin ○ Groovy

Spring Boot
○ 3.2.0 (SNAPSHOT) ○ 3.2.0 (M2) ○ 3.1.4 (SNAPSHOT) [✓ 3.1.3]
○ 3.0.11 (SNAPSHOT) ○ 3.0.10 ○ 2.7.16 (SNAPSHOT) ○ 2.7.15

Project Metadata

Group	com.example
Artifact	Microservices
Name	Microservices
Description	Demo project for Spring Boot
Package name	com.example
Packaging	[✓ Jar] ○ War
Java	○ 20 ○ 17 [✓ 11] ○ 8

Dependencies

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

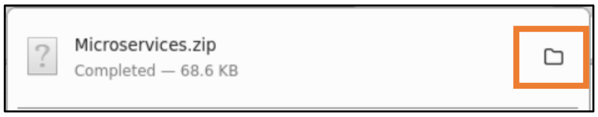
Spring Boot Actuator OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

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

9. Click **Generate** at the bottom of the screen.

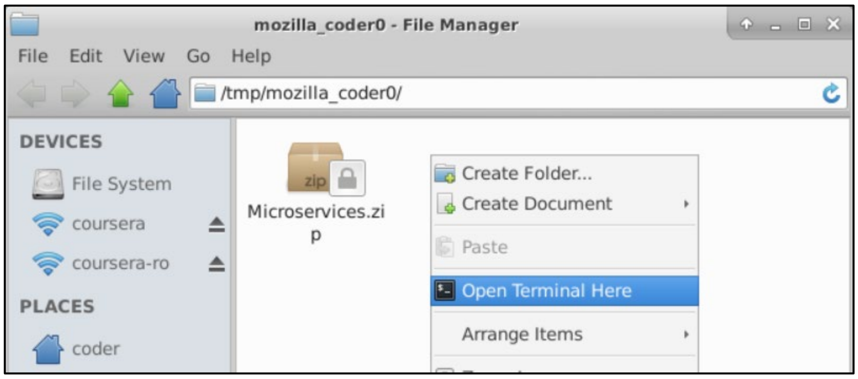
10. Select **Open with** and leave the default of **unzip**. Click **OK**.

11. In the upper right of the browser, click the **downloads** icon and click the **folder** next to your download to open the file location.



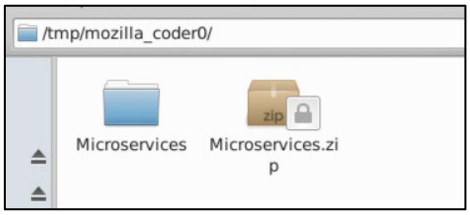
Microservices.zip
Completed — 68.6 KB

12. You should see your zip file in the _coder) folder. Right click in the white space in the window and select **Open Terminal here**.



13. Inside the terminal, type `unzip Microservices.zip` and hit **Enter**.

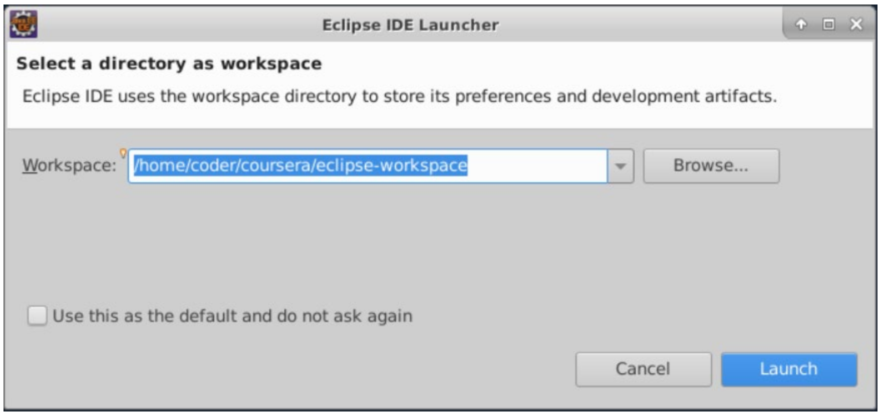
14. Your folder unzips to the location.



15. **Close** all open windows.

16. Now we need to import the project to our IDE in Eclipse as a Maven project. On the desktop, double-click the **Eclipse** icon to open it.

17. In the Eclipse IDE Launcher window, click **Launch** to select the default workspace.

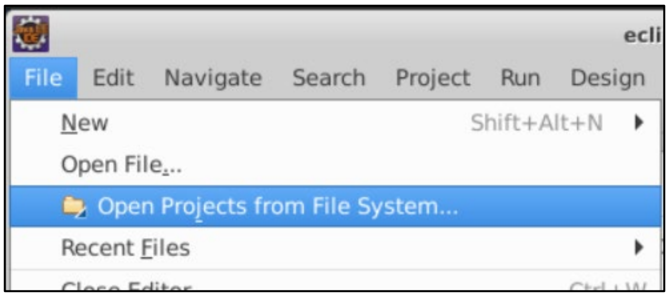


Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: /home/coder/coursera/eclipse-workspace

☐ Use this as the default and do not ask again

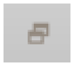
18. When Eclipse opens, click **File > Open Projects from File System...**



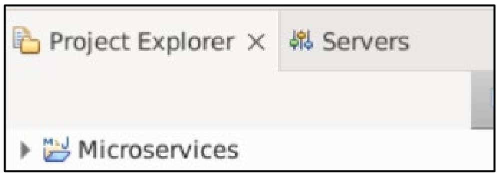
19. In the Import Projects window, click **Directory...** at the upper right.

20. Select the **Microservices** folder and click **Open**. (If you don't see the zip file, click Recent at the top of the left sidebar).

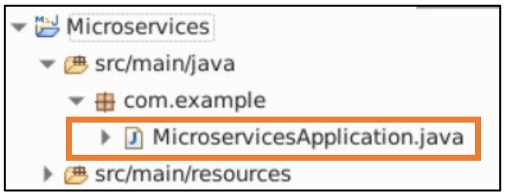
21. Click **Finish** in the Import window.

22. At the far left of your screen click the **Restore icon**  to open the Project Explorer pane (if it isn't already opened).

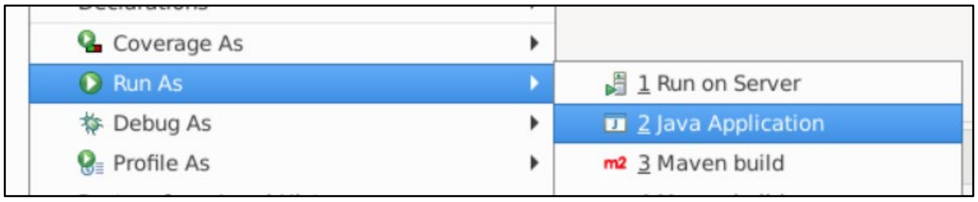
23. You will see your Microservices project in the Project Explorer pane.



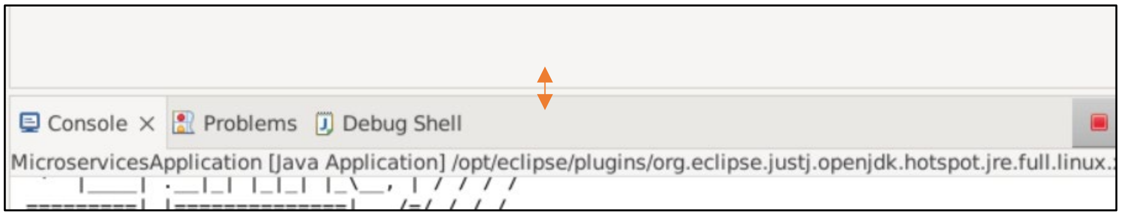
24. Expand **Microservices > src/main/java > com.example**. In the com.example package, you will see the MicroservicesApplication.java file.



25. Right-click the **MicroservicesApplication.java** file and select **Run As > Java Application**.

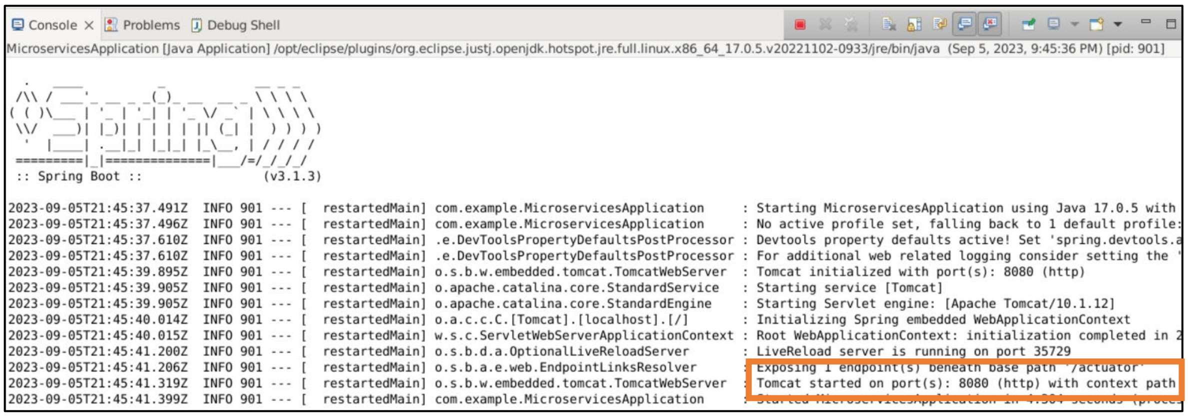


26. At the bottom of your screen, expand the Console pane by dragging the dividing line.



```
MicroservicesApplication [Java Application] /opt/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.:
```

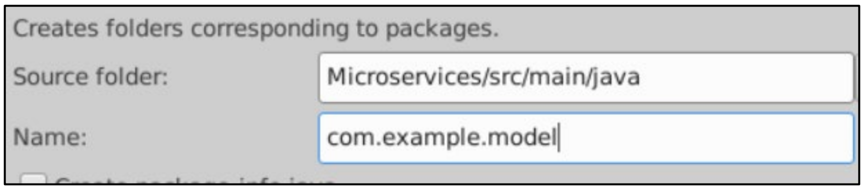
27. In the Console window look for confirmation that Tomcat started on port 8080:



```
MicroservicesApplication [Java Application] /opt/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.5.v20221102-0933/jre/bin/java  (Sep 5, 2023, 9:45:36 PM) [pid: 901]

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.1.3)

2023-09-05T21:45:37.491Z  INFO 901 --- [  restartedMain] com.example.MicroservicesApplication     : Starting MicroservicesApplication using Java 17.0.5 with
2023-09-05T21:45:37.496Z  INFO 901 --- [  restartedMain] com.example.MicroservicesApplication     : No active profile set, falling back to 1 default profile:
2023-09-05T21:45:37.610Z  INFO 901 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.a
2023-09-05T21:45:37.610Z  INFO 901 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the '
2023-09-05T21:45:39.895Z  INFO 901 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2023-09-05T21:45:39.905Z  INFO 901 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-09-05T21:45:39.905Z  INFO 901 --- [  restartedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.12]
2023-09-05T21:45:40.014Z  INFO 901 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-09-05T21:45:40.015Z  INFO 901 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2
2023-09-05T21:45:41.200Z  INFO 901 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2023-09-05T21:45:41.206Z  INFO 901 --- [  restartedMain] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 1 endpoint(s) beneath base path '/actuator'
2023-09-05T21:45:41.319Z  INFO 901 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path
2023-09-05T21:45:41.399Z  INFO 901 --- [  restartedMain] com.example.MicroservicesApplication     : Started MicroservicesApplication in 4.304 seconds (proce
```

Part 2: Create Packages and Classes

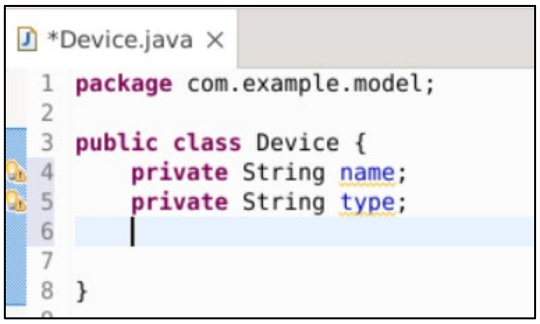
1. Minimize or drag the divider to shrink the Console pane.
2. In the Project Explorer pane at the left, right-click **com.example** and select **New > package**.
3. Name the package **com.example.model** and click **Finish.**



Creates folders corresponding to packages.
Source folder: Microservices/src/main/java
Name: com.example.model

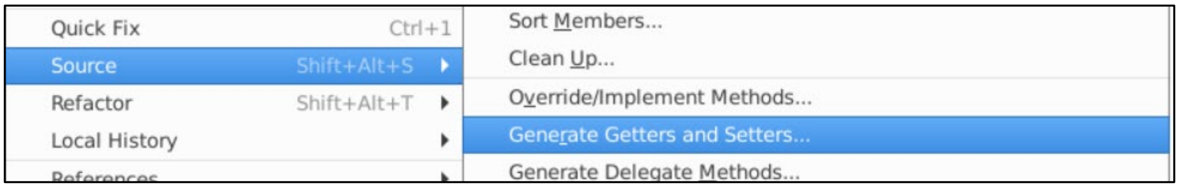
4. In the Project Explorer pane, right-click your new **com.example.model** package and select **New > Class**.
5. Name the class **Device** and click **Finish**.
6. The Device.java file opens. Add the following attributes on the line after the public class:

```
private String name;
private String type;
```



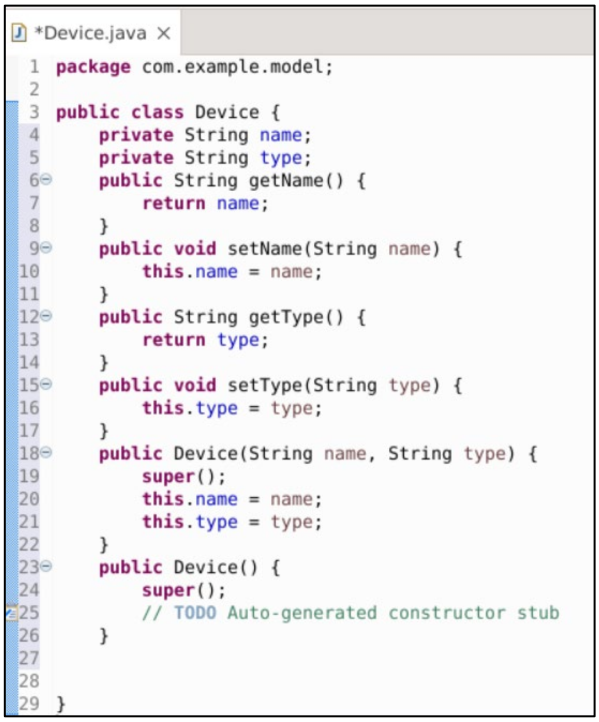
```
package com.example.model;

public class Device {
    private String name;
    private String type;
    

}
```

7. Add a blank line or two, then add the setters and getters by right-clicking on the blank line, selecting **Source > Generate Getters and Setters…**



8. Click **Select All** to select the attributes, then click **Generate**. Your Getters and Setters are added to the code automatically from the fields we selected.
9. Now let's add the Constructor. Right-click on the blank line at the end of the Getters and Setters, and select **Source > Generate Constructor using Fields…**
10. Ensure the two attributes are selected, then click **Generate**. Your Constructor is added automatically to the code.
11. Right-click on the next blank line and select **Source > Generate Constructors from Superclass…**
12. Ensure **Object** is selected and click **Generate**. The superclass constructor is added.

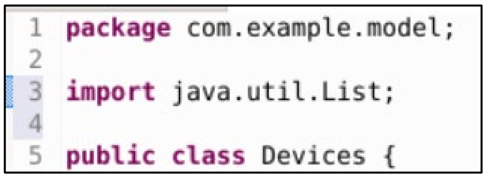
13. Your final code should look like the following:


*Device.java

```java
package com.example.model;

public class Device {
    private String name;
    private String type;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public Device(String name, String type) {
        super();
        this.name = name;
        this.type = type;
    }
    public Device() {
        super();
        // TODO Auto-generated constructor stub
    }


}
```

14. Click the **Save** icon on the toolbar at the upper left to save your changes.
15. Now we will create the Devices model class. Right-click on the **com.example.model** package and select **New > Class**.
16. Name it **Devices** and click **Finish**. (Note: this is *Devices*, not to be confused with the *Device* class that we created earlier).
17. Devices.java opens. Go to line 2 and hit **Enter**. Then, add the following to import the List utility:

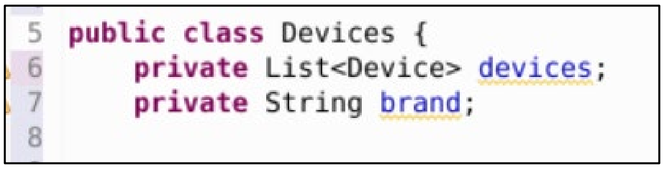
    ```
    import java.util.List;
    ```



```java
package com.example.model;

import java.util.List;

public class Devices {
```

18. Add the following code to the Devices class in the java file, then hit **Enter** a few times:

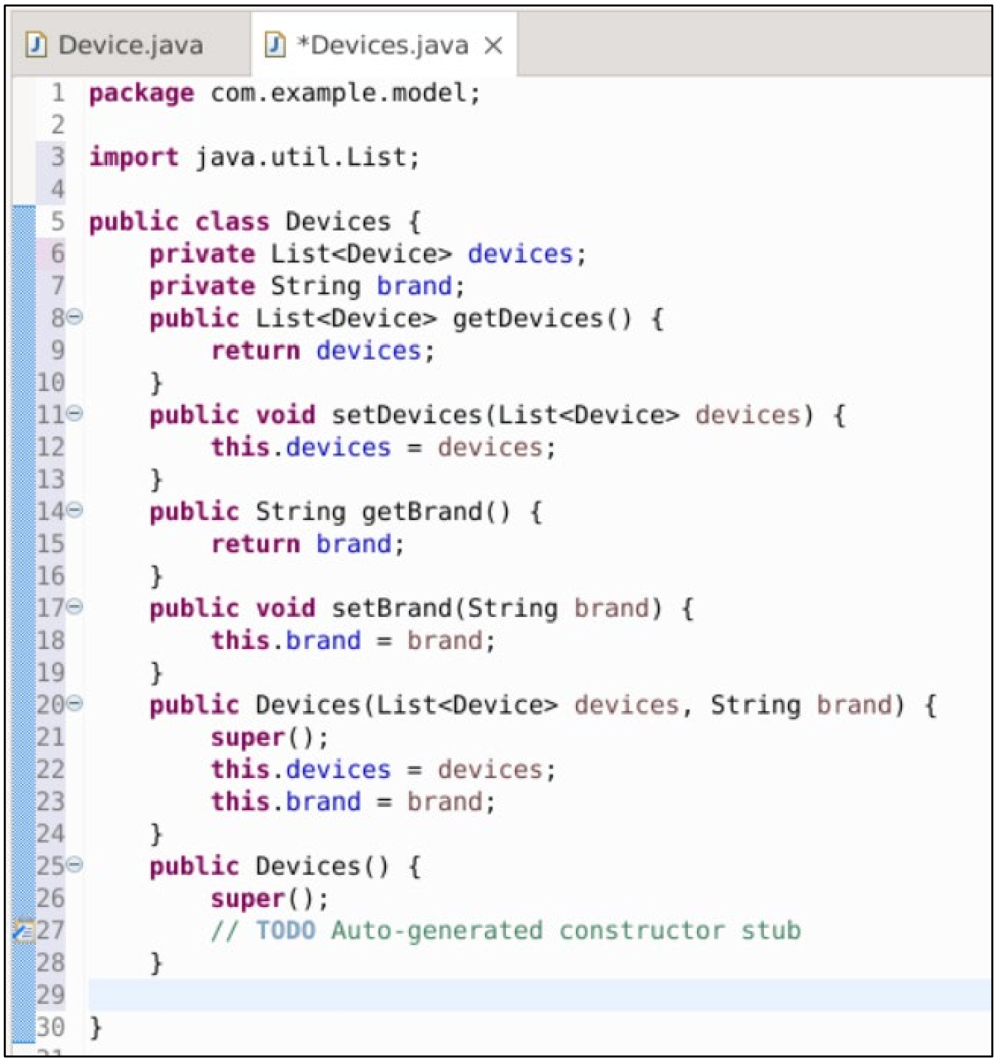
    ```
    private List<Device> devices;
    private String brand;
    ```



```java
public class Devices {
    private List<Device> devices;
    private String brand;

```

We will use the brand attribute later to create two brands: Alpha and Beta.

19. Right-click on one of the blank lines before the closing bracket and select **Source > Generate Getters and Setters…** Click **Select All**, then click **Generate** to automatically add the Setters and Getters to your code.
20. Add the Constructor by right-clicking on the blank line at the end of the Getters and Setters, and select **Source > Generate Constructor using Fields…** Ensure the two attributes are selected, then click **Generate**. Your Constructor is added automatically to the code.
21. Right-click on the next blank line and select **Source > Generate Constructors from Superclass…**
22. Ensure **Object** is selected and click **Generate.** The superclass constructor is added.
23. Your final code should look like the following:

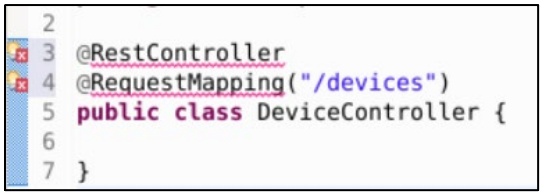


```
package com.example.model;

import java.util.List;

public class Devices {
    private List<Device> devices;
    private String brand;
    public List<Device> getDevices() {
        return devices;
    }
    public void setDevices(List<Device> devices) {
        this.devices = devices;
    }
    public String getBrand() {
        return brand;
    }
    public void setBrand(String brand) {
        this.brand = brand;
    }
    public Devices(List<Device> devices, String brand) {
        super();
        this.devices = devices;
        this.brand = brand;
    }
    public Devices() {
        super();
        // TODO Auto-generated constructor stub
    }

}
```

24. Now let's create a controller package. Right-click on **com.example** in the Project Explorer pane and select **New > Package**. Name it **com.example.controller** and click **Finish**.
25. Right-click on your new controller package and select **New > Class**. Name it **DeviceController** and click **Finish**.

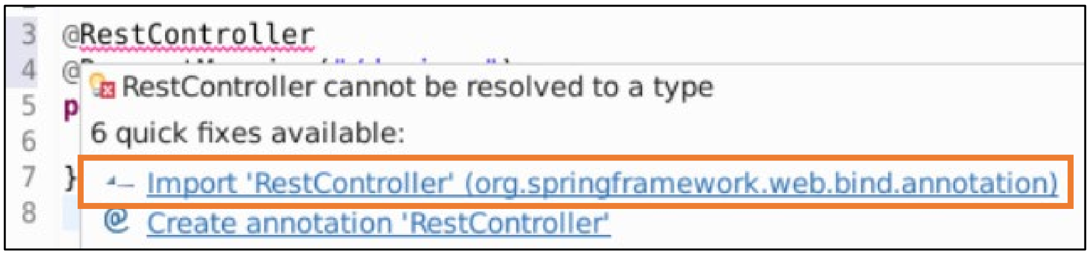
26. The DeviceController class java file opens. Above the class declaration, add the following:

```
@RestController
@RequestMapping("/devices")
```



```
@RestController
@RequestMapping("/devices")
public class DeviceController {

}
```

27. You'll notice there are errors on these lines. Hover over @RestController and select the **Import 'RestController'** option.



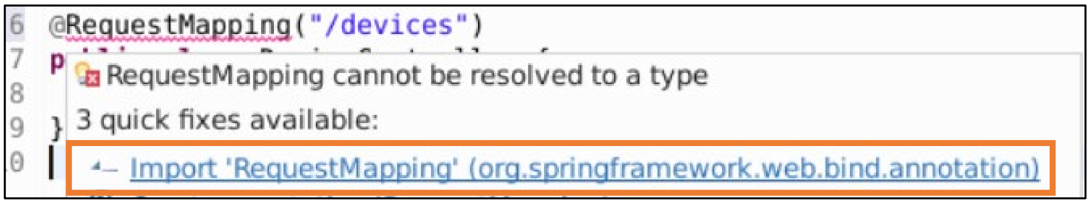
```
@RestController
```

RestController cannot be resolved to a type

6 quick fixes available:

Import 'RestController' (org.springframework.web.bind.annotation)
Create annotation 'RestController'

28. Hover over @RequestMapping and select the **Import RequestMapping** option.



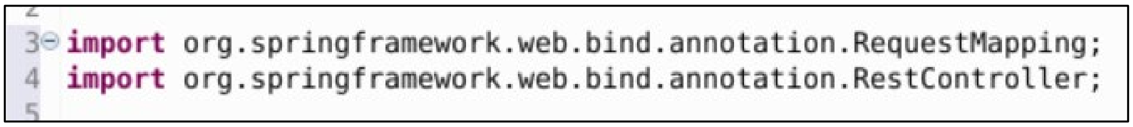
```
@RequestMapping("/devices")
```

RequestMapping cannot be resolved to a type

3 quick fixes available:

Import 'RequestMapping' (org.springframework.web.bind.annotation)

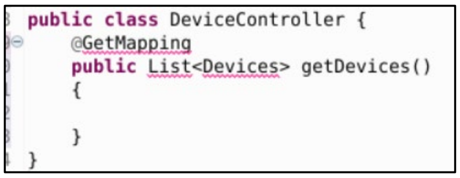
29. The import lines are added automatically towards the top of the page:



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

30. Inside the DeviceController class, create a getDevices() annotated with @GetMapping, and return List<Devices> as follows:

```
@GetMapping
Public List<Devices> getDevices()
{
}
```

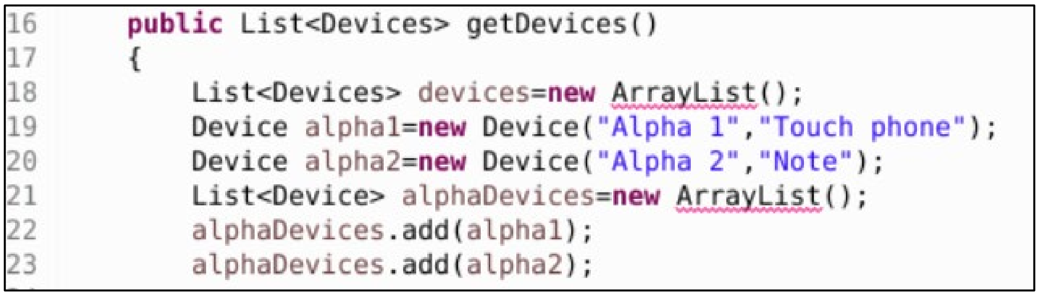


```
public class DeviceController {
    @GetMapping
    public List<Devices> getDevices()
    {

    }
}
```

31. Hover over **GetMapping**, **List**, and <**Devices**> and select the import options. The import syntax will be automatically added towards the beginning of the page.

32. Now, we will create two lists of devices inside the List, an Alpha and Beta list. First, let's create the Alpha list and add two devices to it:

```
List<Devices> devices=new ArrayList();
Device alpha1=new Device("Alpha 1","Touch phone");
Device alpha2=new Device("Alpha 2","Note");
List<Device> alphaDevices=new ArrayList();
alphaDevices.add(alpha1);
alphaDevices.add(alpha2);
```

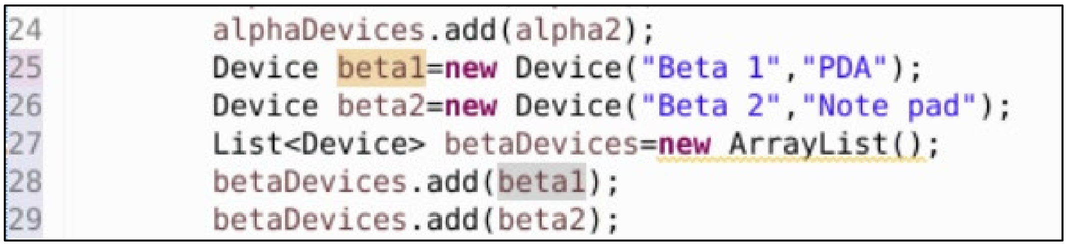


```
    public List<Devices> getDevices()
    {
        List<Devices> devices=new ArrayList();
        Device alpha1=new Device("Alpha 1","Touch phone");
        Device alpha2=new Device("Alpha 2","Note");
        List<Device> alphaDevices=new ArrayList();
        alphaDevices.add(alpha1);
        alphaDevices.add(alpha2);
```

33. Hover over **ArrayList**, and select the **Import** option.

34. Now, we will create the Beta list and add two devices to it:

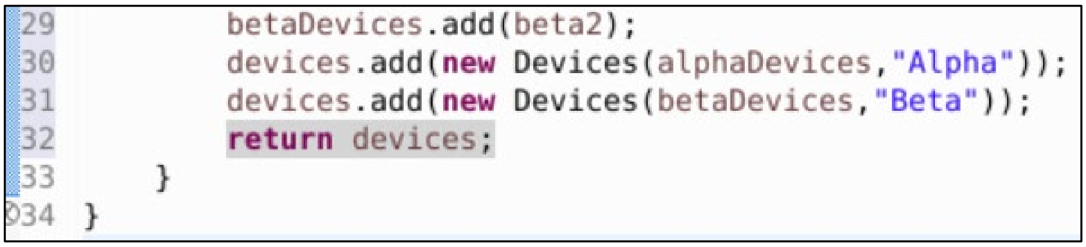
```
Device beta1=new Device("Beta 1","PDA");
Device beta2=new Device("Beta 2","Note pad");
List<Device> betaDevices=new ArrayList();
betaDevices.add(beta1);
betaDevices.add(beta2);
```



```
        alphaDevices.add(alpha2);
        Device beta1=new Device("Beta 1","PDA");
        Device beta2=new Device("Beta 2","Note pad");
        List<Device> betaDevices=new ArrayList();
        betaDevices.add(beta1);
        betaDevices.add(beta2);
```

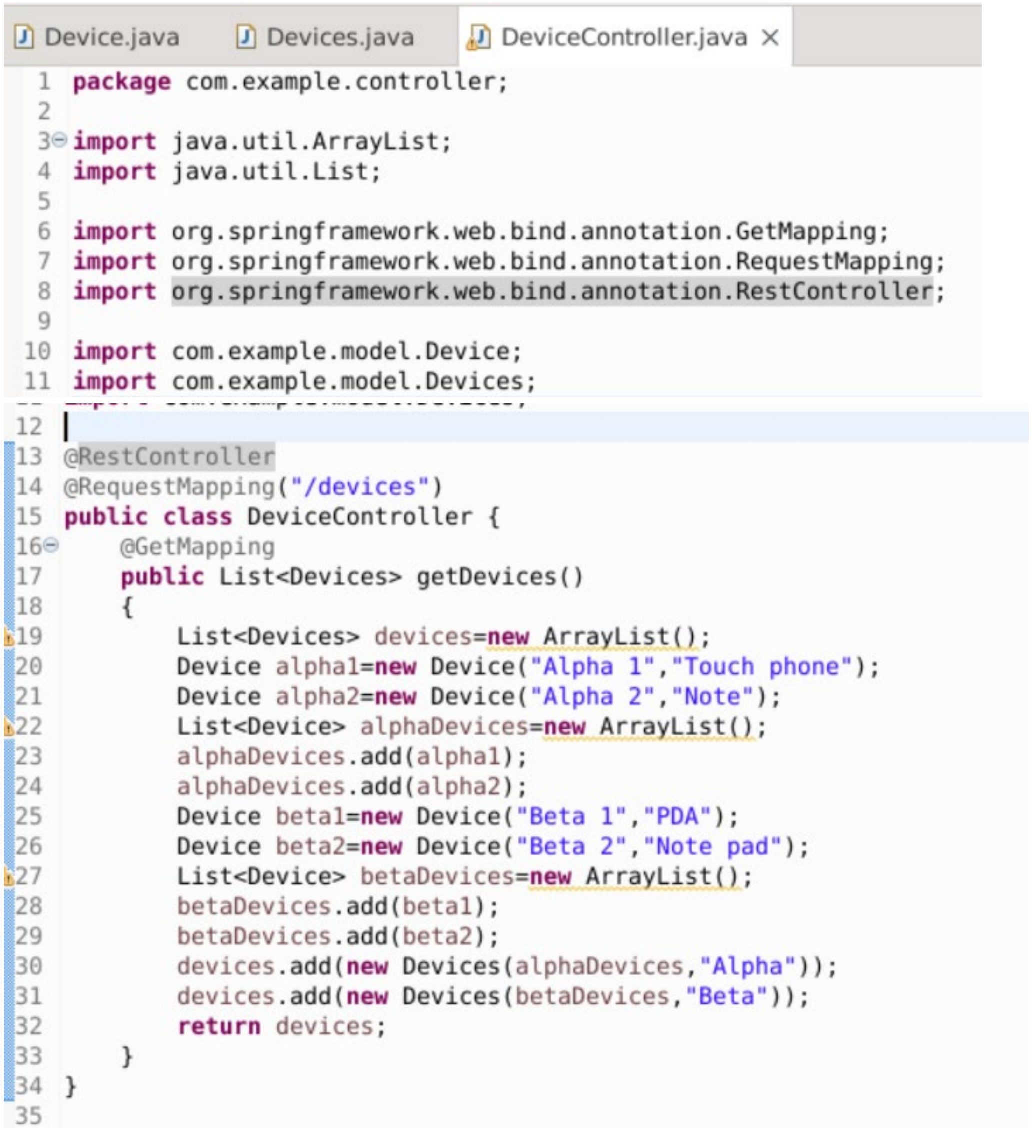
35. Add alphaDevices and betaDevices to the list and return the devices:

```
devices.add(new Devices(alphaDevices,"Alpha"));
devices.add(new Devices(betaDevices,"Beta"));
return devices;
```



```
        betaDevices.add(beta2);
        devices.add(new Devices(alphaDevices,"Alpha"));
        devices.add(new Devices(betaDevices,"Beta"));
        return devices;
    }
}
```

36. Your final code should look like the following:

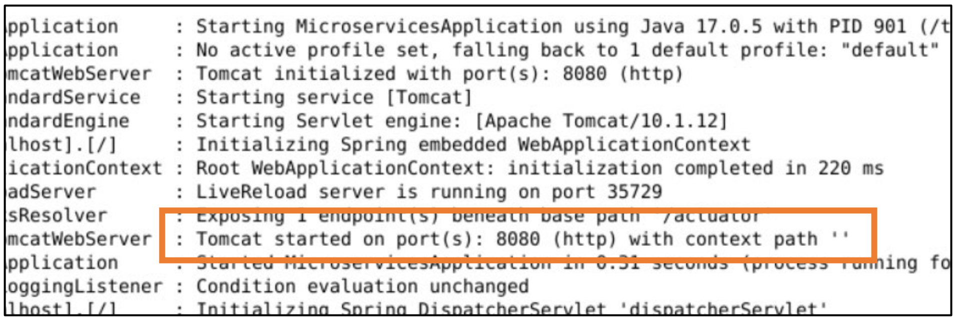


```java
package com.example.controller;

import java.util.ArrayList;
import java.util.List;

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

import com.example.model.Device;
import com.example.model.Devices;

@RestController
@RequestMapping("/devices")
public class DeviceController {
    @GetMapping
    public List<Devices> getDevices()
    {
        List<Devices> devices=new ArrayList();
        Device alpha1=new Device("Alpha 1","Touch phone");
        Device alpha2=new Device("Alpha 2","Note");
        List<Device> alphaDevices=new ArrayList();
        alphaDevices.add(alpha1);
        alphaDevices.add(alpha2);
        Device beta1=new Device("Beta 1","PDA");
        Device beta2=new Device("Beta 2","Note pad");
        List<Device> betaDevices=new ArrayList();
        betaDevices.add(beta1);
        betaDevices.add(beta2);
        devices.add(new Devices(alphaDevices,"Alpha"));
        devices.add(new Devices(betaDevices,"Beta"));
        return devices;
    }
}
```

37. Click **Save All** in the toolbar to save all your changes.

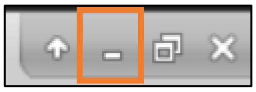
38. In your Project Explorer, right-click on **MicroservicesApplication.java** and select **Run As > Java Application**.

39. Check your console logs to ensure Tomcat started on port 8080:



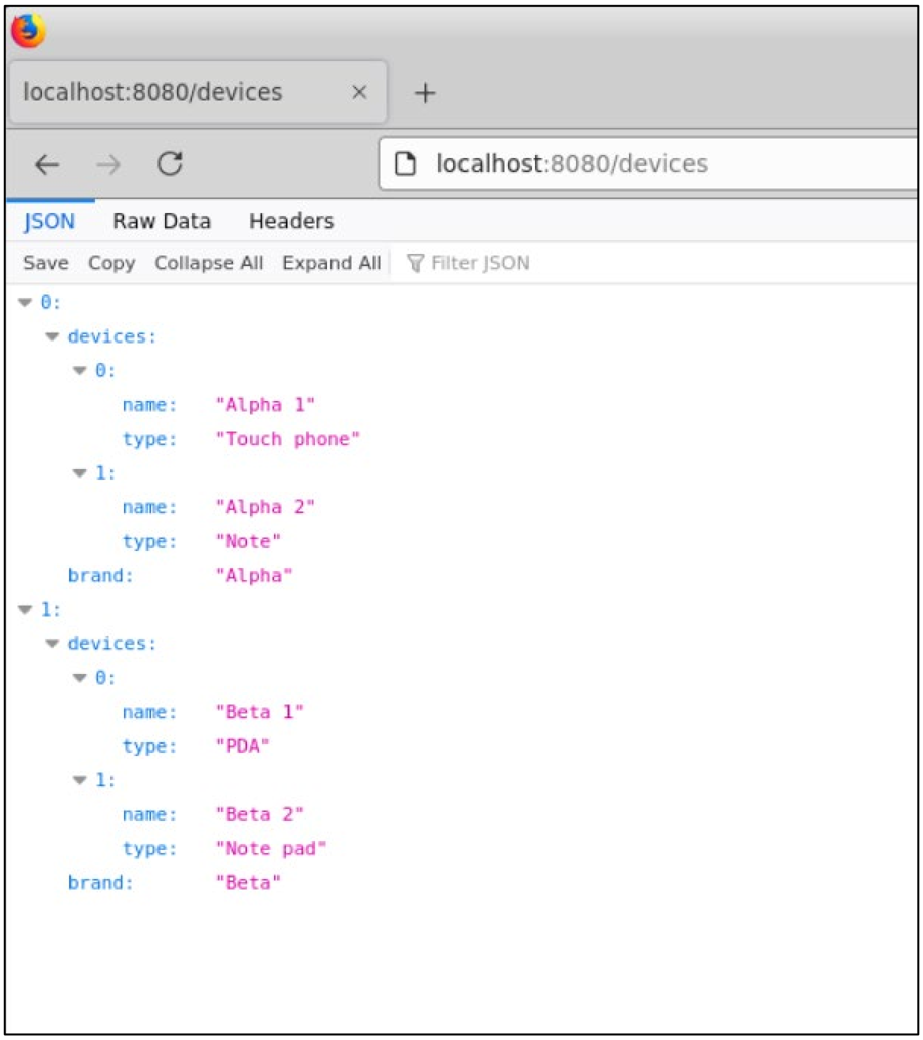
```
pplication     : Starting MicroservicesApplication using Java 17.0.5 with PID 901 (/t
pplication     : No active profile set, falling back to 1 default profile: "default"
mcatWebServer  : Tomcat initialized with port(s): 8080 (http)
ndardService   : Starting service [Tomcat]
ndardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.12]
lhost].[/]     : Initializing Spring embedded WebApplicationContext
icationContext : Root WebApplicationContext: initialization completed in 220 ms
adServer       : LiveReload server is running on port 35729
sResolver      : Exposing 1 endpoint(s) beneath base path '/actuator'
mcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
pplication     : Started MicroservicesApplication in 0.31 seconds (process running fo
oggingListener : Condition evaluation unchanged
lhost].[/]     : Initializing Spring DispatcherServlet 'dispatcherServlet'
```

40. Minimize your Eclipse window by clicking the **Minimize** icon in the upper right corner.



41. Open **Firefox** and type **http://localhost:8080/devices** into the URL and click **Enter**.

42. You should see a response containing the list of devices:



```
▼ 0:
  ▼ devices:
    ▼ 0:
        name:    "Alpha 1"
        type:    "Touch phone"
    ▼ 1:
        name:    "Alpha 2"
        type:    "Note"
    brand:       "Alpha"
▼ 1:
  ▼ devices:
    ▼ 0:
        name:    "Beta 1"
        type:    "PDA"
    ▼ 1:
        name:    "Beta 2"
        type:    "Note pad"
    brand:       "Beta"
```

***End of lab