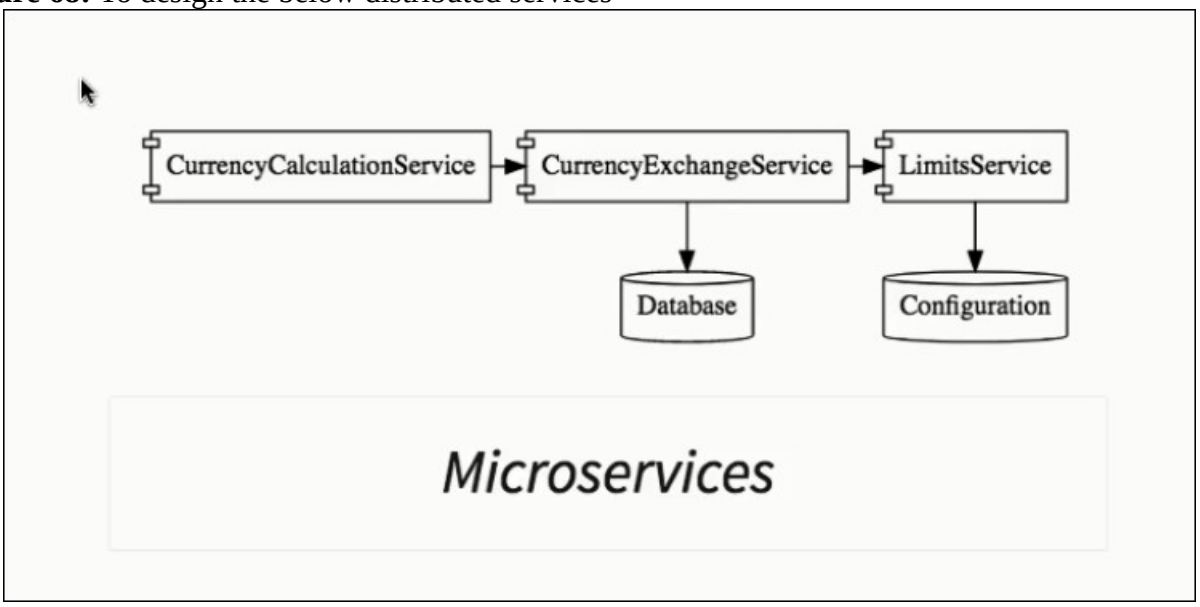
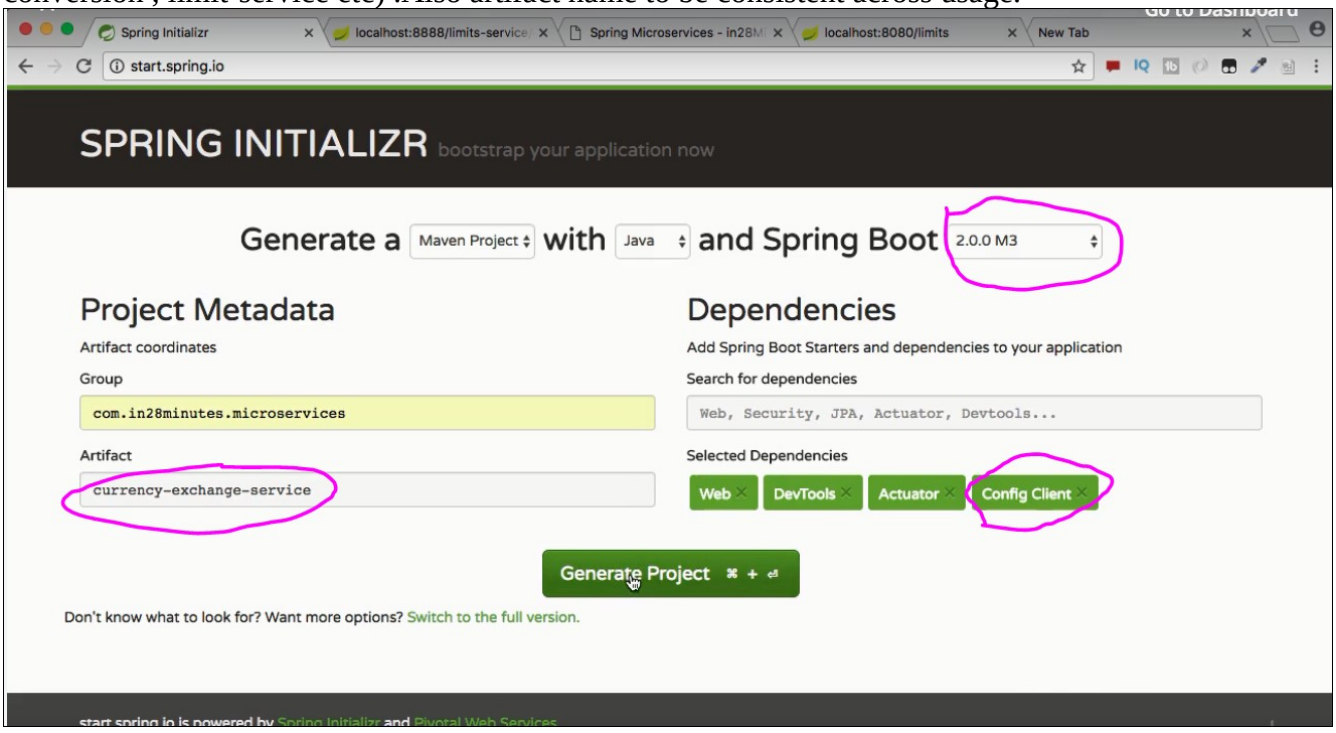
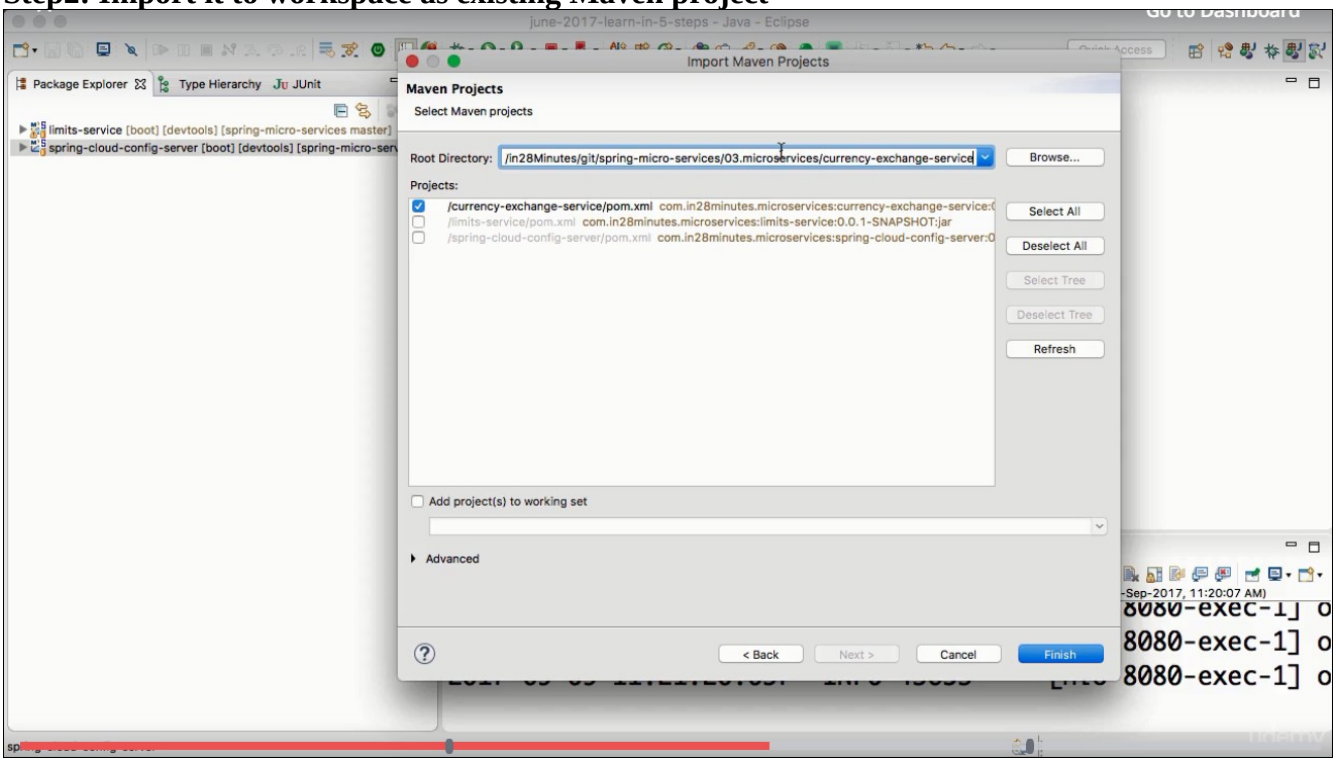
Lecture 68: To design the below distributed services



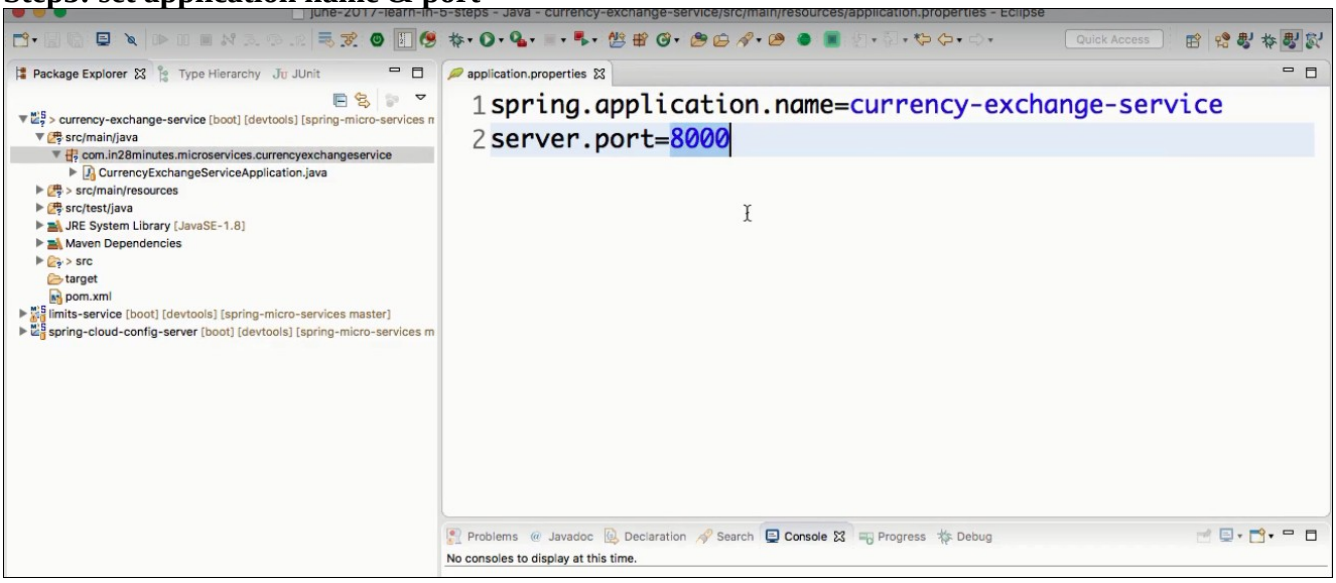
Lecture 69,70,71:Setup currency-exchange-service
Action1: Create currency-exchange-service
Step1: Download the configured springsboot project
****Note:** we need **Config Client** here & Spring-Boot version same as other services (currency-conversion , limit-service etc) .Also artifact name to be consistent across usage.



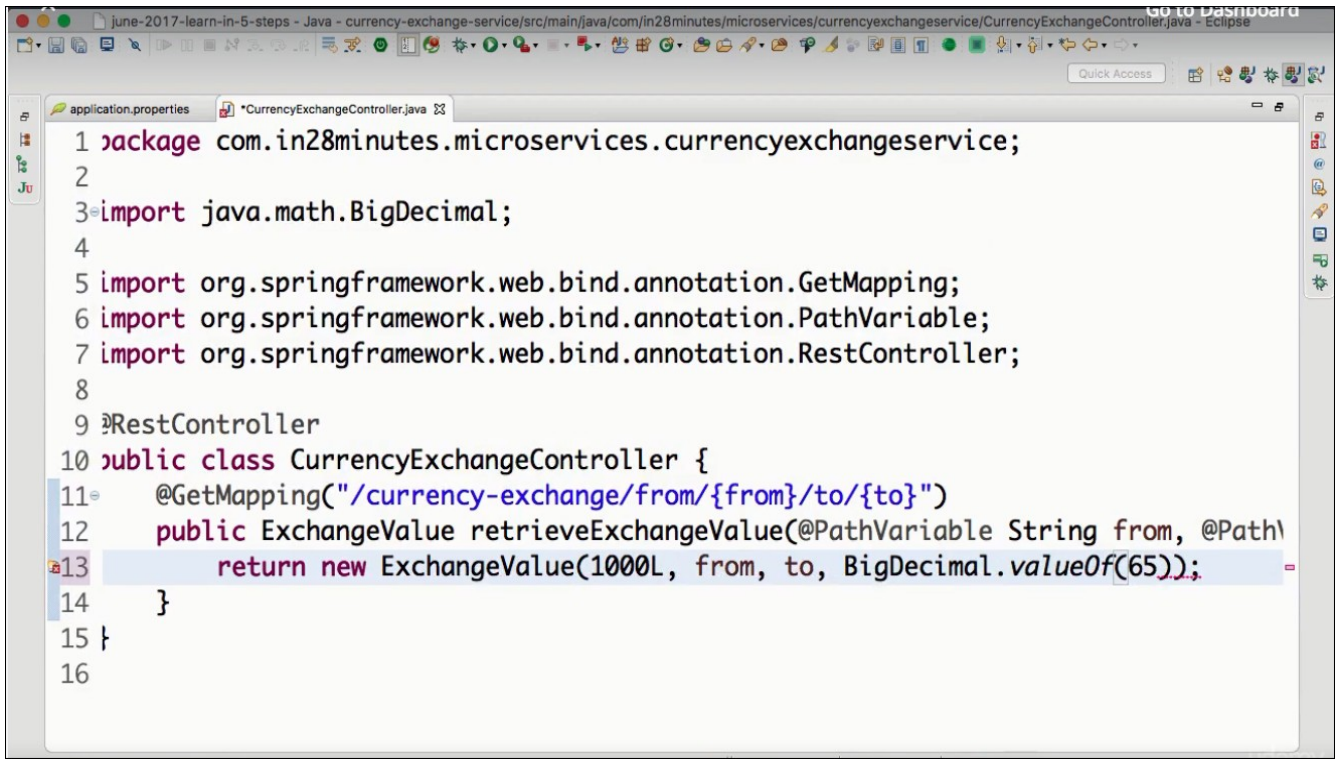
Step2: Import it to workspace as existing Maven project



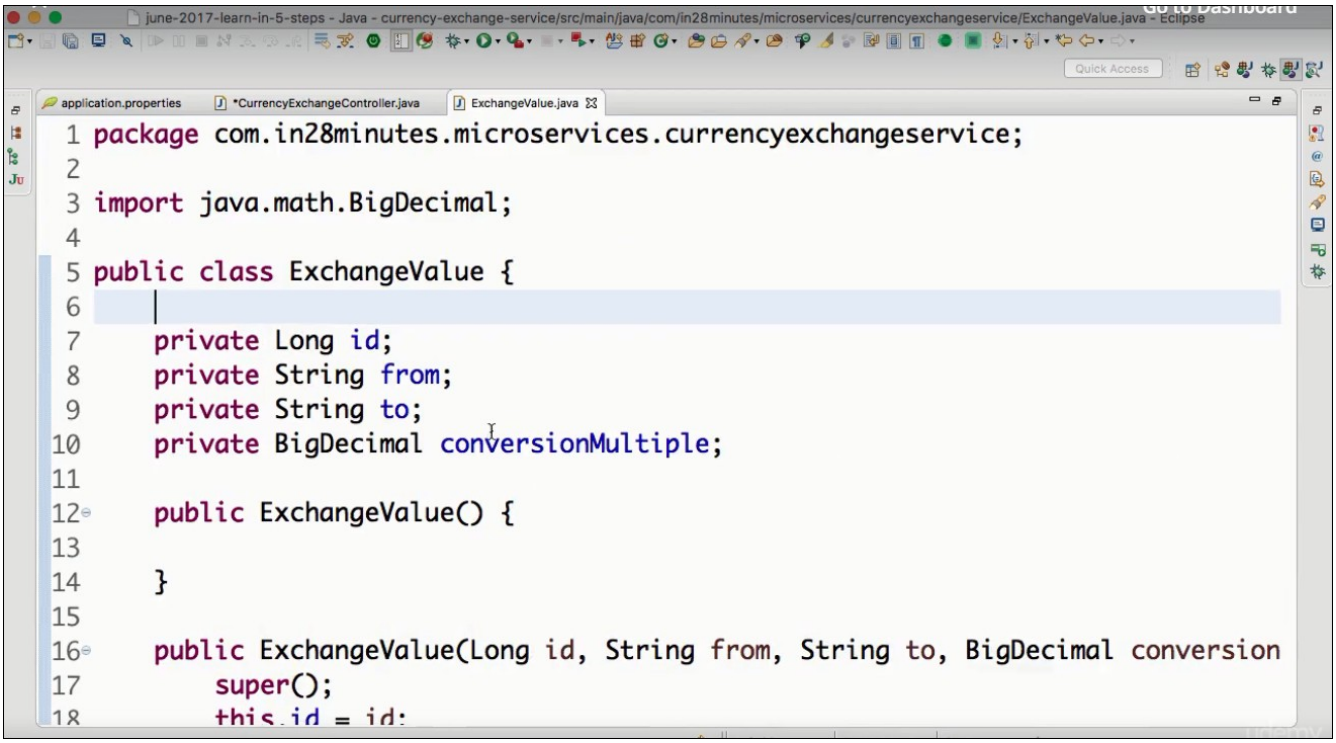
Step3: set application name & port



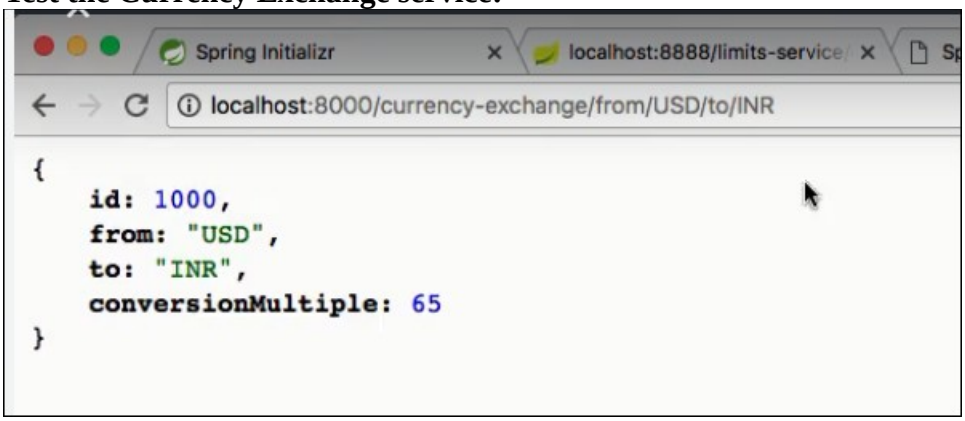
Step4: Add currency controller functionality
Define controller:



Define Bean:

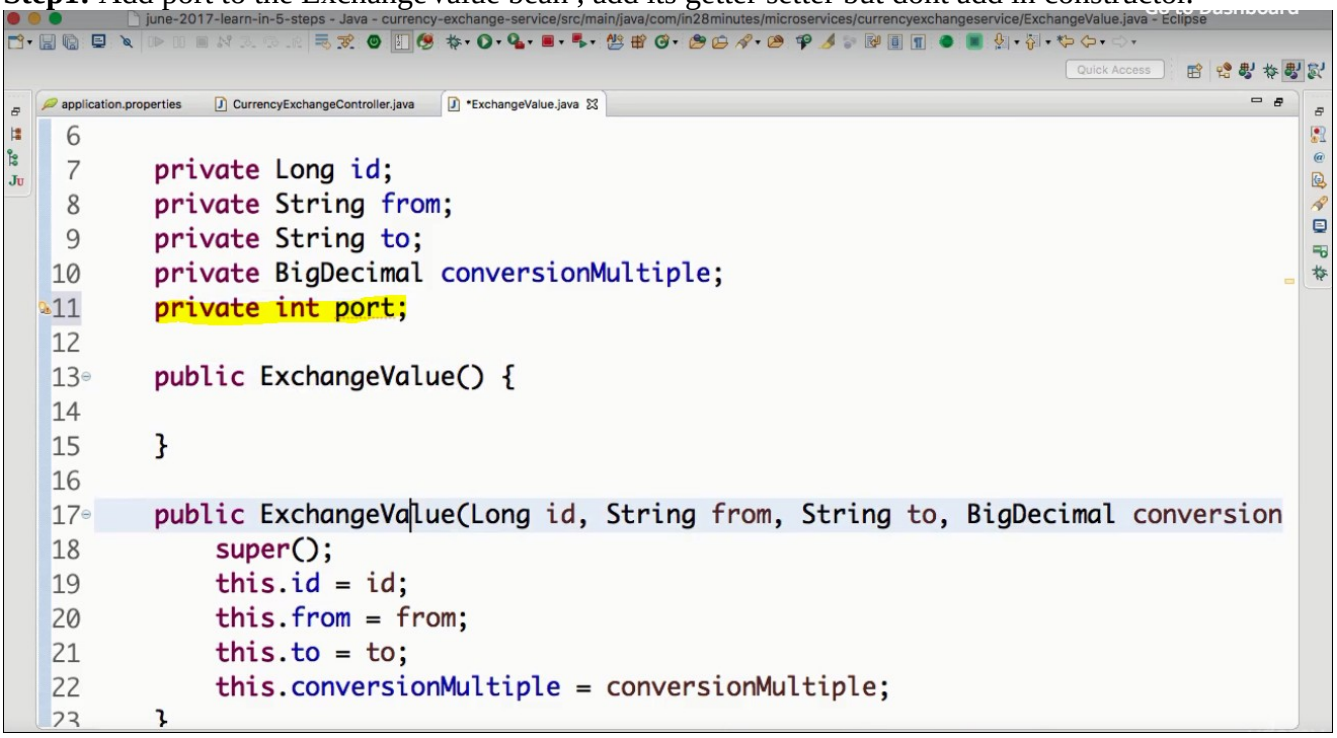


Test the Currency Exchange service:

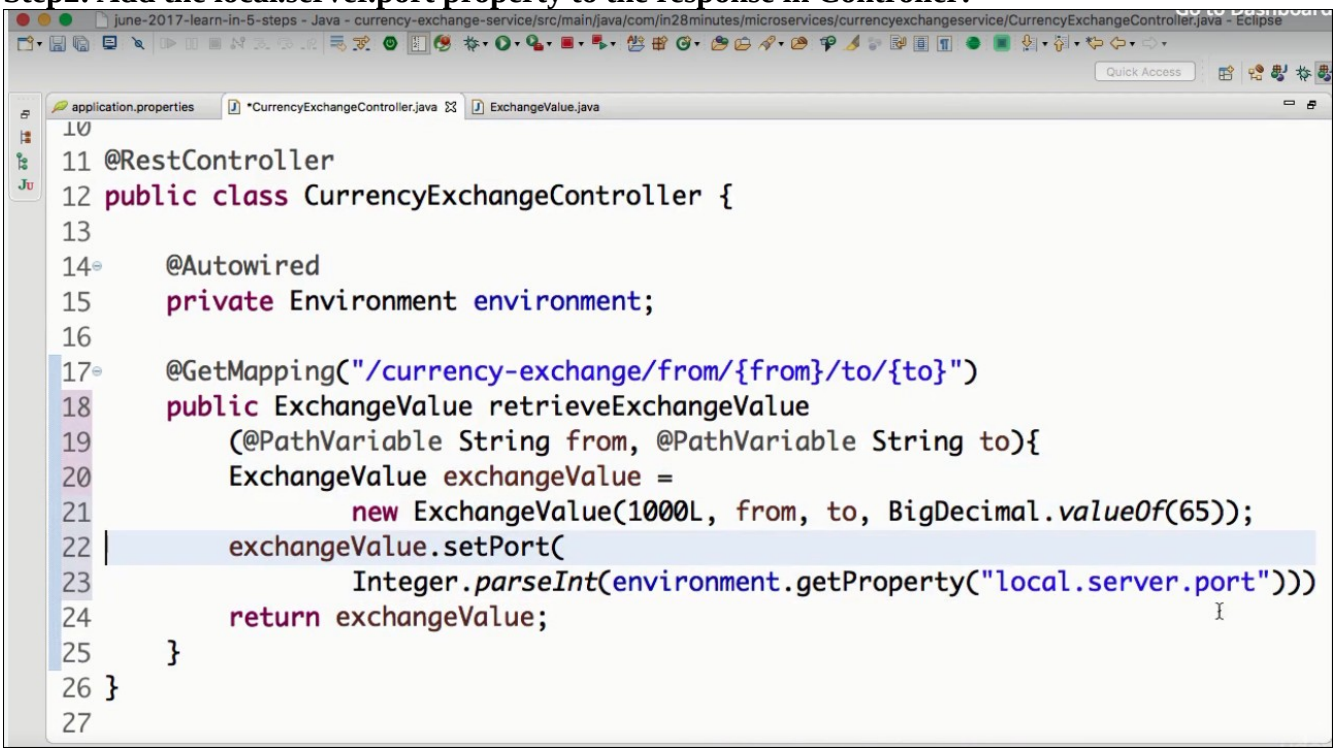


Action2: setup dynamic port in the response of currency-exchange-service

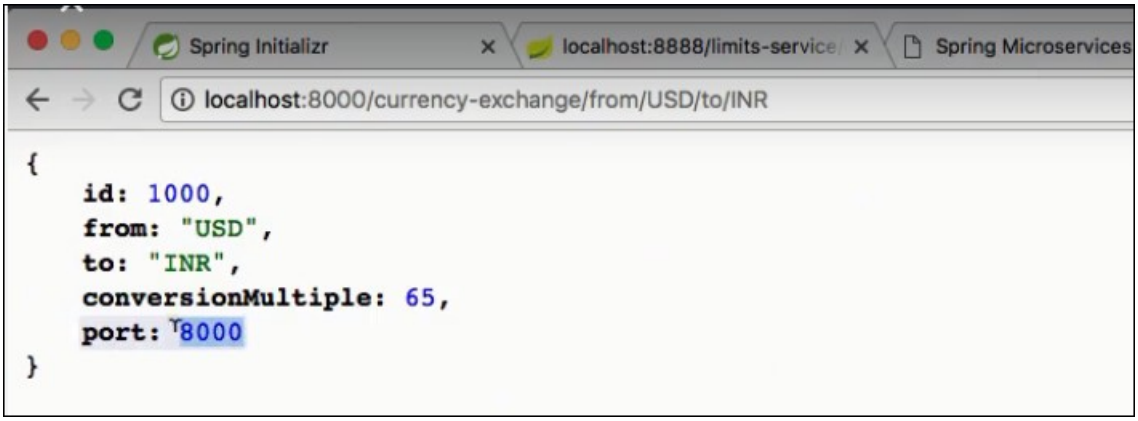
Step1: Add port to the ExchangeValue bean , add its getter setter but dont add in constructor.



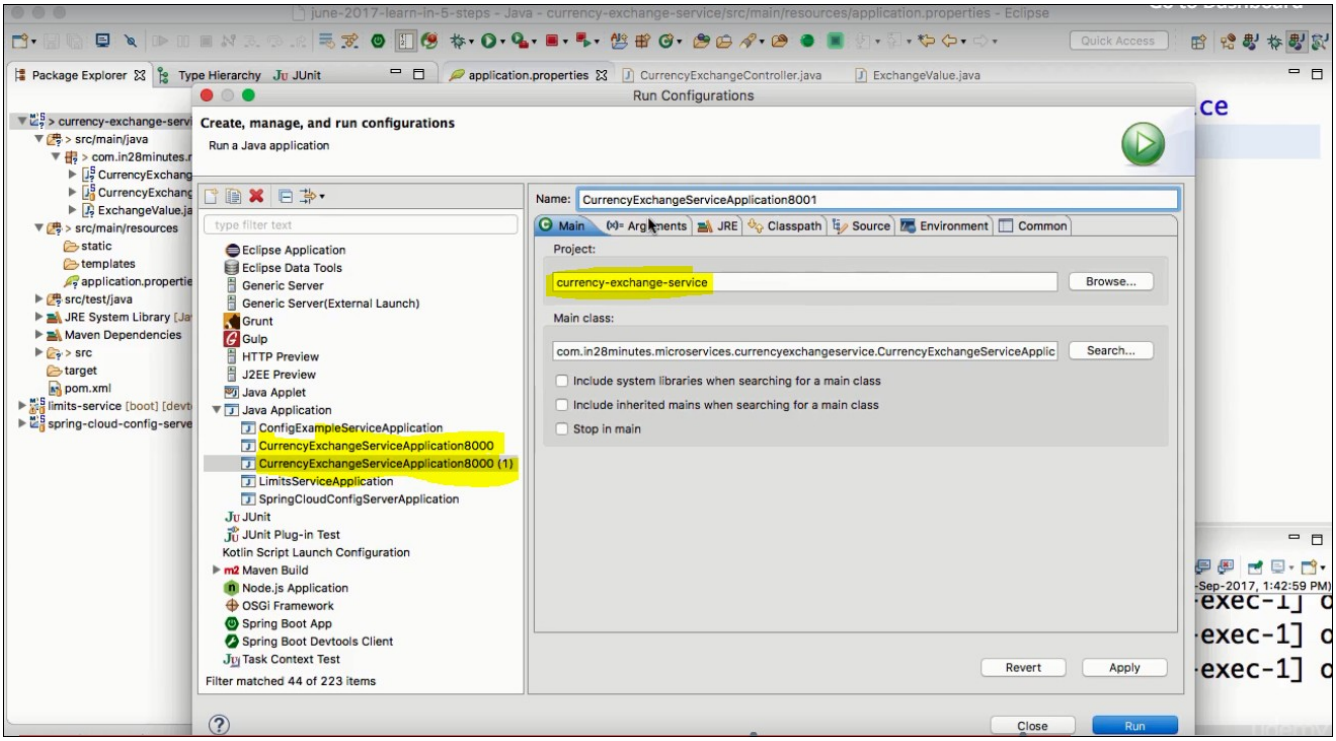
Step2: Add the local.server.port property to the response in Controller:



Step3: Test to see the port value in response

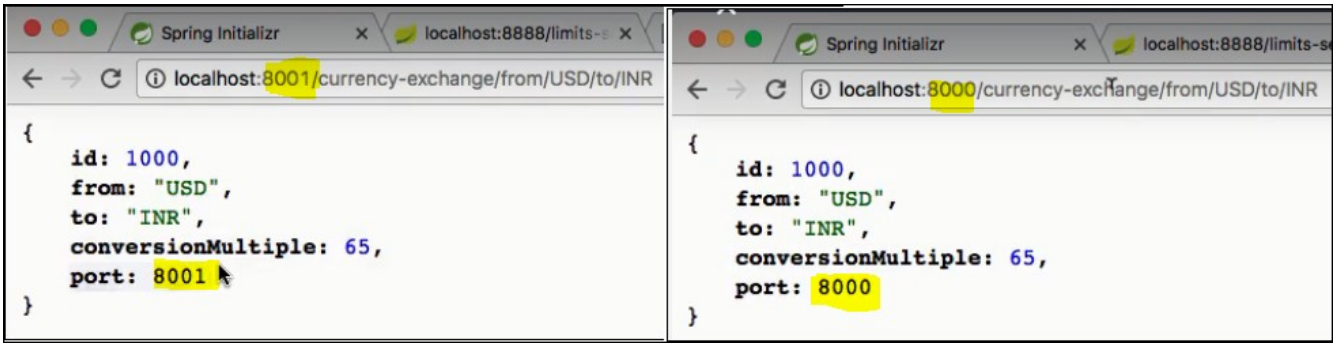


Step4: To run this application on different port create separate run configurations.



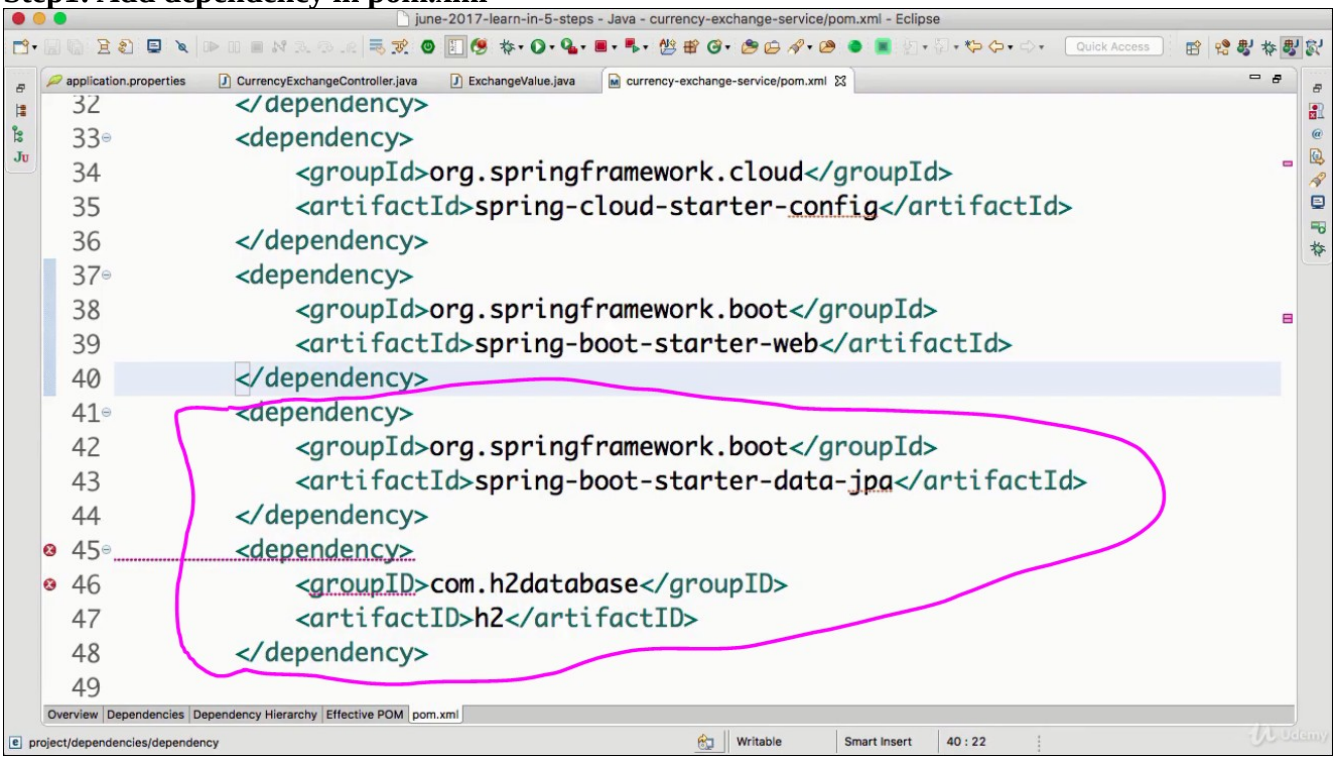
Pass the VM Arguments as shown below:

Verify the response on both the ports:

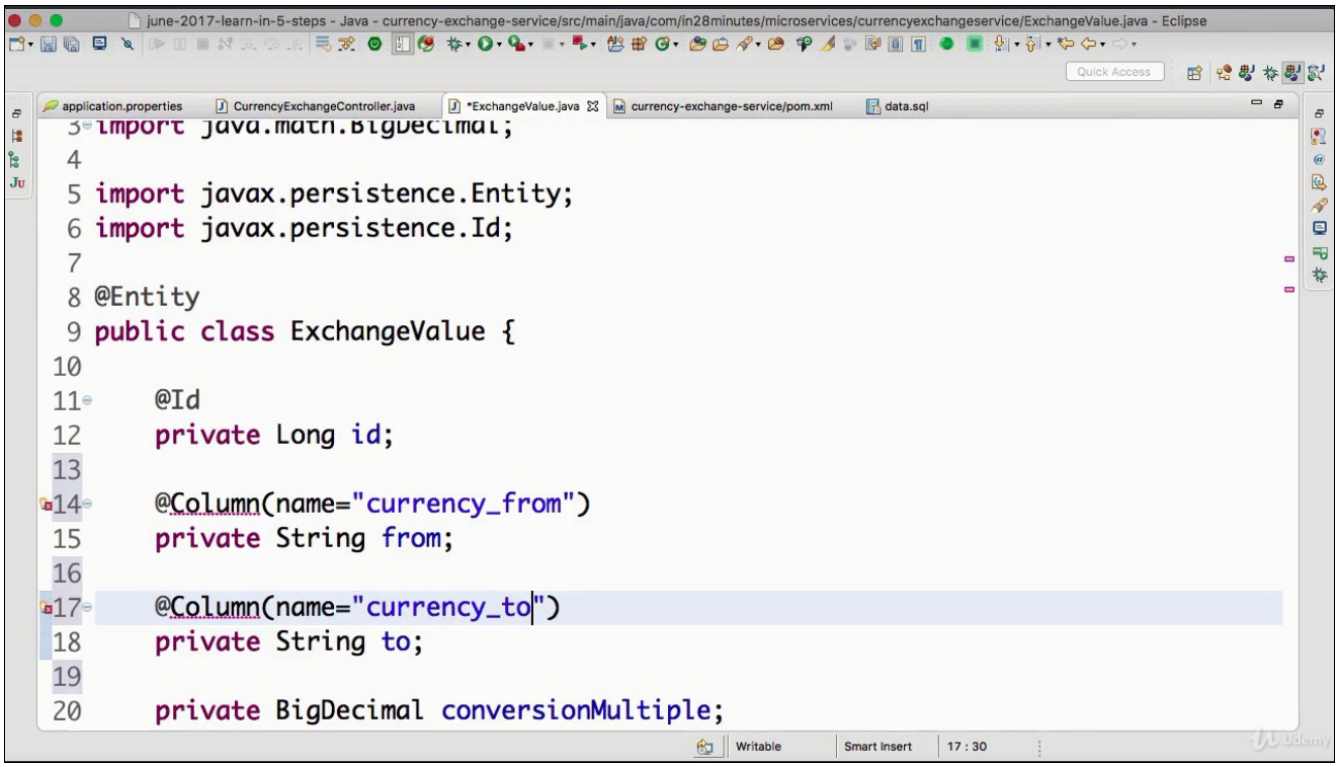


Lecture 72
Action 3: Configure JPA & Initialize data & add JPA repository

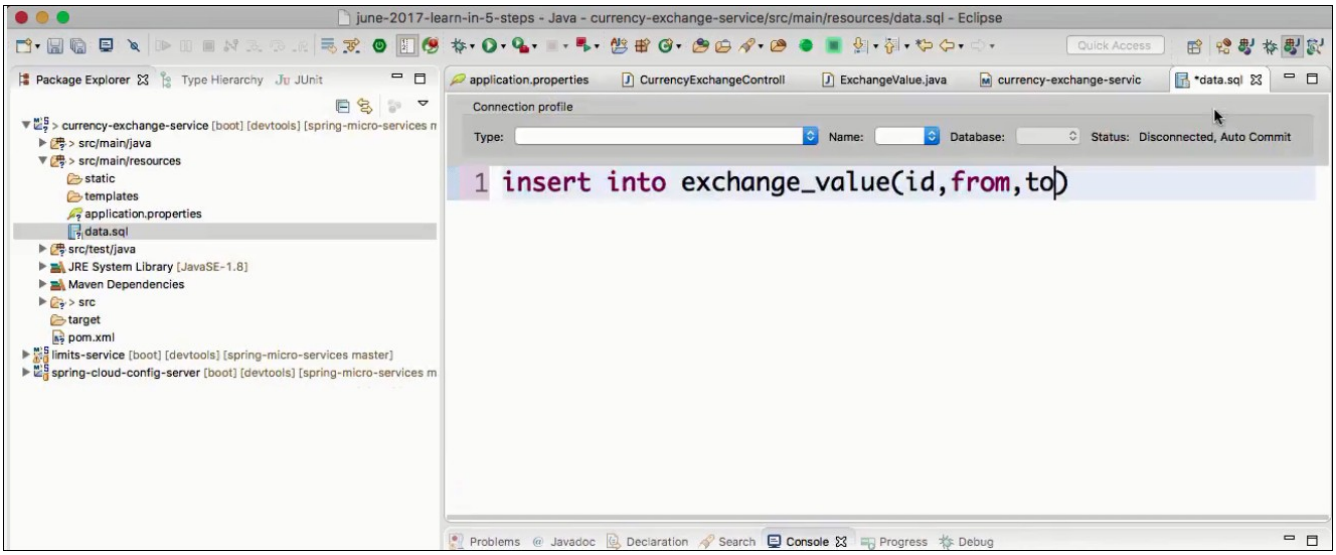
Step1: Add dependency in pom.xml

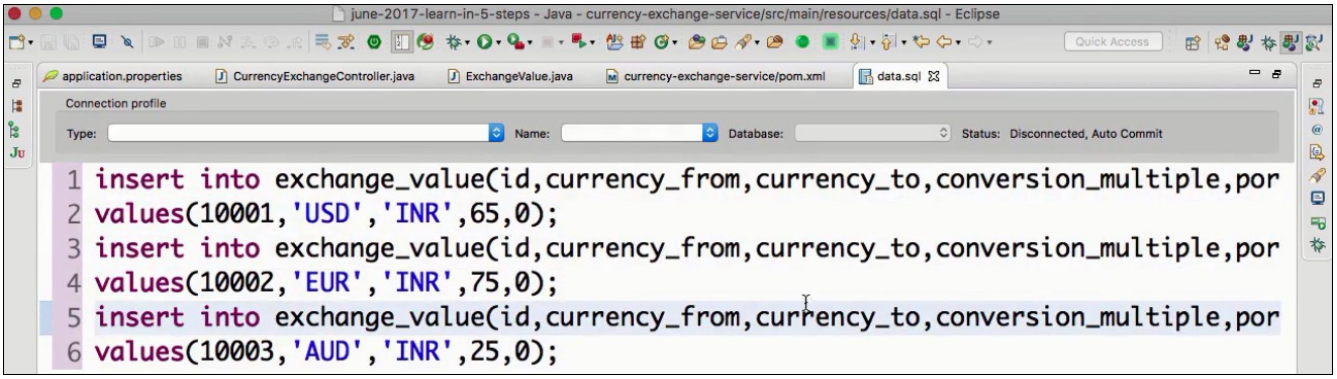


Step2: Add Entity

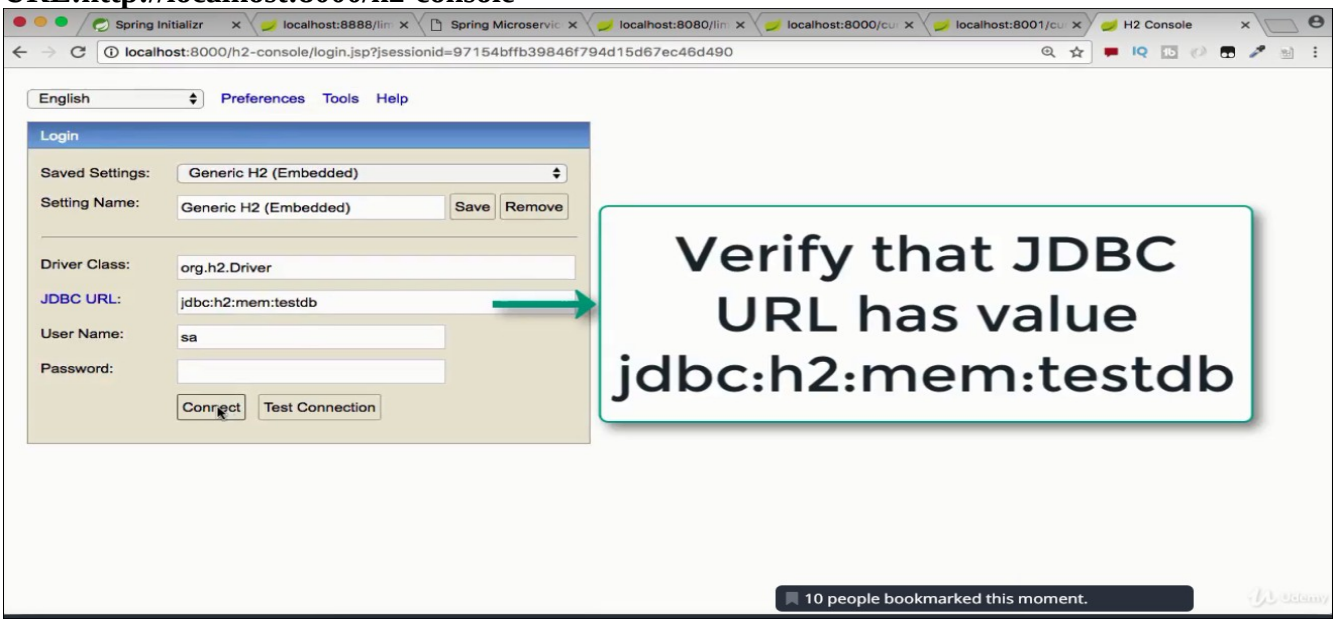


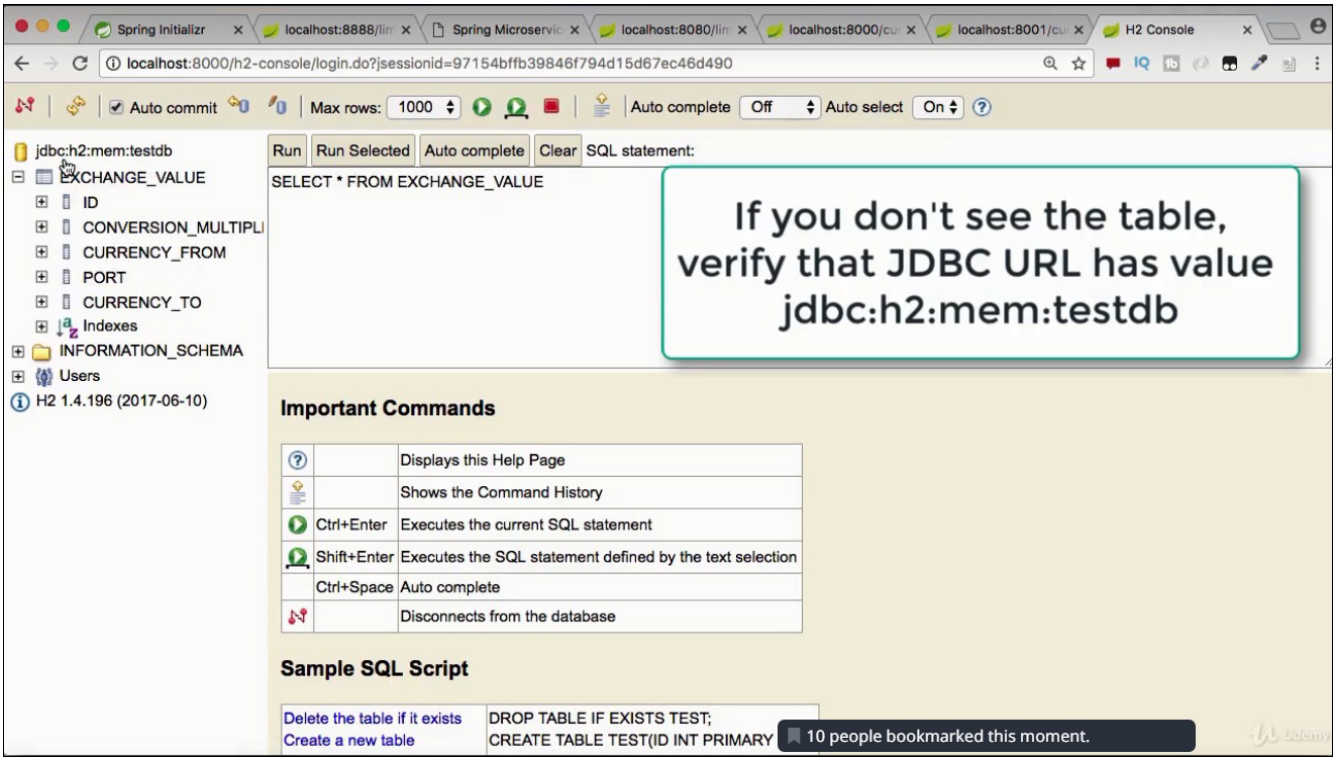
Step3: Add the file data.sql & add following sql to it:





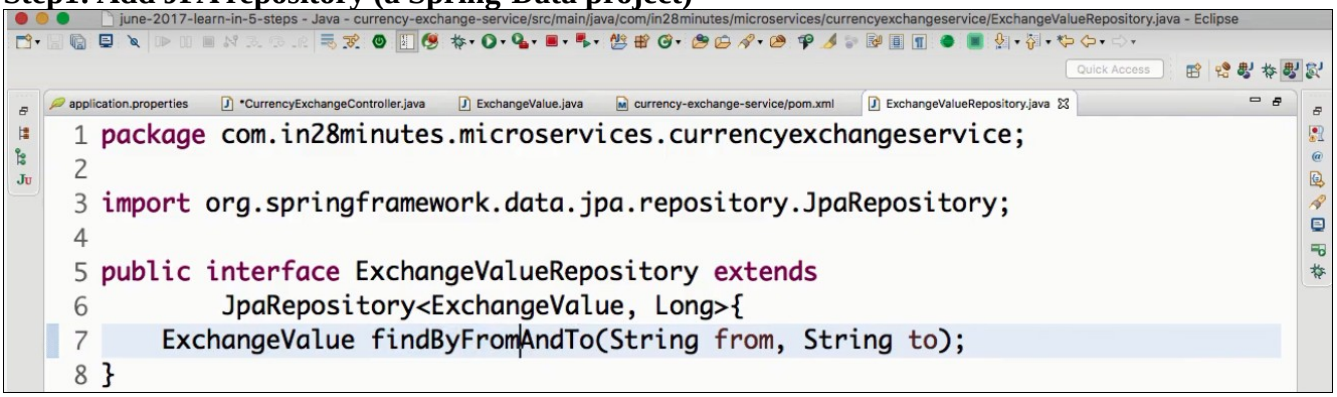
Step4: Restart the application
Step5:Verify that the table is created in the H2O console:
URL:http://localhost:8000/h2-console




Spring Initializr
localhost:8888/lim
Spring Microservic
localhost:8080/lim
localhost:8000/cu
localhost:8001/cu
H2 Console
localhost:8000/h2-console/login.do?jsessionid=97154bffb39846f794d15d67ec46d490
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:mem:testdb
EXCHANGE_VALUE
ID
CONVERSION_MULTIPLI
CURRENCY_FROM
PORT
CURRENCY_TO
Indexes
INFORMATION_SCHEMA
Users
H2 1.4.196 (2017-06-10)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM EXCHANGE_VALUE
If you don't see the table, verify that JDBC URL has value jdbc:h2:mem:testdb
Important Commands
Displays this Help Page
Shows the Command History
Ctrl+Enter
Executes the current SQL statement
Shift+Enter
Executes the SQL statement defined by the text selection
Ctrl+Space
Auto complete
Disconnects from the database
Sample SQL Script
Delete the table if it exists
Create a new table
DROP TABLE IF EXISTS TEST;
CREATE TABLE TEST(ID INT PRIMARY
10 people bookmarked this moment.

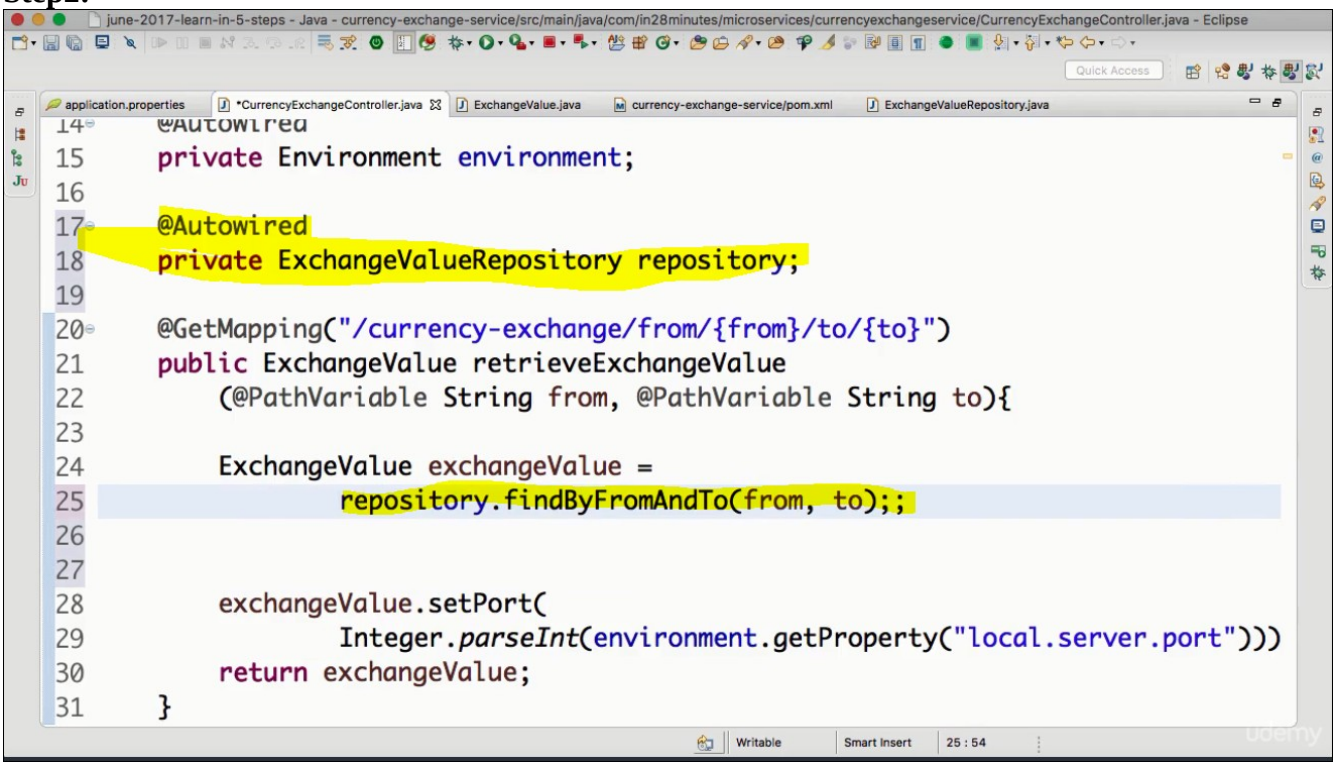
Lecture 73: Add JPA to currency Exchange service(I.e Create a JPA repository)

Step1: Add JPA repository (a Spring-Data project)

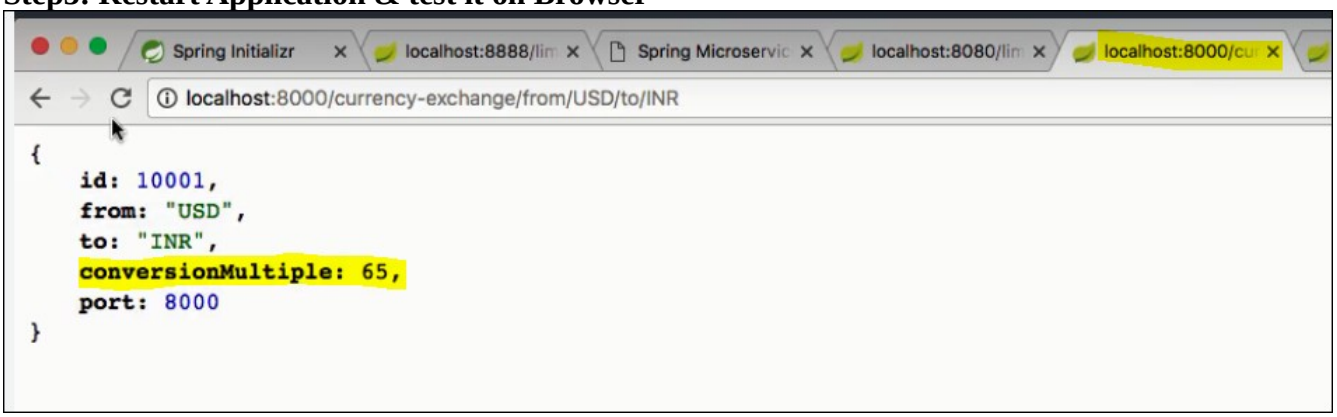


** findByFromAndTo → JPA in Spring -Data will implement the method to find By From & To.

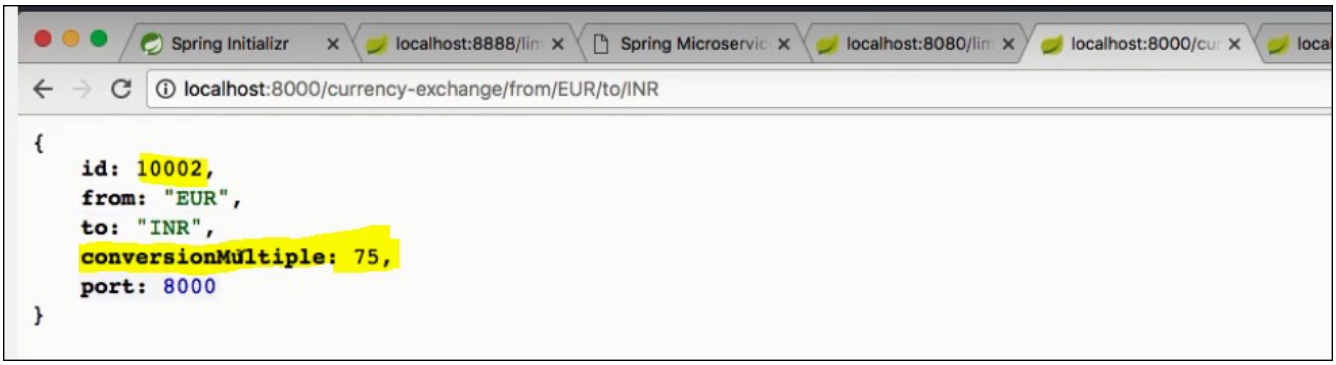
Step2:



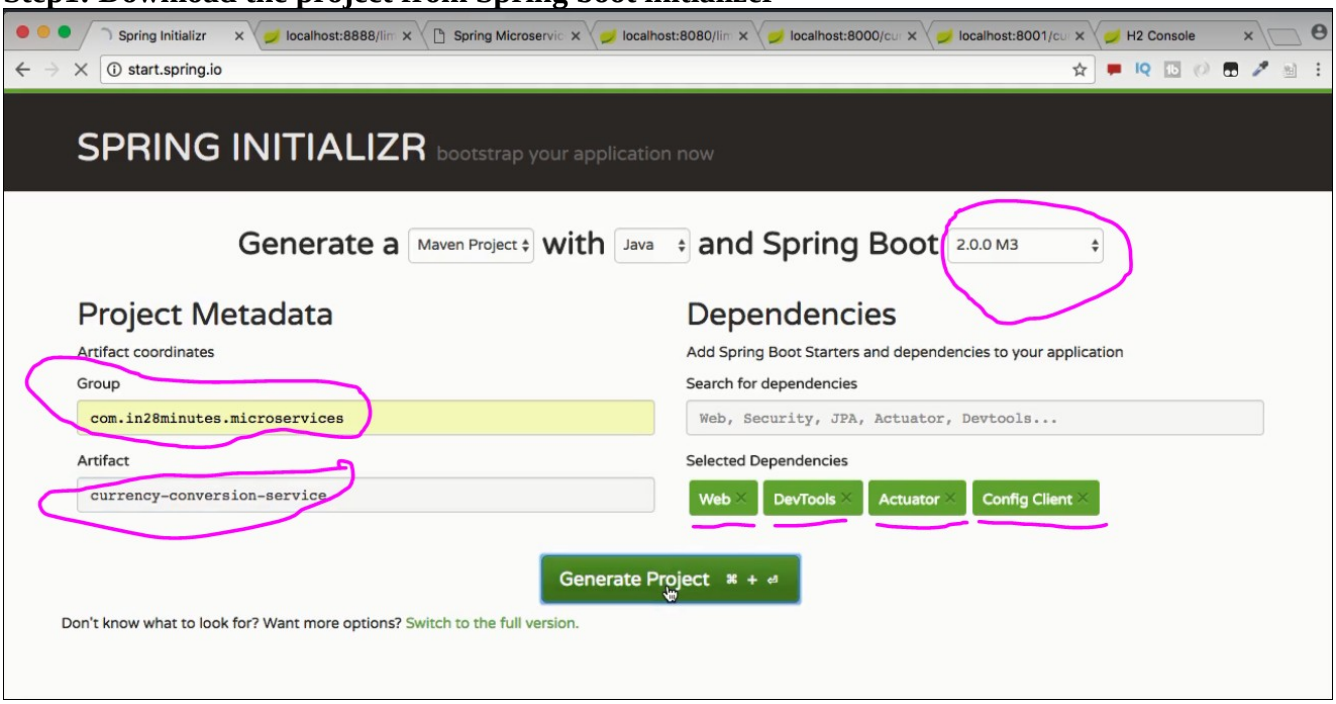
Step3: Restart Application & test it on Browser



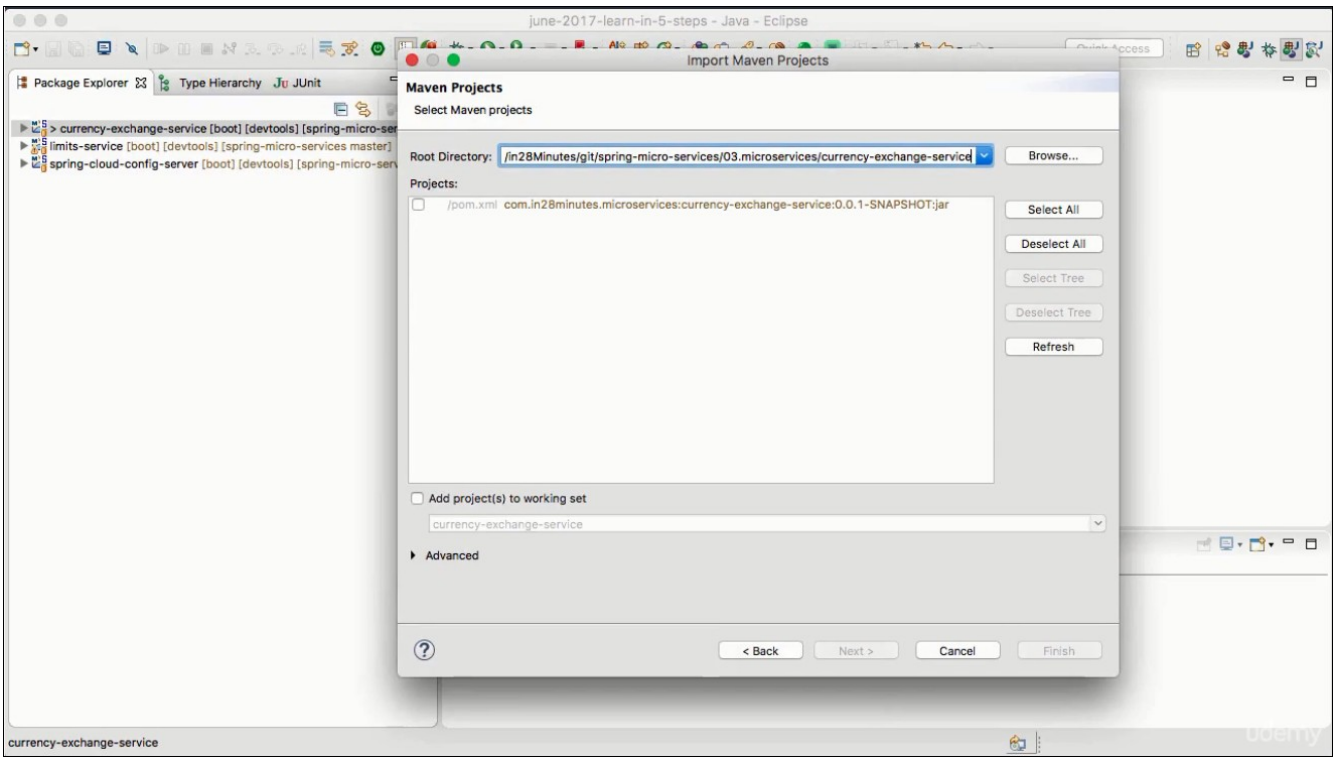
Test it with another From & To:



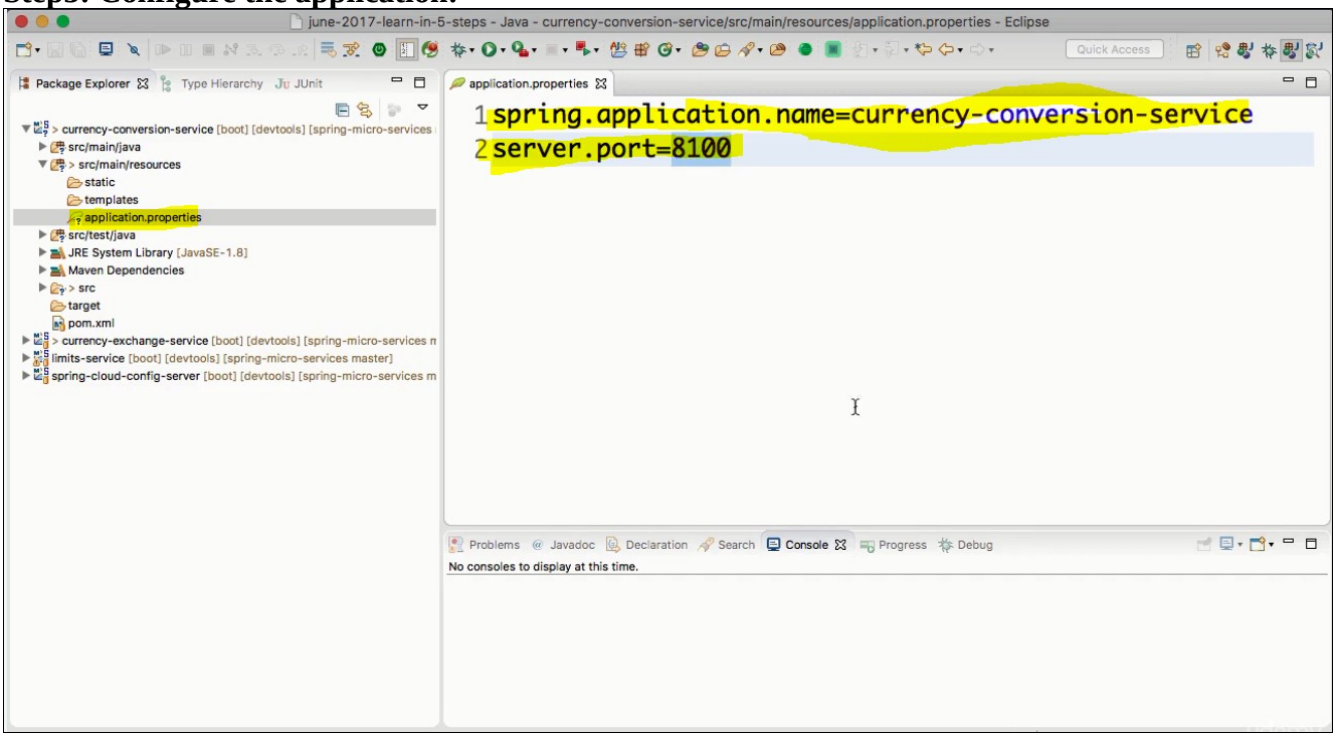
Lecture 74: Setup Currency Conversion Microservice
Step1: Download the project from Spring boot initializer



Step2: Import the downloaded project
File → Import → Existing Maven Project → (below)

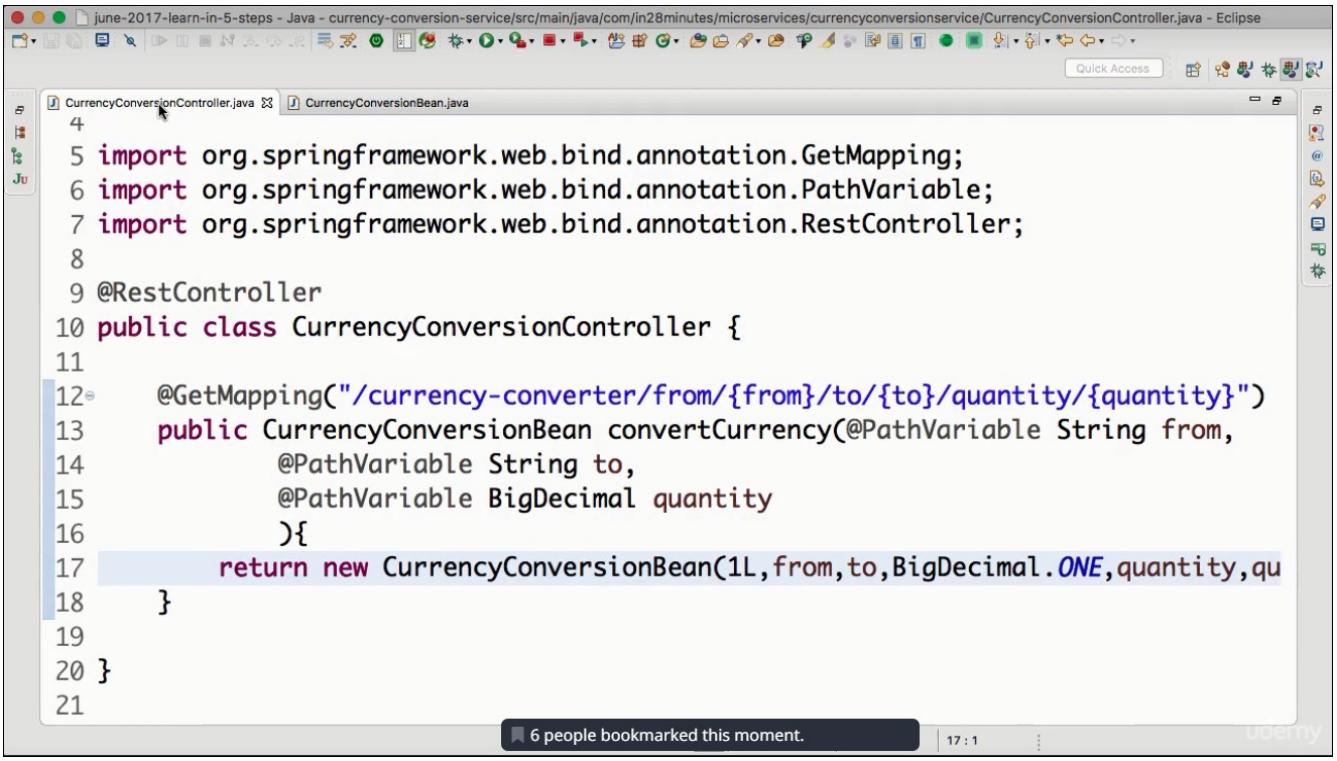


Step3: Configure the application:

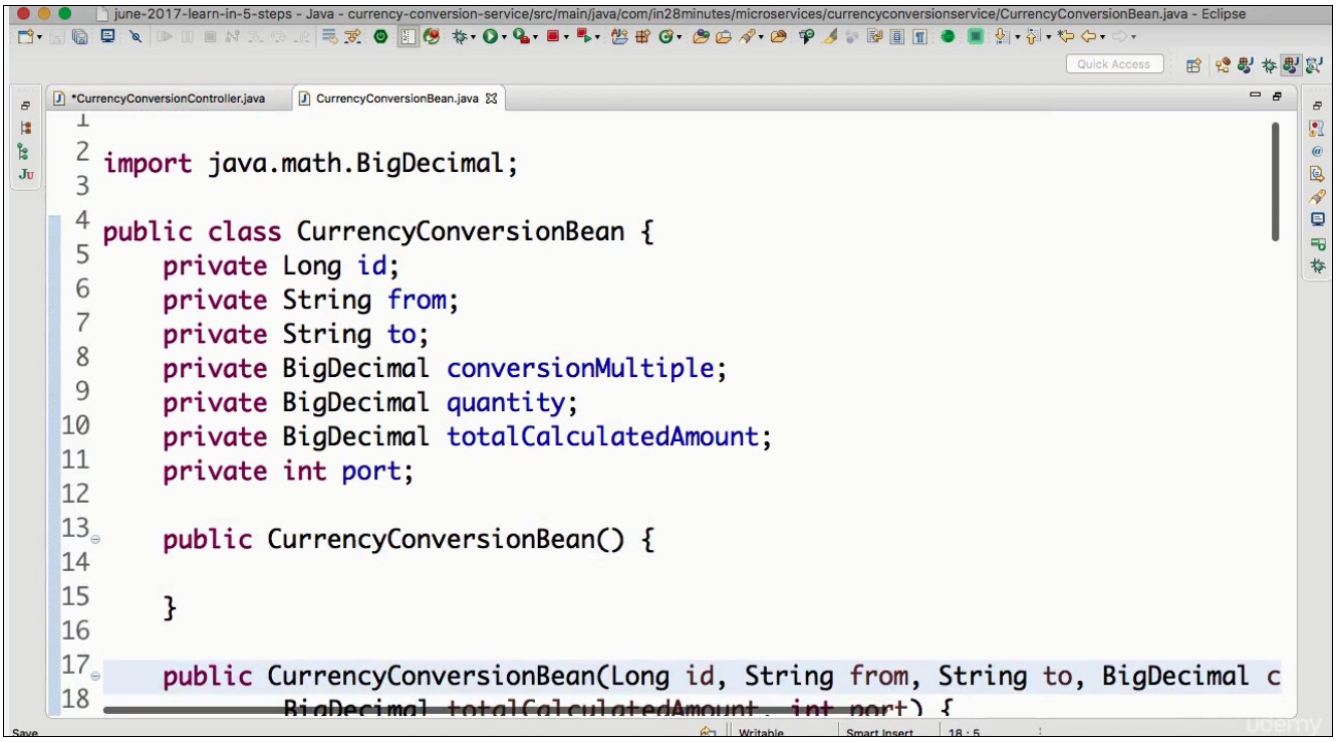


Step4: Add Controller & Bean

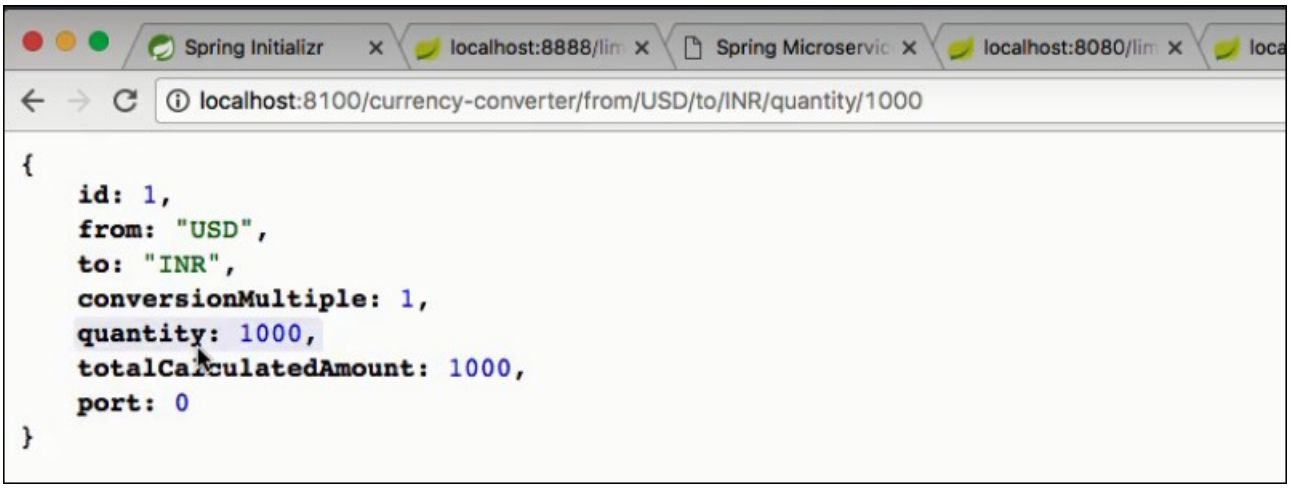
As of now return a hard coded bean to test.


june-2017-learn-in-5-steps - Java - currency-conversion-service/src/main/java/com/in28minutes/microservices/currencyconversionservice/CurrencyConversionController.java - Eclipse
CurrencyConversionController.java
CurrencyConversionBean.java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class CurrencyConversionController {
@GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")
public CurrencyConversionBean convertCurrency(@PathVariable String from,
@PathVariable String to,
@PathVariable BigDecimal quantity
){
return new CurrencyConversionBean(1L,from,to,BigDecimal.ONE,quantity,qu
}
}
6 people bookmarked this moment.
17 : 1

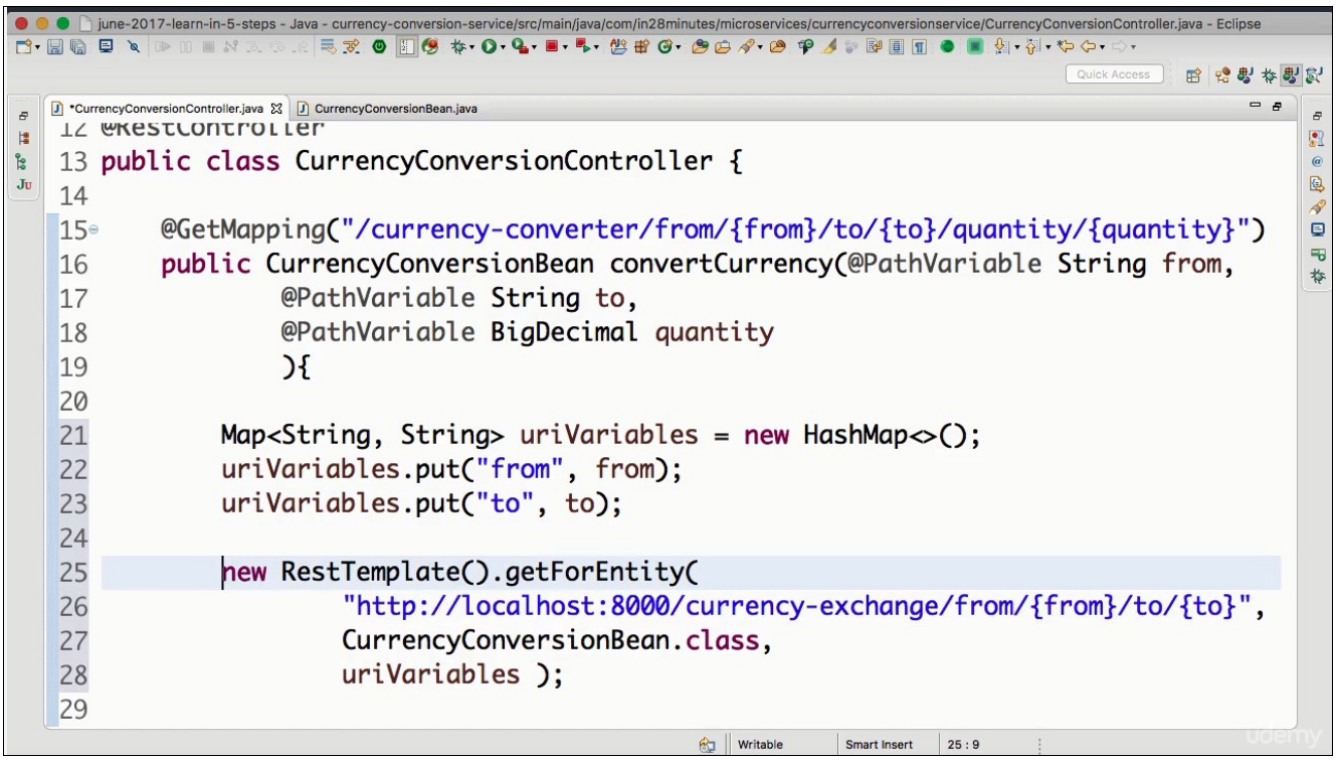
Bean:


june-2017-learn-in-5-steps - Java - currency-conversion-service/src/main/java/com/in28minutes/microservices/currencyconversionservice/CurrencyConversionBean.java - Eclipse
*CurrencyConversionController.java
CurrencyConversionBean.java
import java.math.BigDecimal;
public class CurrencyConversionBean {
private Long id;
private String from;
private String to;
private BigDecimal conversionMultiple;
private BigDecimal quantity;
private BigDecimal totalCalculatedAmount;
private int port;
public CurrencyConversionBean() {
}
public CurrencyConversionBean(Long id, String from, String to, BigDecimal c

Test:



Lecture 76: Invoke Currency Exchange Service from Currency Converter service



```
22          Map<String, String> uriVariables = new HashMap<>();
23          uriVariables.put("from", from);
24          uriVariables.put("to", to);
25
26          ResponseEntity<CurrencyConversionBean> responseEntity = new RestTemplat
27                  "http://localhost:8000/currency-exchange/from/{from}/to/{to}",
28                  CurrencyConversionBean.class,
29                  uriVariables );
30
31          CurrencyConversionBean response = responseEntity.getBody();
32
33          return new CurrencyConversionBean(response.getId(),from,to,response.get
34                  quantity,quantity.multiply(response.getConversionMultiple()),re
35      }
```

Check the output:

****

Use Spring Cloud Version - Finchley.M8

Section 4, Lecture 77

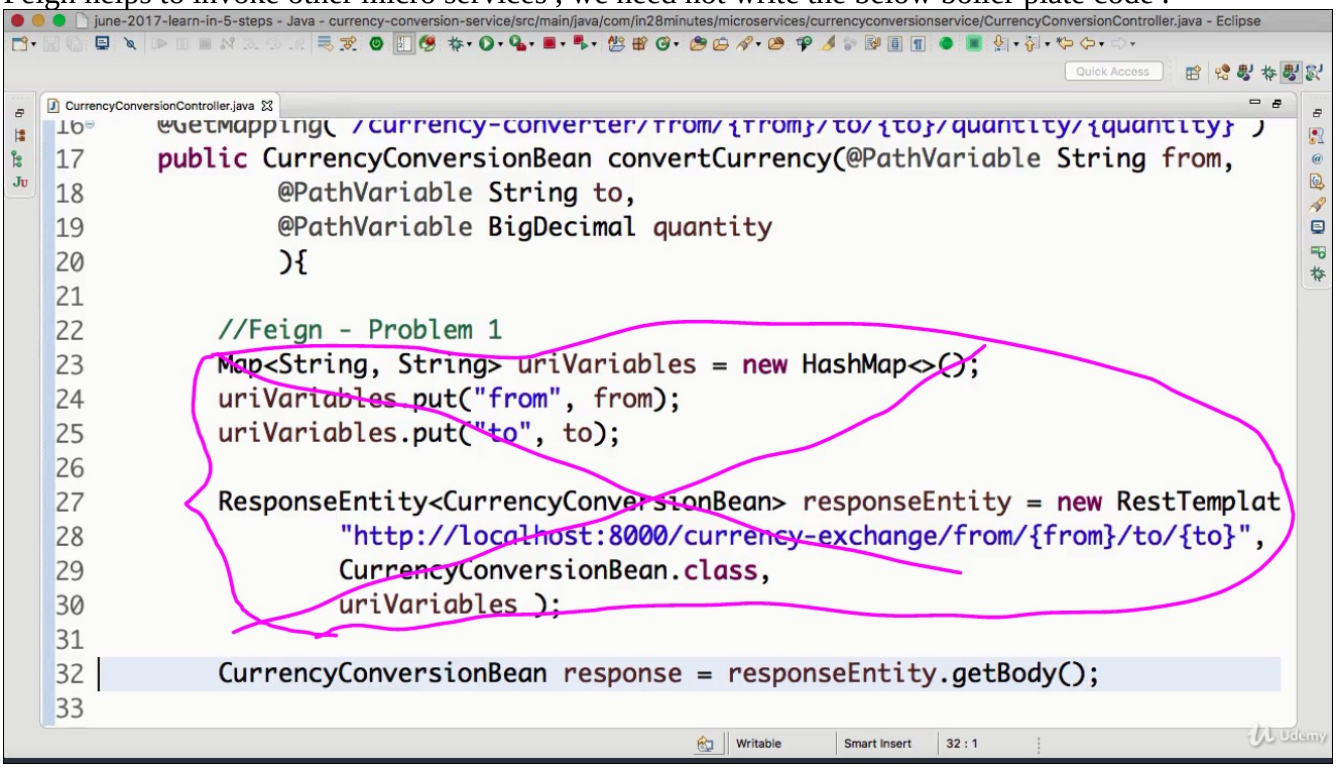
Problem with M9 Dependencies -> https://github.com/spring-cloud/spring-cloud-openfeign/issues/13

We recommend using **Finchley.M8** for now

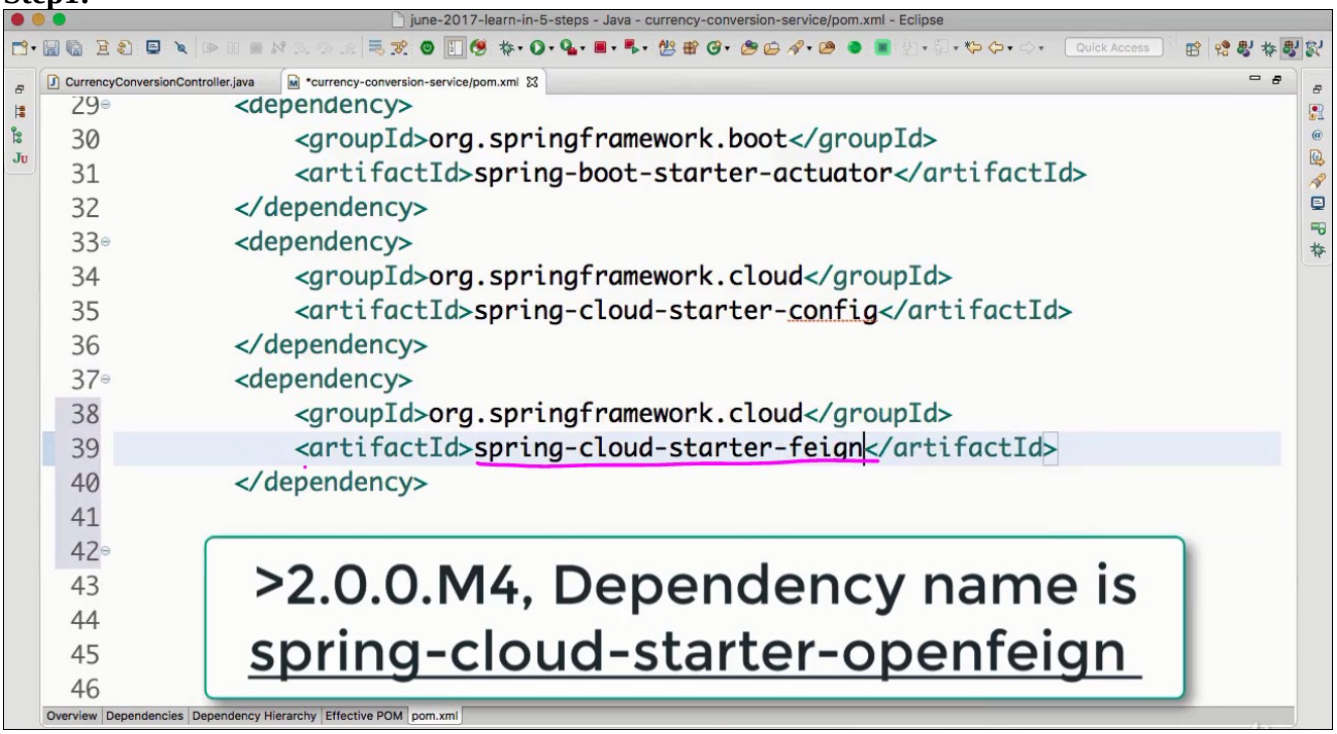
In pom.xml, use

```
<spring-cloud.version>Finchley.M8</spring-cloud.version>
```

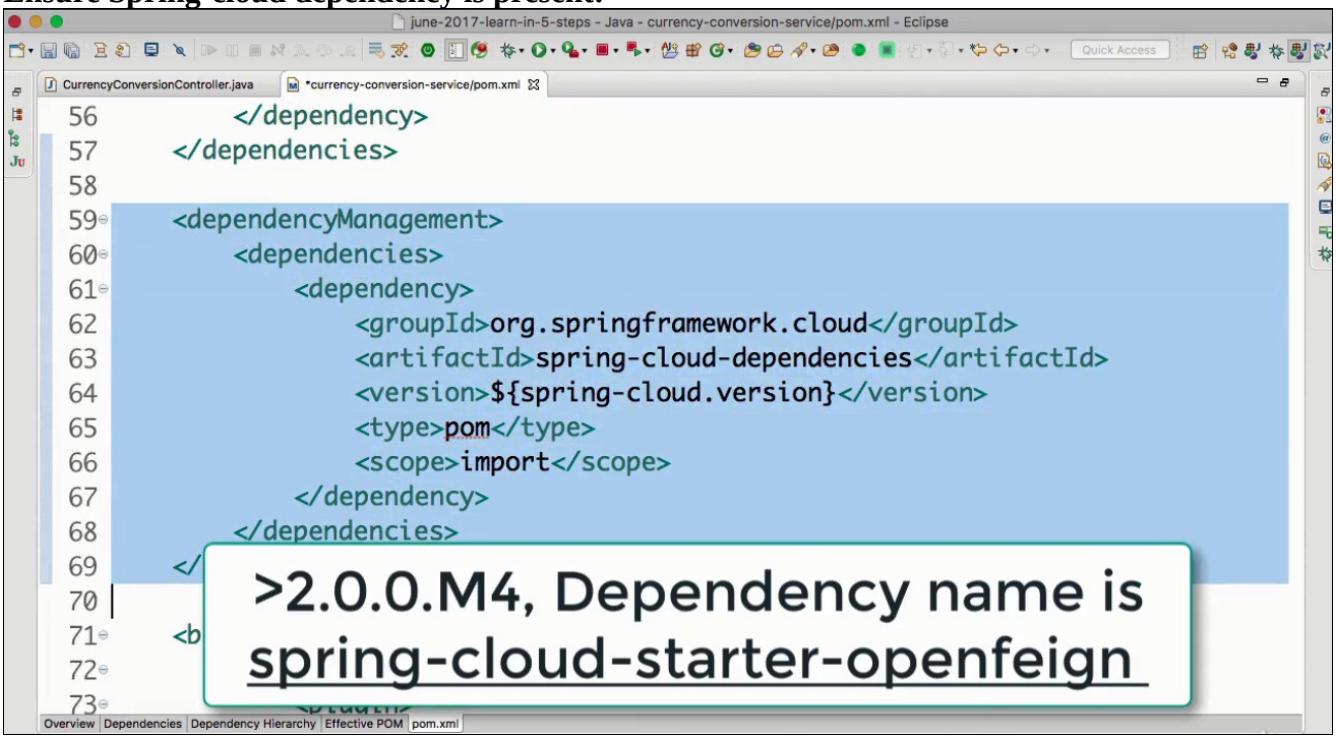
Lecture 77 : Disclaimer to use Finchley.M8
Lecture 78: Using Feign REST Client.
Feign helps to invoke other micro services , we need not write the below boiler plate code .



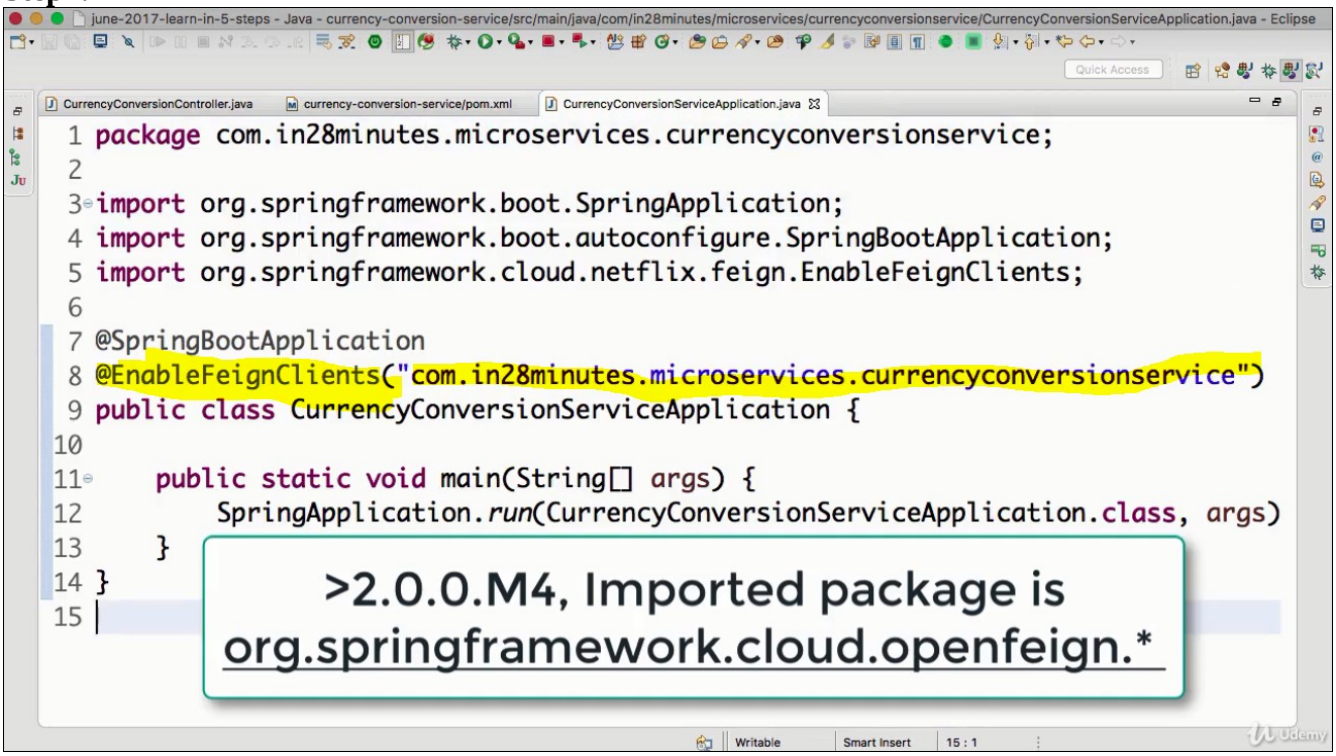
Action5: Add Feign support
Step1:



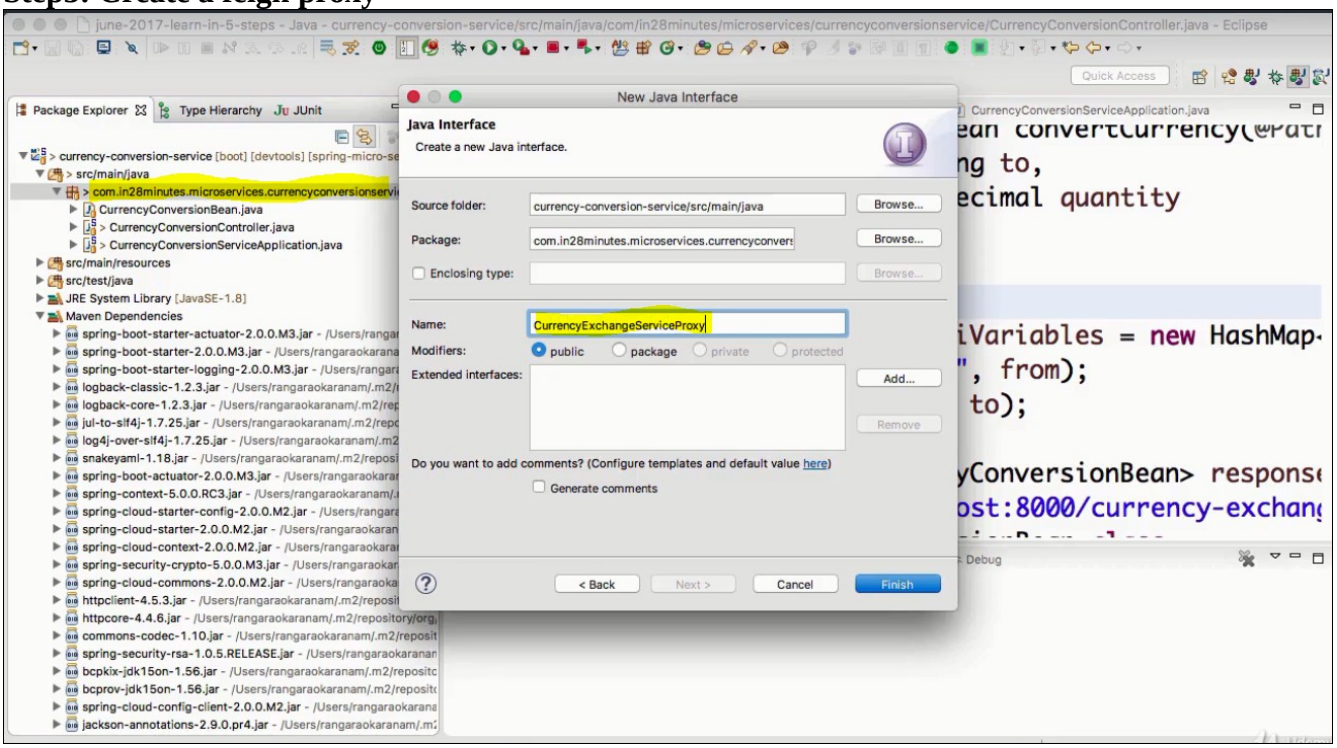
Ensure Spring-cloud dependency is present:

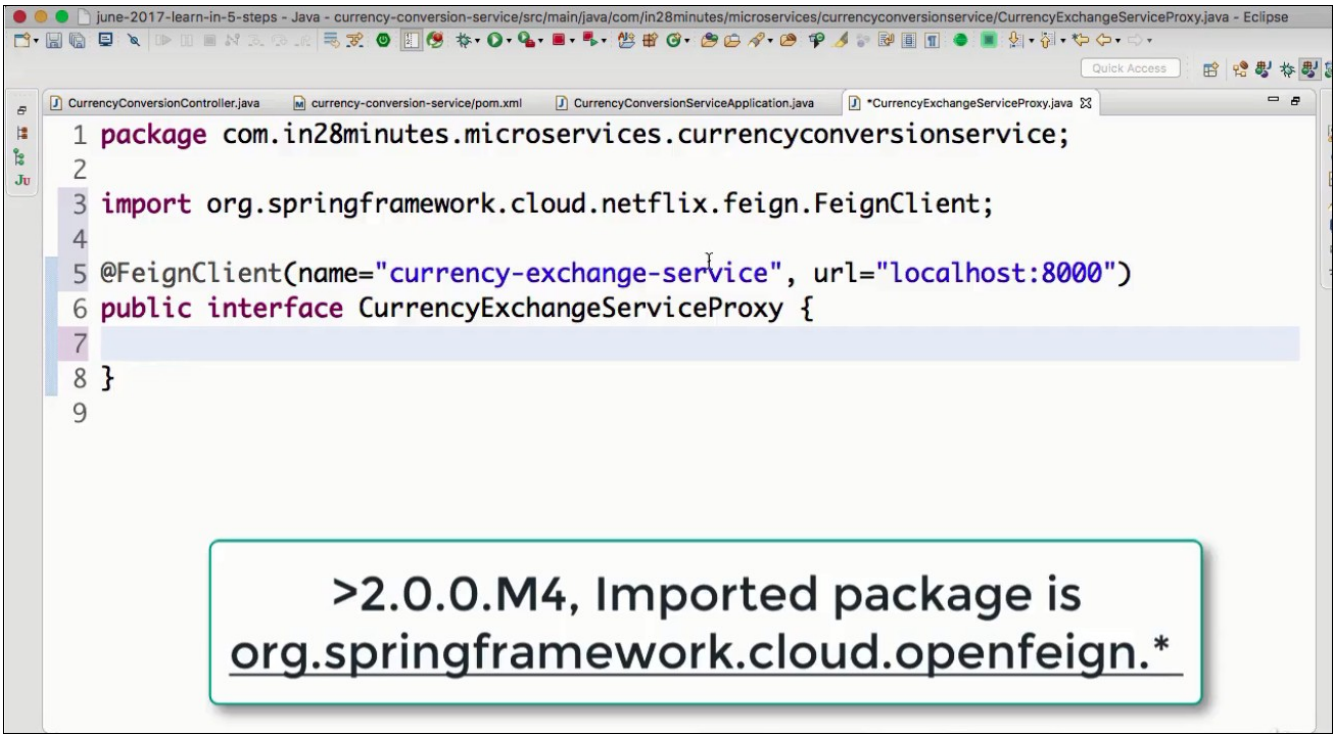


Step2:

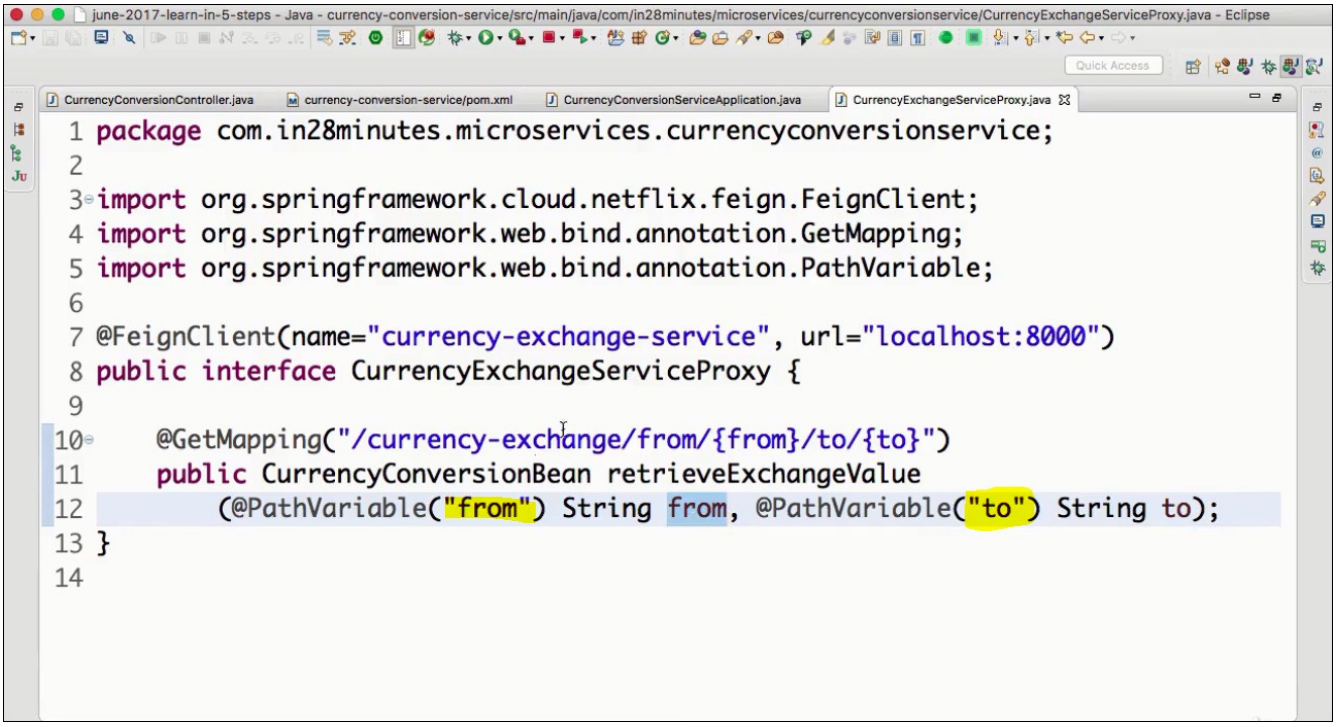


Step3: Create a feign proxy

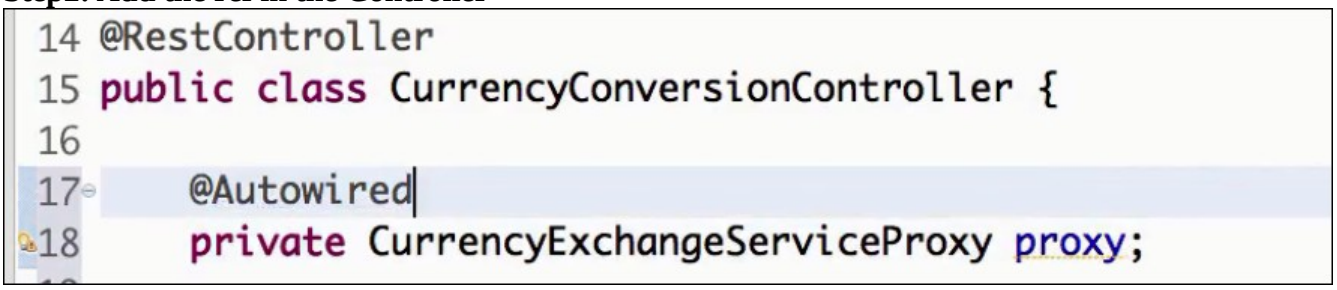




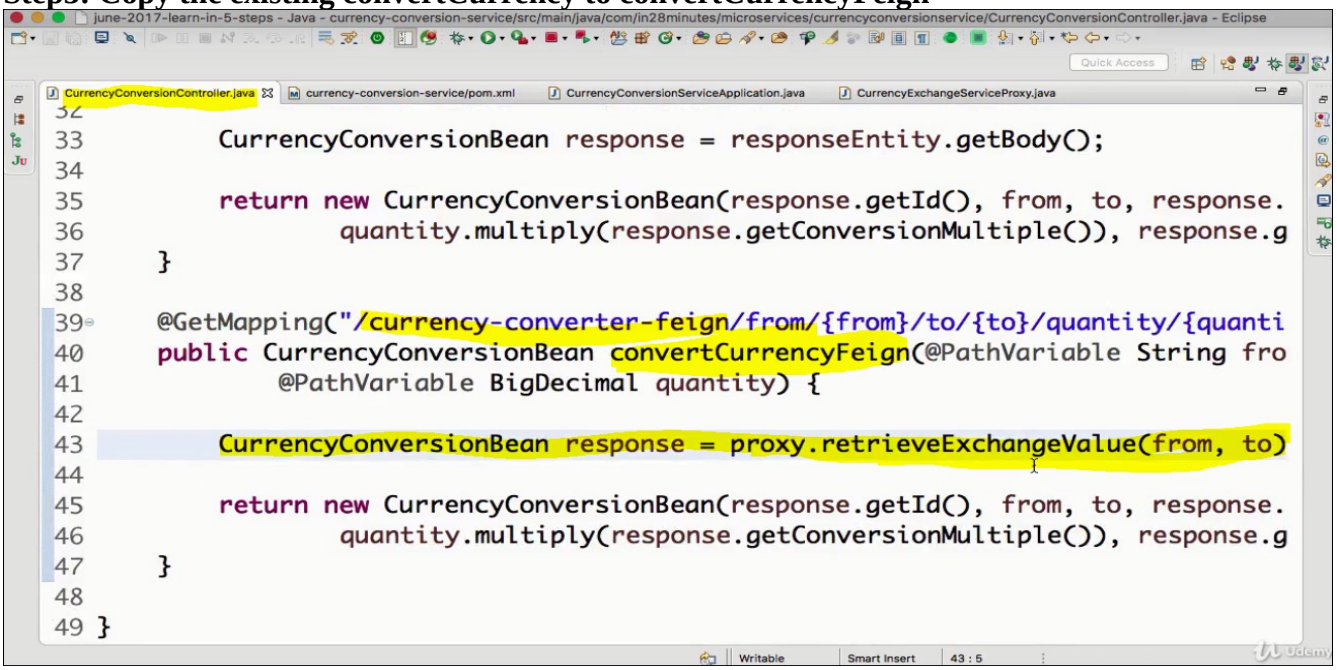
Define the implementation services: We copy the method that would get called from the CurrencyExchangeServiceController & twik a bit.



Step2: Add the ref in the Controller



Step3: Copy the existing convertCurrency to convertCurrencyFeign



Step4: Restart CurrencyConverterService and test: