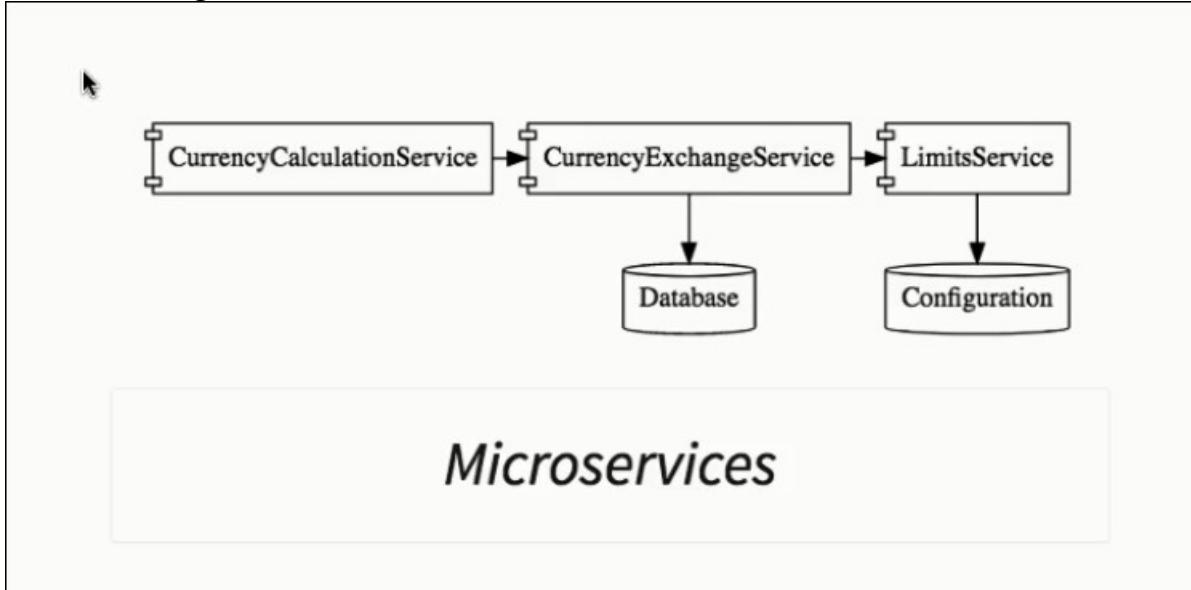


Lecture 68: To design the below distributed services



Lecture 69,70,71: Setup currency-exchange-service

Action1: Create currency-exchange-service

Step1: Download the configured springboot 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.

Generate a with and Spring Boot

Project Metadata

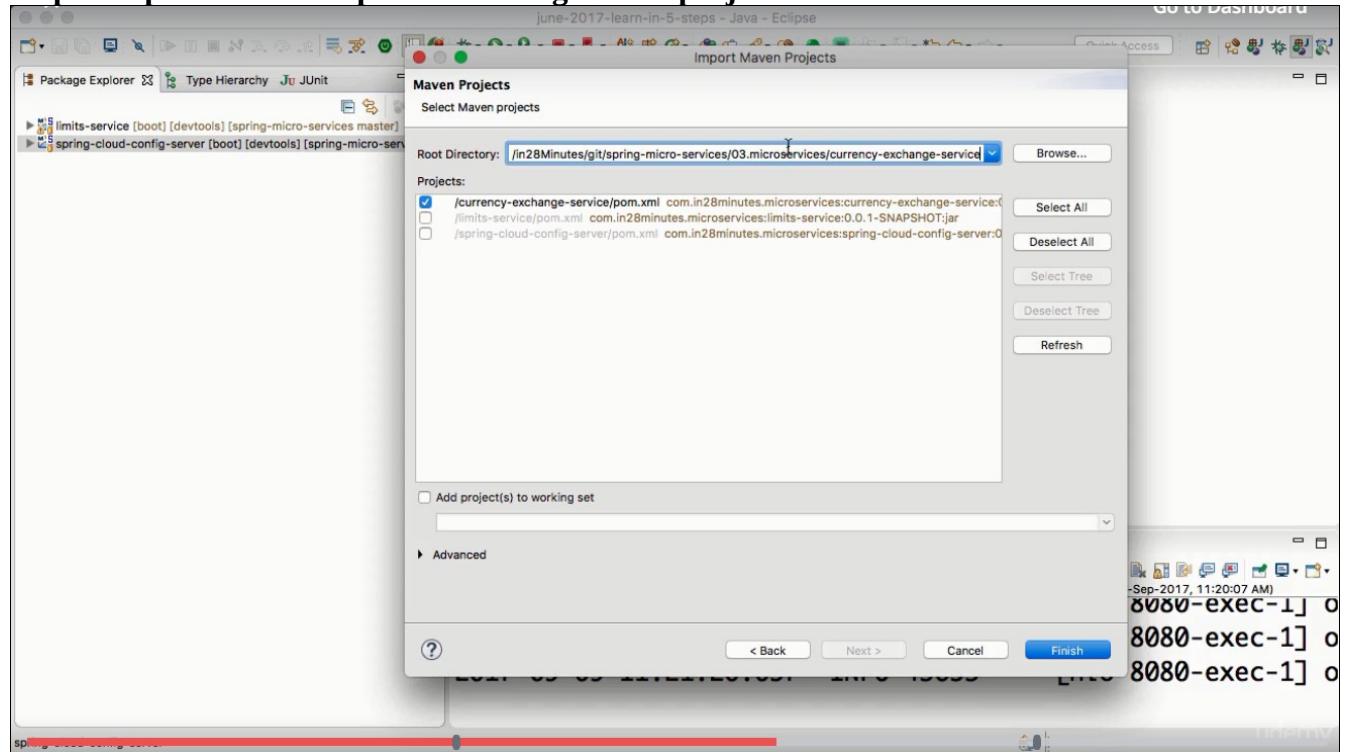
Artifact coordinates
Group
Artifact

Dependencies

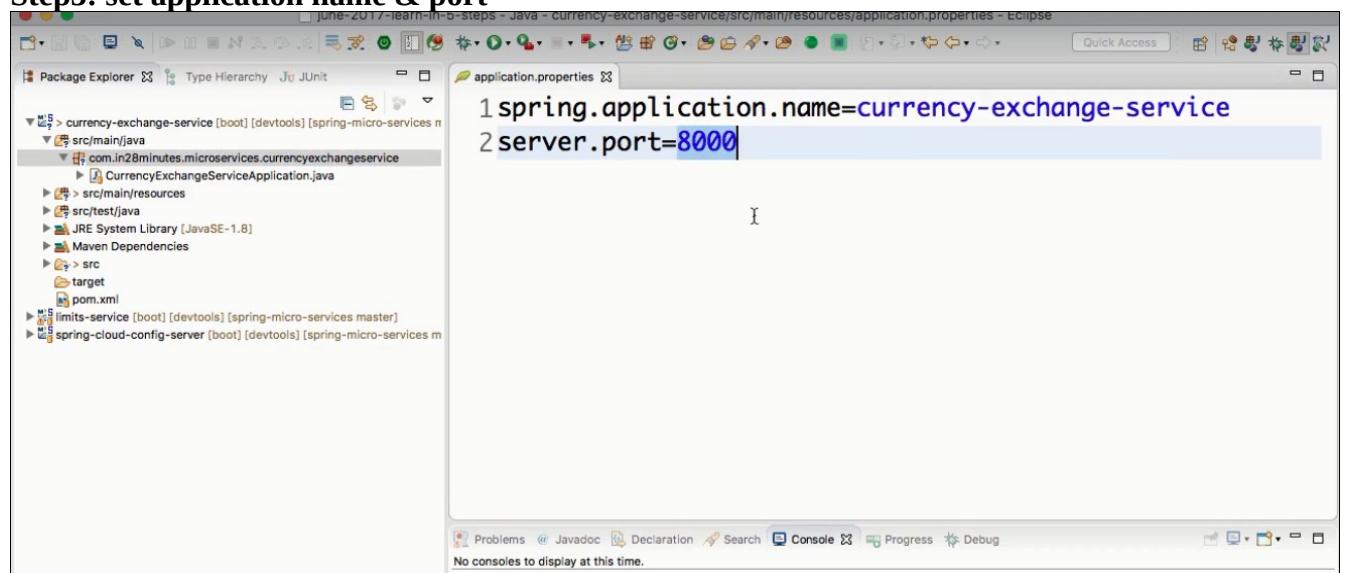
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Selected Dependencies

Don't know what to look for? Want more options? [Switch to the full version.](#)

Step2: Import it to workspace as existing Maven project

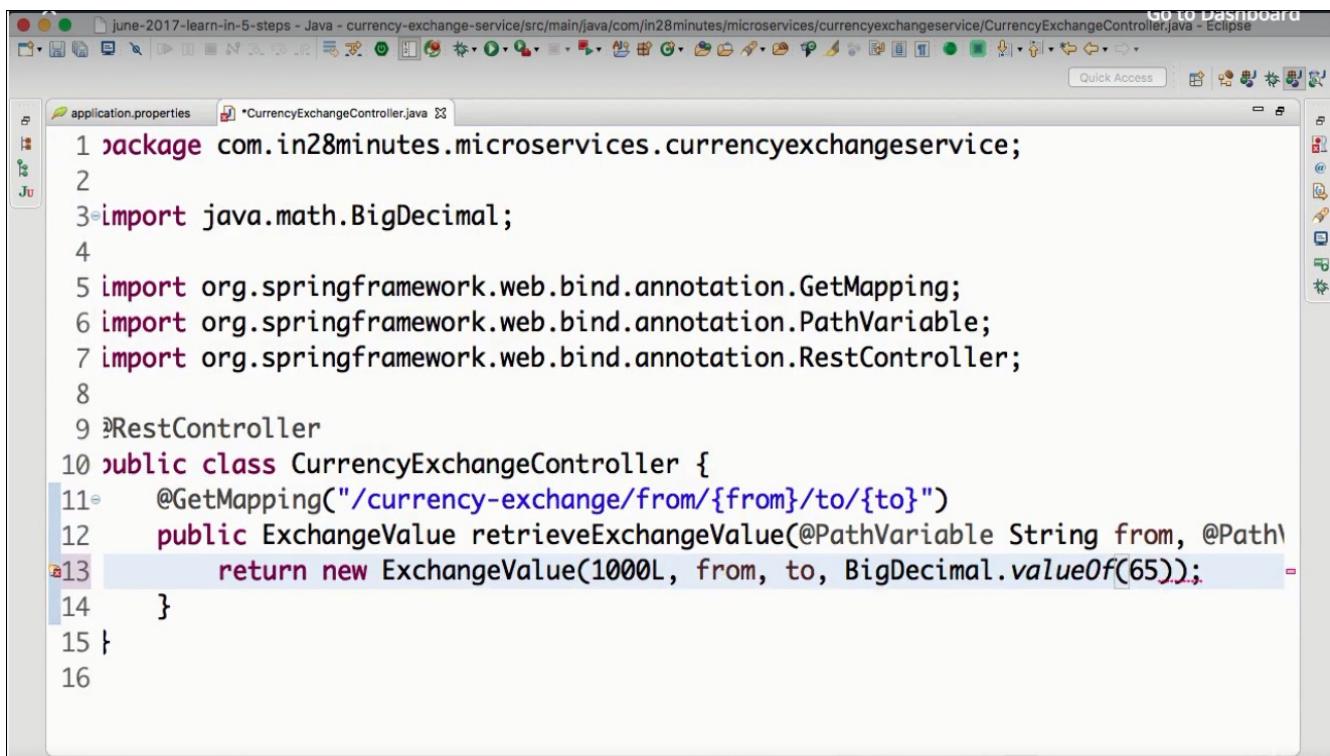


Step3: set application name & port



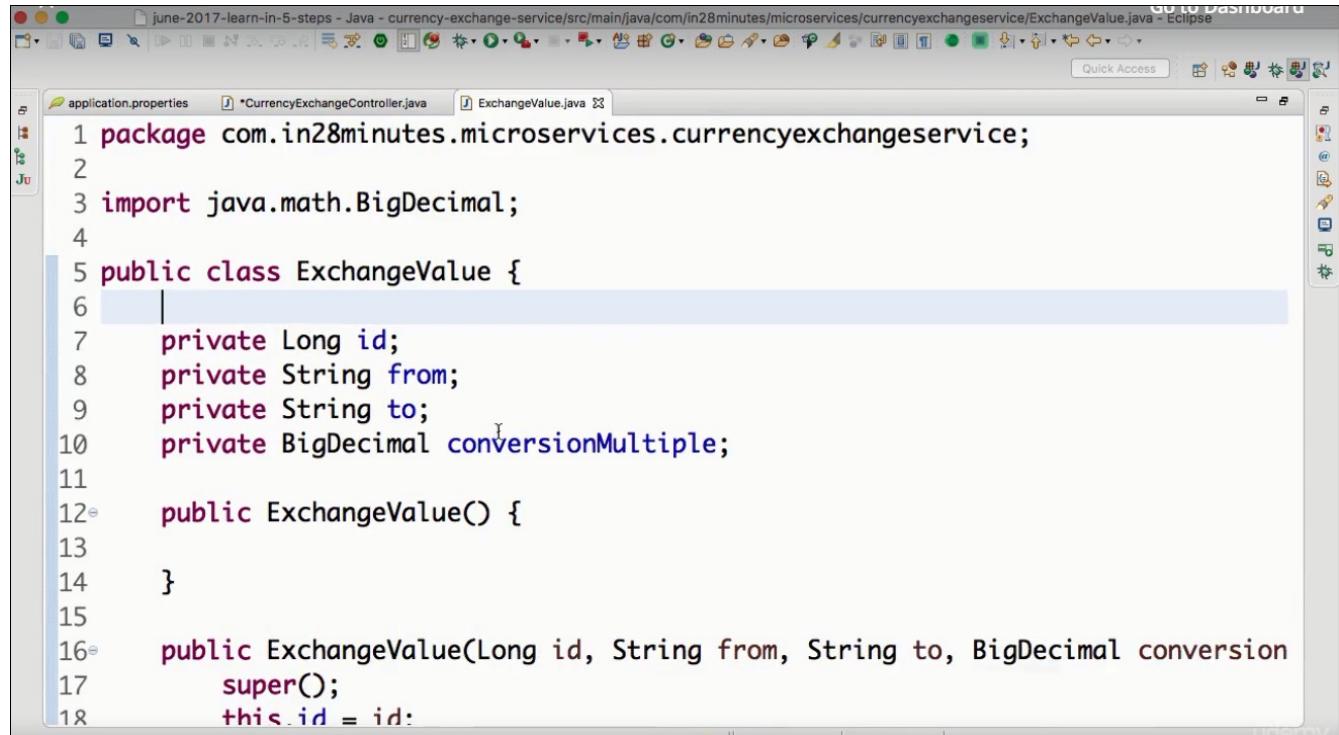
Step4: Add currency controller functionality

Define controller:



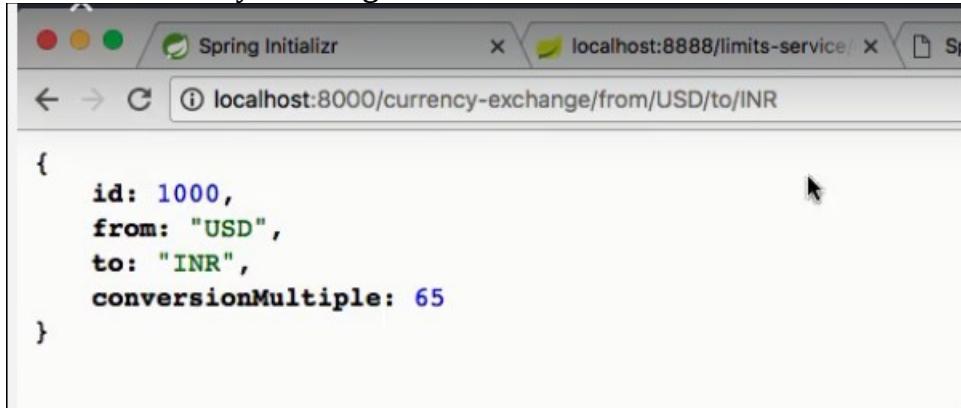
```
1 package com.in28minutes.microservices.currencyexchangeservice;
2
3 import java.math.BigDecimal;
4
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PathVariable;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @RestController
10 public class CurrencyExchangeController {
11     @GetMapping("/currency-exchange/from/{from}/to/{to}")
12     public ExchangeValue retrieveExchangeValue(@PathVariable String from, @PathVariable String to) {
13         return new ExchangeValue(1000L, from, to, BigDecimal.valueOf(65));
14     }
15 }
16
```

Define Bean:



```
1 package com.in28minutes.microservices.currencyexchangeservice;
2
3 import java.math.BigDecimal;
4
5 public class ExchangeValue {
6     private Long id;
7     private String from;
8     private String to;
9     private BigDecimal conversionMultiple;
10
11     public ExchangeValue() {
12     }
13
14     public ExchangeValue(Long id, String from, String to, BigDecimal conversionMultiple) {
15         super();
16         this.id = id;
17     }
18 }
```

Test the Currency Exchange service:

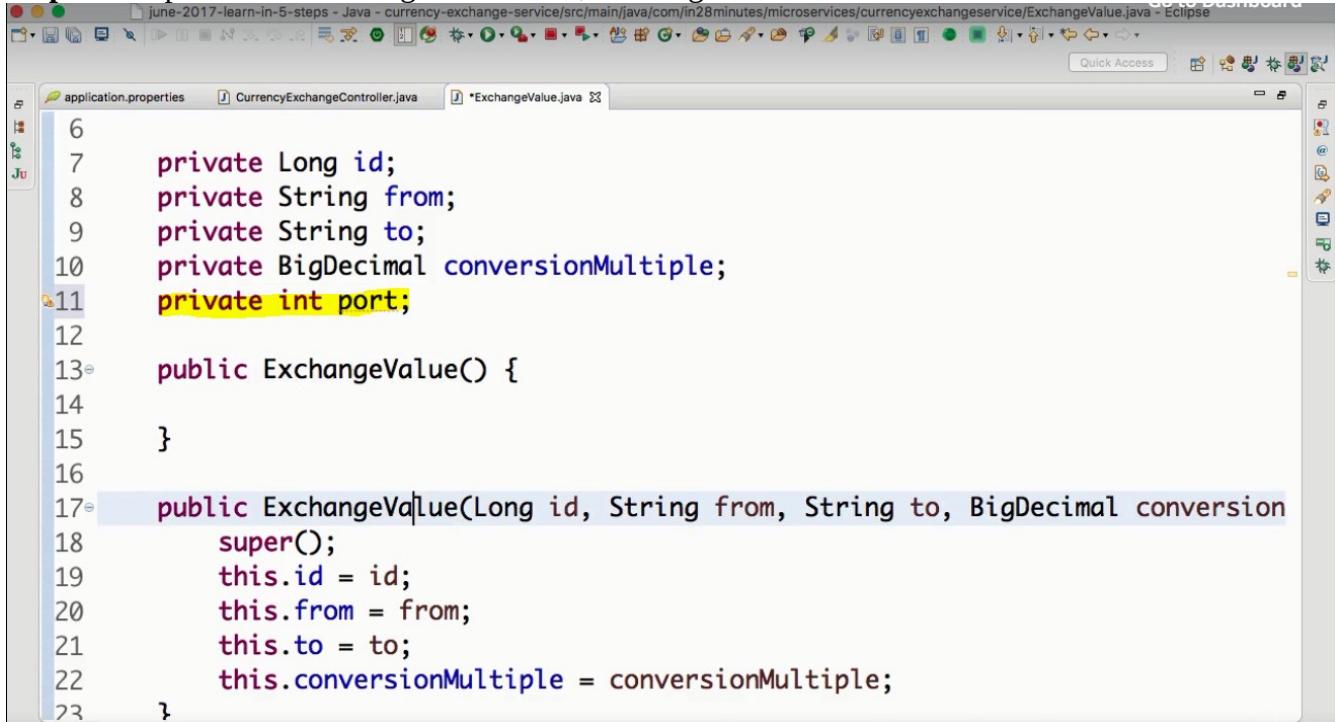


A screenshot of a web browser window. The title bar says "Spring Initializr" and the address bar shows "localhost:8000/currency-exchange/from/USD/to/INR". The main content area displays a JSON object:

```
{  
  id: 1000,  
  from: "USD",  
  to: "INR",  
  conversionMultiple: 65  
}
```

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

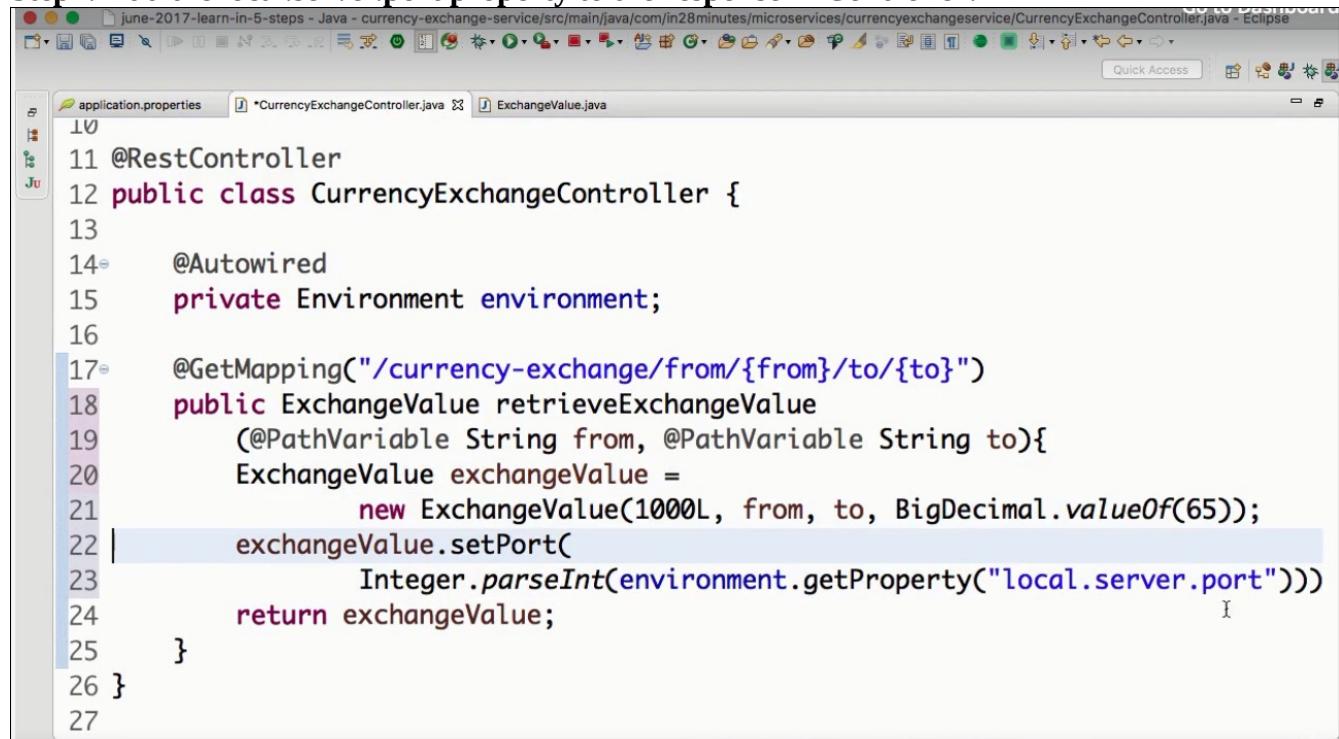
Step1: Add port to the ExchangeValue bean , add its getter setter but dont add in constructor.



A screenshot of the Eclipse IDE interface. The central editor shows the `ExchangeValue.java` file. The code defines a class with fields for id, from, to, conversionMultiple, and port, along with a constructor and a parameterized constructor that calls the super constructor and initializes the fields.

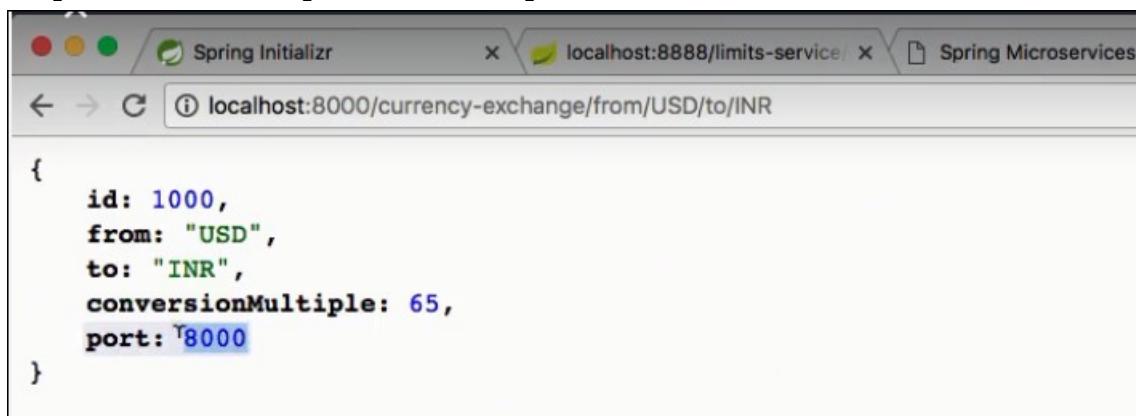
```
6  
7     private Long id;  
8     private String from;  
9     private String to;  
10    private BigDecimal conversionMultiple;  
11    private int port;  
12  
13    public ExchangeValue() {  
14    }  
15  
16    public ExchangeValue(Long id, String from, String to, BigDecimal conversion  
17        super();  
18        this.id = id;  
19        this.from = from;  
20        this.to = to;  
21        this.conversionMultiple = conversionMultiple;  
22    }  
23 }
```

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



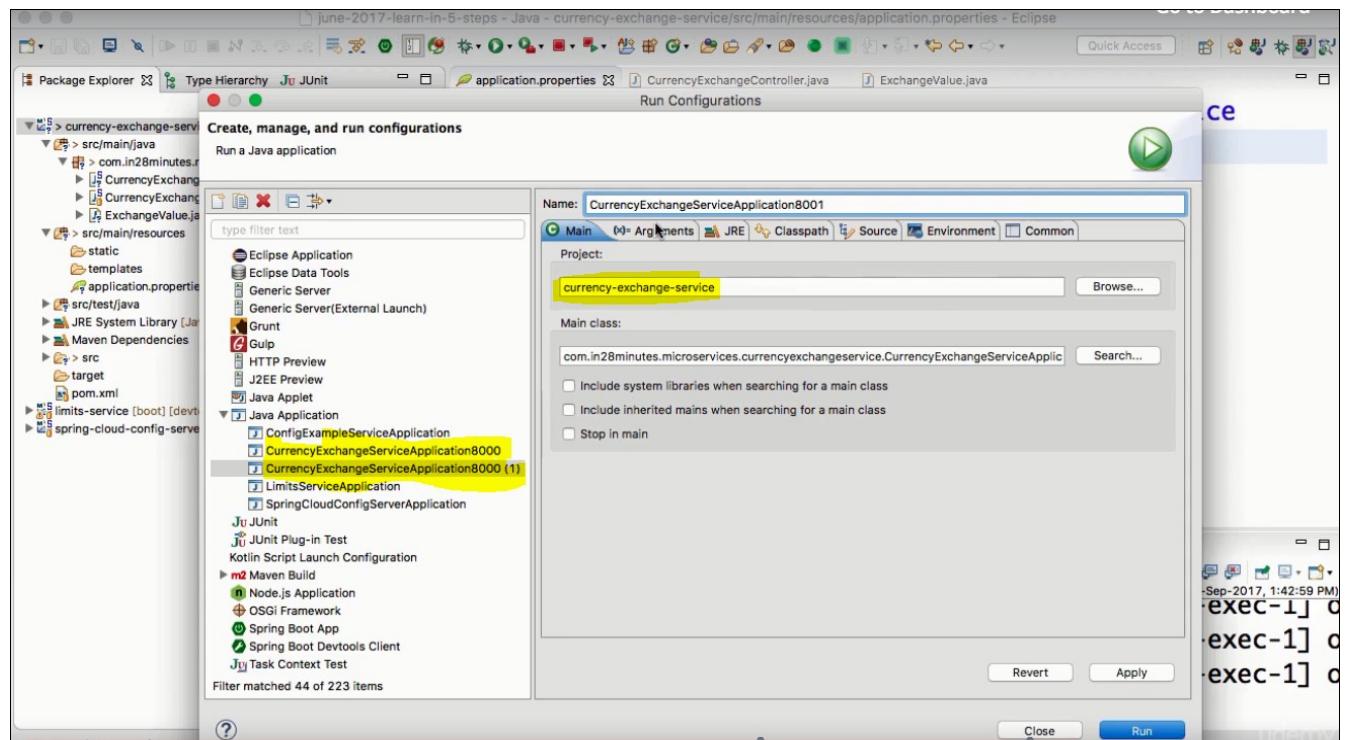
```
11 @RestController
12 public class CurrencyExchangeController {
13
14     @Autowired
15     private Environment environment;
16
17     @GetMapping("/currency-exchange/from/{from}/to/{to}")
18     public ExchangeValue retrieveExchangeValue
19         (@PathVariable String from, @PathVariable String to){
20         ExchangeValue exchangeValue =
21             new ExchangeValue(1000L, from, to, BigDecimal.valueOf(65));
22         exchangeValue.setPort(
23             Integer.parseInt(environment.getProperty("local.server.port")));
24         return exchangeValue;
25     }
26 }
27
```

Step3: Test to see the port value in response



```
{
  id: 1000,
  from: "USD",
  to: "INR",
  conversionMultiple: 65,
  port: 8000
}
```

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:

The image shows two side-by-side browser windows. Both have the title 'Spring Initializr' and show a JSON response at the URL 'localhost:8001/currency-exchange/from/USD/to/INR' and 'localhost:8000/currency-exchange/from/USD/to/INR' respectively. The JSON response is identical in both cases:

```
{
  "id": 1000,
  "from": "USD",
  "to": "INR",
  "conversionMultiple": 65,
  "port": 8001
}
```

Lecture 72

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

Step1: Add dependency in pom.xml

The screenshot shows the Eclipse IDE interface with the 'pom.xml' file open. The file contains the following XML code:

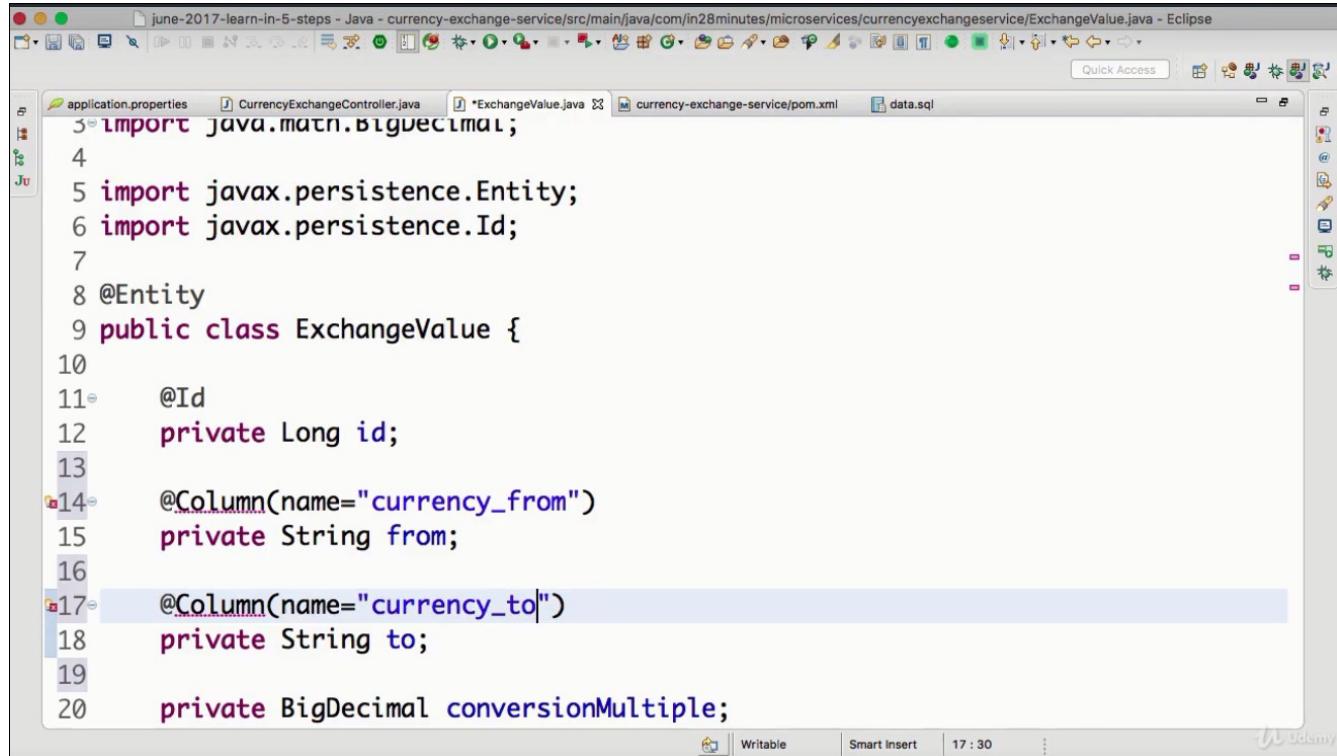
```

<dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactID>h2</artifactID>
</dependency>

```

A pink oval highlights the last three dependency blocks (lines 40 to 48). The Eclipse toolbar at the top and various project files like 'application.properties' and 'CurrencyExchangeController.java' are visible.

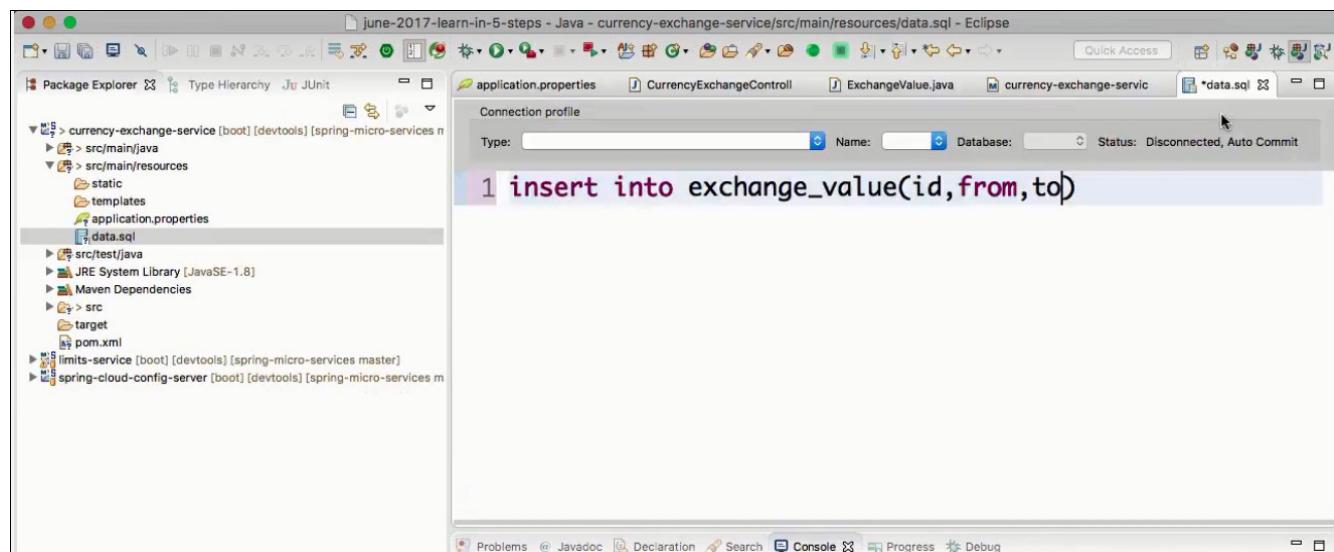
Step2: Add Entity



The screenshot shows the Eclipse IDE interface with the file `ExchangeValue.java` open in the editor. The code defines a Java entity class `ExchangeValue` with fields `id`, `from`, `to`, and `conversionMultiple`. The `from` and `to` fields are annotated with `@Column` and have type `String`.

```
import java.math.BigDecimal;
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity
public class ExchangeValue {
    @Id
    private Long id;
    @Column(name="currency_from")
    private String from;
    @Column(name="currency_to")
    private String to;
    private BigDecimal conversionMultiple;
}
```

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



The screenshot shows the Eclipse IDE interface with the file `data.sql` open in the editor. The file contains a single SQL `INSERT` statement:

```
1 insert into exchange_value(id,from,to)
```

```
1 insert into exchange_value(id,currency_from,currency_to,conversion_multiple,por
2 values(10001, 'USD', 'INR',65,0);
3 insert into exchange_value(id,currency_from,currency_to,conversion_multiple,por
4 values(10002, 'EUR', 'INR',75,0);
5 insert into exchange_value(id,currency_from,currency_to,conversion_multiple,por
6 values(10003, 'AUD', 'INR',25,0);
```

Step4: Restart the application

Step5: Verify that the table is created in the H2O console:

URL:<http://localhost:8000/h2-console>

Verify that JDBC URL has value **jdbc:h2:mem:testdb**

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

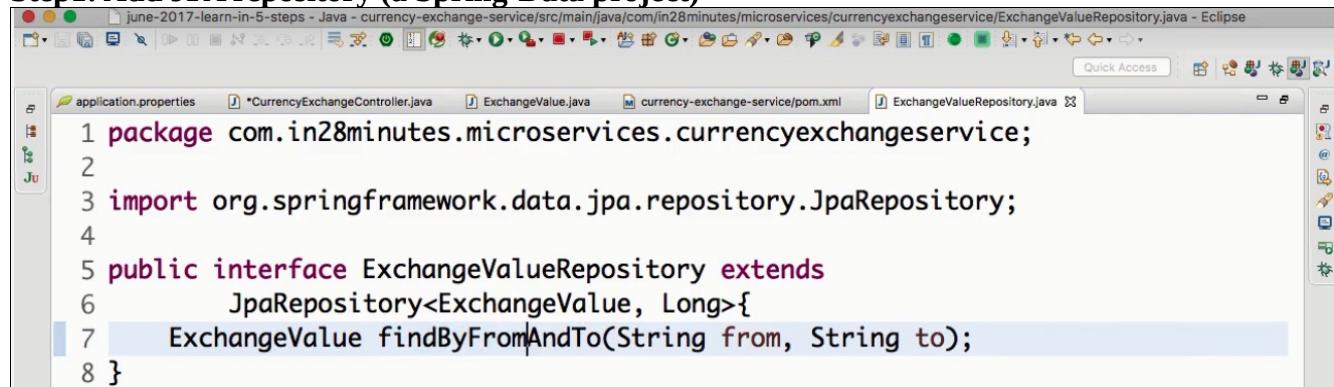
Sample SQL Script

```
Delete the table if it exists      DROP TABLE IF EXISTS TEST;
Create a new table                CREATE TABLE TEST(ID INT PRIMARY KEY);
```

10 people bookmarked this moment.

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

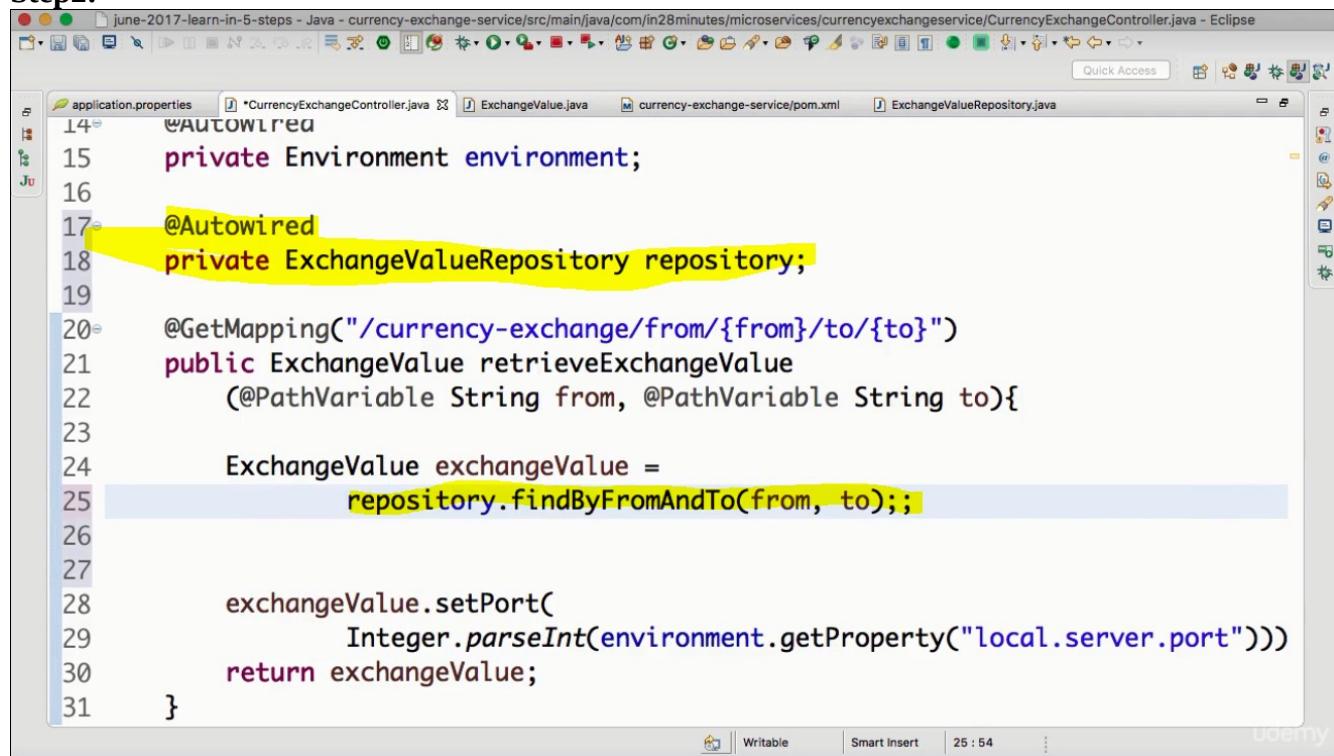
Step1: Add JPA repository (a Spring-Data project)



```
1 package com.in28minutes.microservices.currencyexchangeservice;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface ExchangeValueRepository extends
6     JpaRepository<ExchangeValue, Long>{
7     ExchangeValue findByFromAndTo(String from, String to);
8 }
```

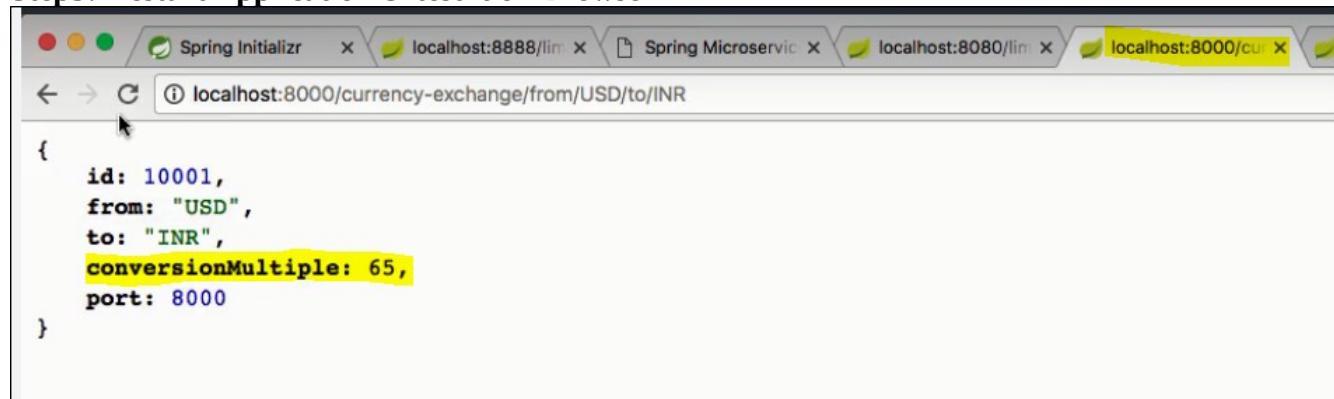
** findByFromAndTo → JPA in Spring -Data will implement the method to find By From & To.

Step2:



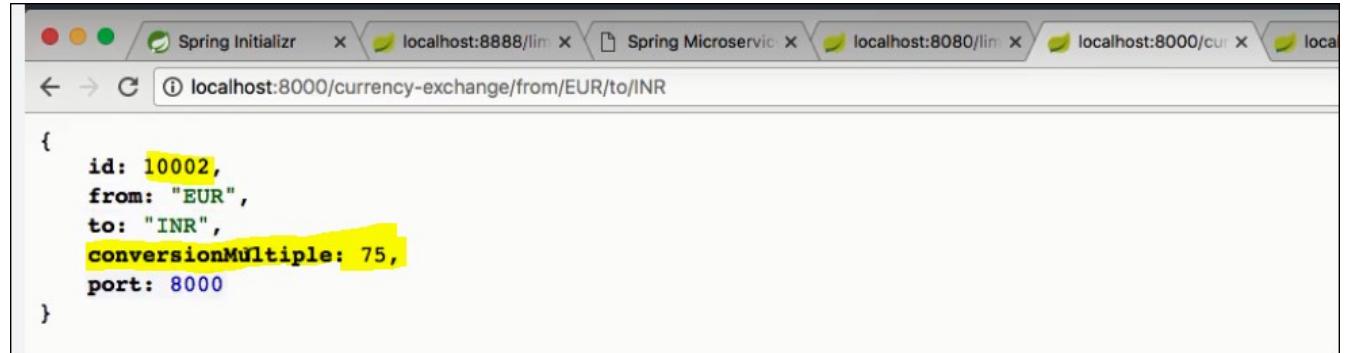
```
14 @Autowired
15 private Environment environment;
16
17 @Autowired
18 private ExchangeValueRepository repository;
19
20 @GetMapping("/currency-exchange/from/{from}/to/{to}")
21 public ExchangeValue retrieveExchangeValue
22     (@PathVariable String from, @PathVariable String to){
23
24     ExchangeValue exchangeValue =
25         repository.findByFromAndTo(from, to);;
26
27
28     exchangeValue.setPort(
29         Integer.parseInt(environment.getProperty("local.server.port")));
30
31 }
```

Step3: Restart Application & test it on Browser



```
{ "id": 10001, "from": "USD", "to": "INR", "conversionMultiple: 65, "port": 8000 }
```

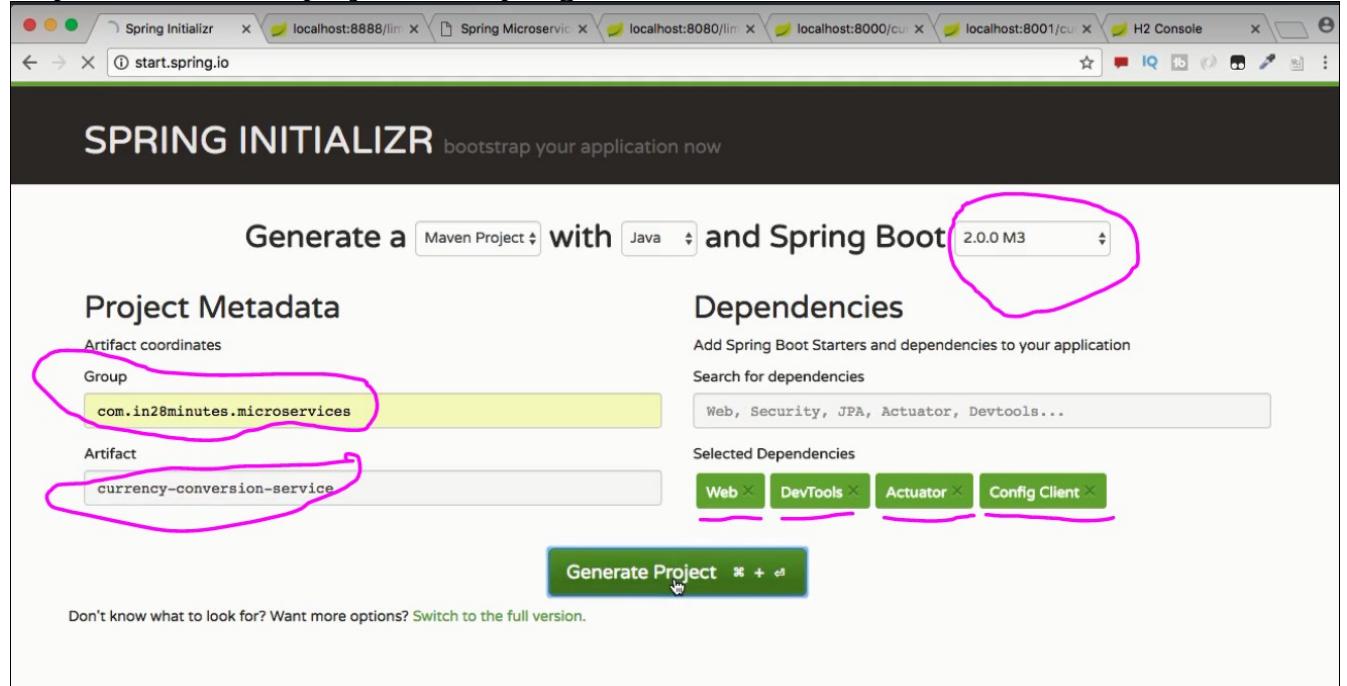
Test it with another From & To:



```
{  
  id: 10002,  
  from: "EUR",  
  to: "INR",  
  conversionMultiple: 75,  
  port: 8000  
}
```

Lecture 74: Setup Currency Conversion Microservice

Step1: Download the project from Spring boot initializer



Generate a with and Spring Boot

Project Metadata

Artifact coordinates

Group

Artifact

Dependencies

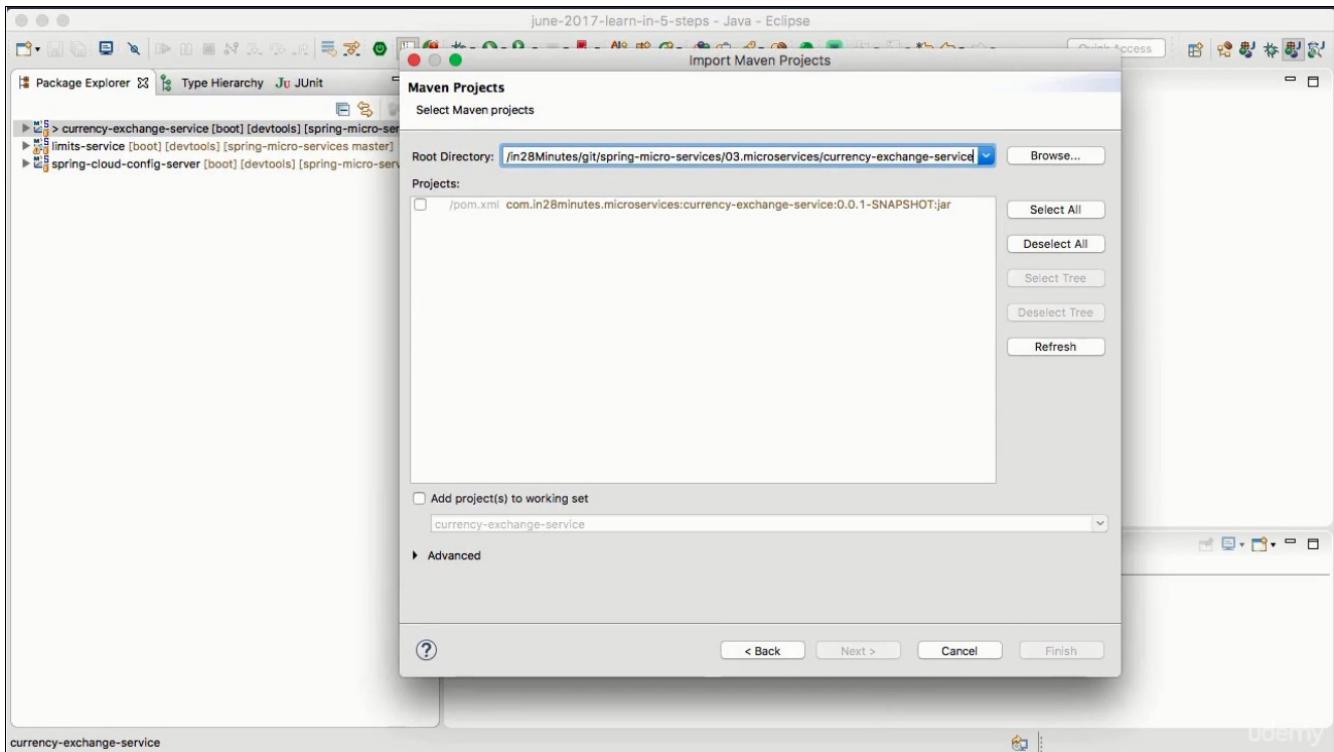
Add Spring Boot Starters and dependencies to your application

Search for dependencies

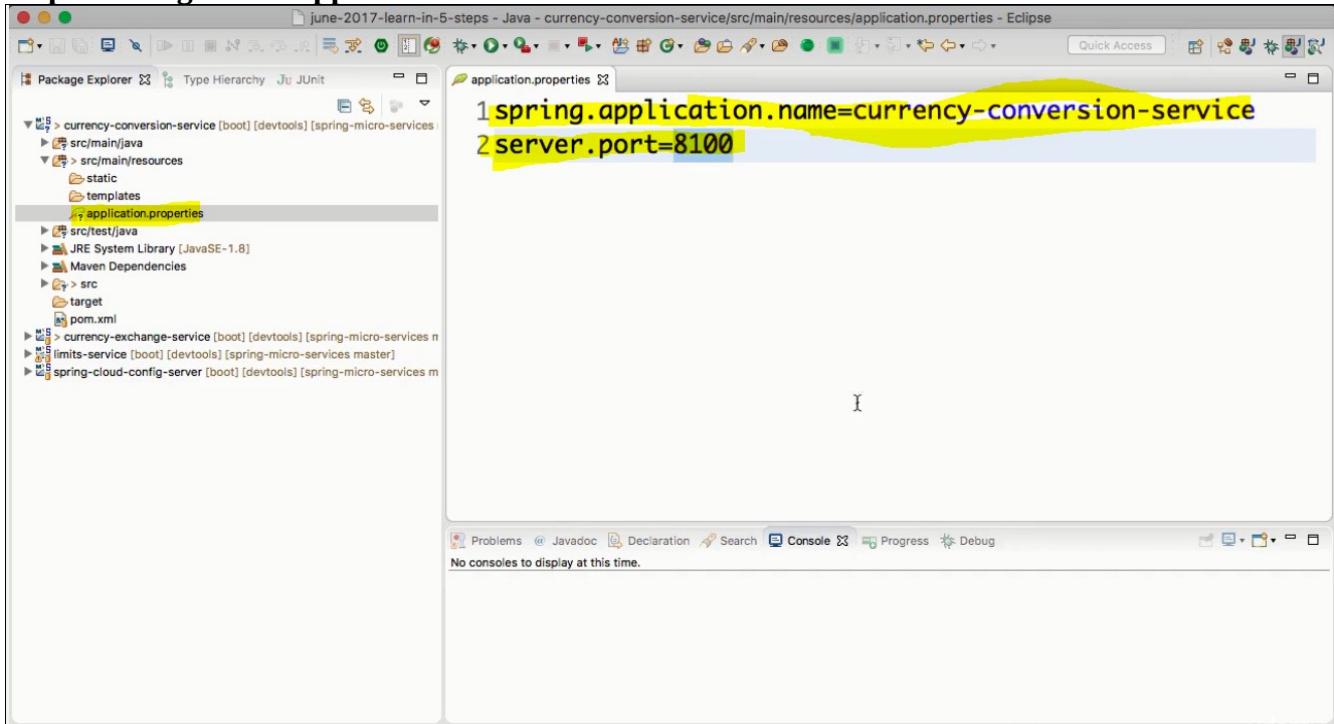
Selected Dependencies

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.

The screenshot shows the Eclipse IDE interface with the title bar "june-2017-learn-in-5-steps - Java - currency-conversion-service/src/main/java/com/in28minutes/microservices/currencyconversionservice/CurrencyConversionController.java - Eclipse". The code editor displays the following Java code:

```
4
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PathVariable;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @RestController
10 public class CurrencyConversionController {
11
12     @GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")
13     public CurrencyConversionBean convertCurrency(@PathVariable String from,
14             @PathVariable String to,
15             @PathVariable BigDecimal quantity
16     ){
17         return new CurrencyConversionBean(1L,from,to,BigDecimal.ONE,quantity,qu
18     }
19
20 }
21
```

A tooltip at the bottom of the code editor says "6 people bookmarked this moment." and shows the time "17:1". The "udemy" logo is visible in the bottom right corner.

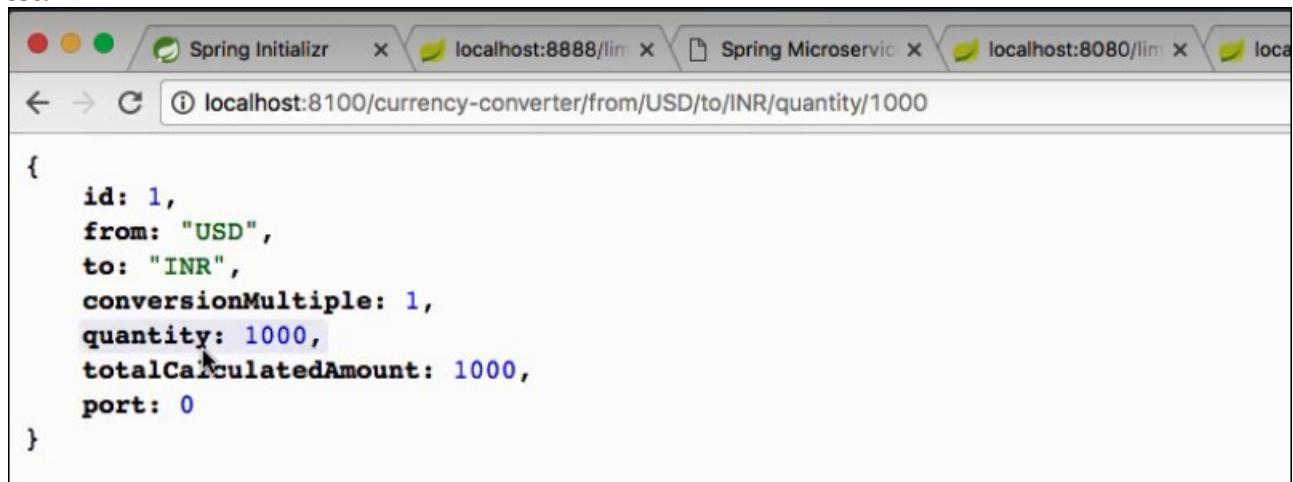
Bean:

The screenshot shows the Eclipse IDE interface with the title bar "june-2017-learn-in-5-steps - Java - currency-conversion-service/src/main/java/com/in28minutes/microservices/currencyconversionservice/CurrencyConversionBean.java - Eclipse". The code editor displays the following Java code:

```
1
2 import java.math.BigDecimal;
3
4 public class CurrencyConversionBean {
5     private Long id;
6     private String from;
7     private String to;
8     private BigDecimal conversionMultiple;
9     private BigDecimal quantity;
10    private BigDecimal totalCalculatedAmount;
11    private int port;
12
13    public CurrencyConversionBean() {
14
15    }
16
17    public CurrencyConversionBean(Long id, String from, String to, BigDecimal c
18        BigDecimal totalCalculatedAmount, int port) {
```

The code editor has a "Save" button at the bottom left. The "udemy" logo is visible in the bottom right corner.

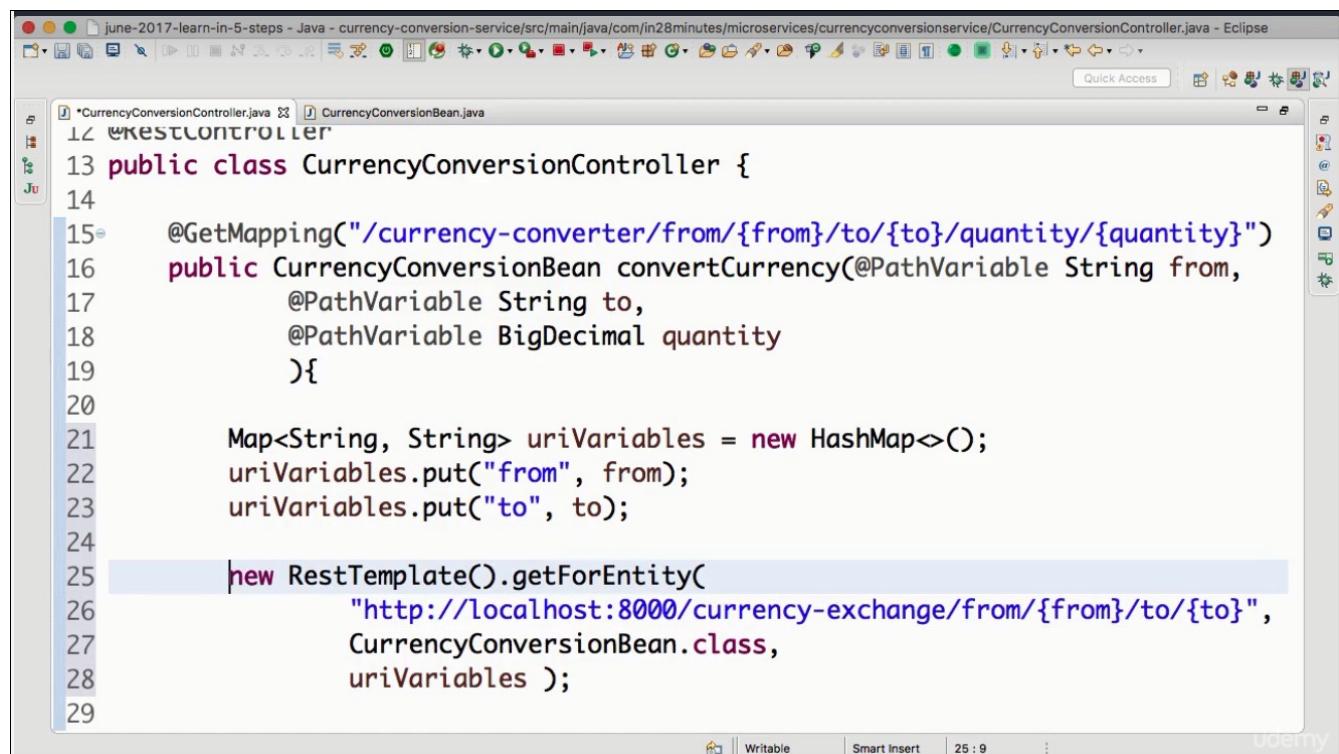
Test:



The screenshot shows a browser window with multiple tabs open. The active tab is labeled 'localhost:8100/currency-converter/from/USD/to/INR/quantity/1000'. The page content displays a JSON object:

```
{  
  id: 1,  
  from: "USD",  
  to: "INR",  
  conversionMultiple: 1,  
  quantity: 1000,  
  totalCalculatedAmount: 1000,  
  port: 0  
}
```

Lecture 76: Invoke Currency Exchange Service from Currency Converter service



The screenshot shows the Eclipse IDE interface with the 'CurrencyConversionController.java' file open. The code implements a REST controller to handle currency conversion requests:

```
13 public class CurrencyConversionController {  
14  
15     @GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")  
16     public CurrencyConversionBean convertCurrency(@PathVariable String from,  
17                                                 @PathVariable String to,  
18                                                 @PathVariable BigDecimal quantity  
19     ){  
20  
21         Map<String, String> uriVariables = new HashMap<>();  
22         uriVariables.put("from", from);  
23         uriVariables.put("to", to);  
24  
25         new RestTemplate().getForEntity(  
26             "http://localhost:8000/currency-exchange/from/{from}/to/{to}",  
27             CurrencyConversionBean.class,  
28             uriVariables );  
29     }
```

```

22     Map<String, String> uriVariables = new HashMap<>();
23     uriVariables.put("from", from);
24     uriVariables.put("to", to);
25
26     ResponseEntity<CurrencyConversionBean> responseEntity = new RestTemplate()
27         .exchange(
28             "http://localhost:8000/currency-exchange/from/{from}/to/{to}",
29             HttpMethod.GET,
30             null,
31             CurrencyConversionBean.class,
32             uriVariables );
33
34     CurrencyConversionBean response = responseEntity.getBody();
35
36     return new CurrencyConversionBean(response.getId(), from, to, response.get
37         quantity, quantity.multiply(response.getConversionMultiple()), re
38         sponse.getTimestamp());
39 }

```

Check the output:

The screenshot shows a browser window with several tabs open. The active tab's address bar shows the URL `localhost:8100/currency-converter/from/USD/to/INR/quantity/1000`. The page content displays a JSON object representing the conversion result:

```
{
  id: 10001,
  from: "USD",
  to: "INR",
  conversionMultiple: 65,
  quantity: 1000,
  totalCalculatedAmount: 65000,
  port: 8000
}
```

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 .

```
16 @GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")
17 public CurrencyConversionBean convertCurrency(@PathVariable String from,
18     @PathVariable String to,
19     @PathVariable BigDecimal quantity
20 ){
21
22     //Feign - Problem 1
23     Map<String, String> uriVariables = new HashMap<>();
24     uriVariables.put("from", from);
25     uriVariables.put("to", to);
26
27     ResponseEntity<CurrencyConversionBean> responseEntity = new RestTemplate()
28         .getForEntity(
29             "http://localhost:8000/currency-exchange/from/{from}/to/{to}",
30             CurrencyConversionBean.class,
31             uriVariables );
32
33     CurrencyConversionBean response = responseEntity.getBody();
```

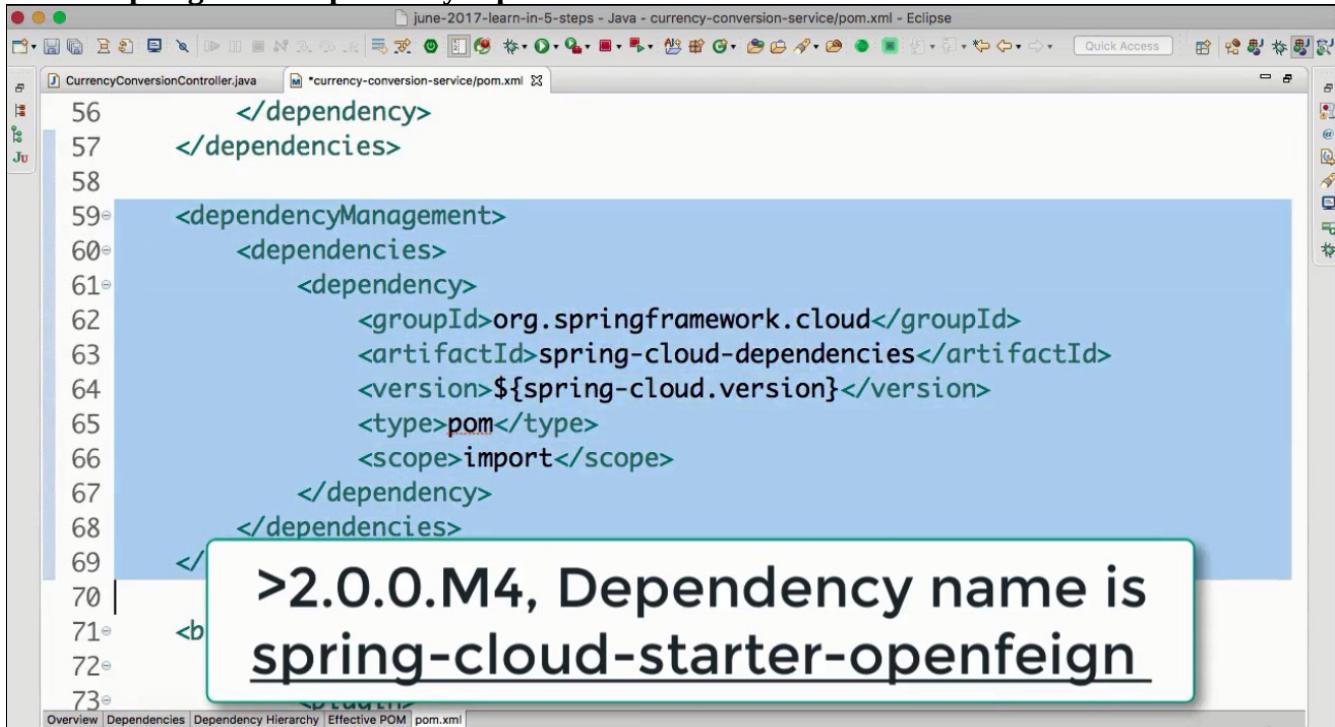
Action5: Add Feign support

Step1:

```
29 <dependency>
30     <groupId>org.springframework.boot</groupId>
31     <artifactId>spring-boot-starter-actuator</artifactId>
32 </dependency>
33 <dependency>
34     <groupId>org.springframework.cloud</groupId>
35     <artifactId>spring-cloud-starter-config</artifactId>
36 </dependency>
37 <dependency>
38     <groupId>org.springframework.cloud</groupId>
39     <artifactId>spring-cloud-starter-feign</artifactId>
40 </dependency>
```

>2.0.0.M4, Dependency name is
spring-cloud-starter-openfeign

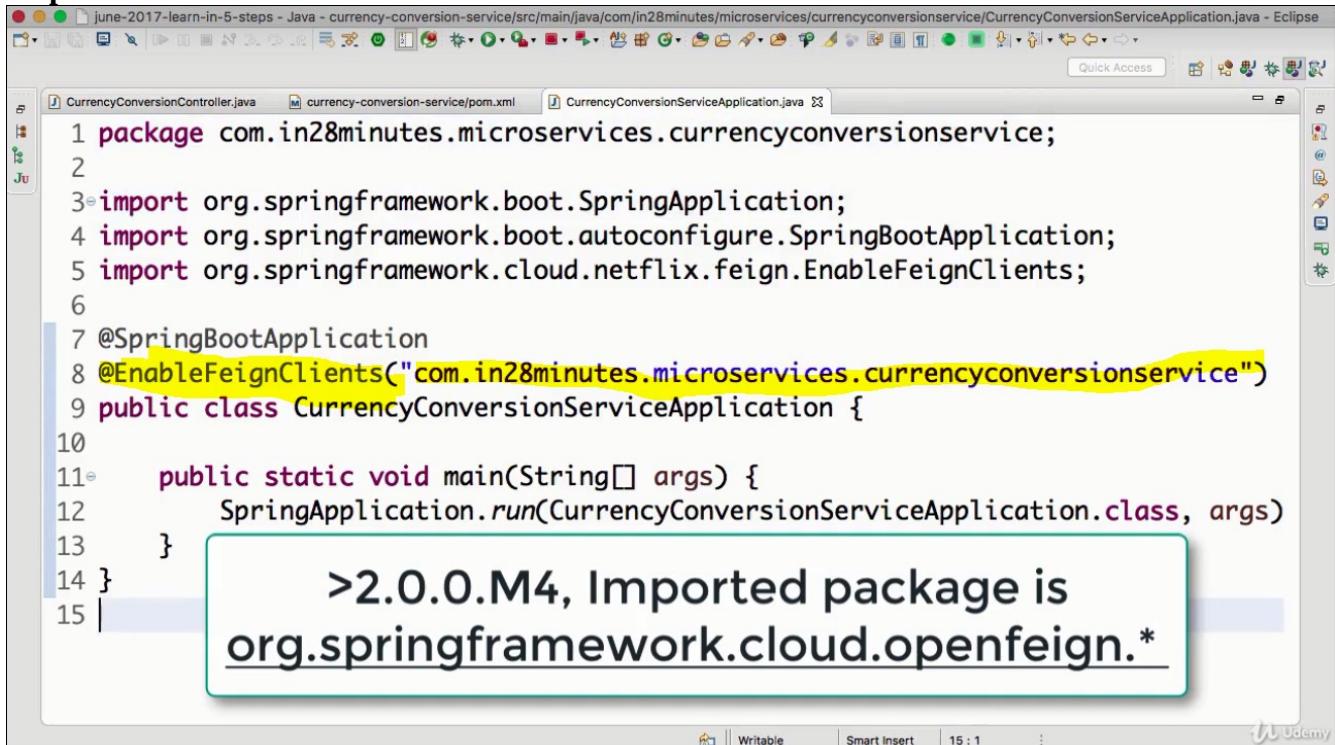
Ensure Spring-cloud dependency is present:



```
56      </dependency>
57  </dependencies>
58
59  <dependencyManagement>
60      <dependencies>
61          <dependency>
62              <groupId>org.springframework.cloud</groupId>
63              <artifactId>spring-cloud-dependencies</artifactId>
64              <version>${spring-cloud.version}</version>
65              <type>pom</type>
66              <scope>import</scope>
67          </dependency>
68      </dependencies>
69
70  <b><b>
71  <b><b>
72
73
```

>2.0.0.M4, Dependency name is spring-cloud-starter-openfeign

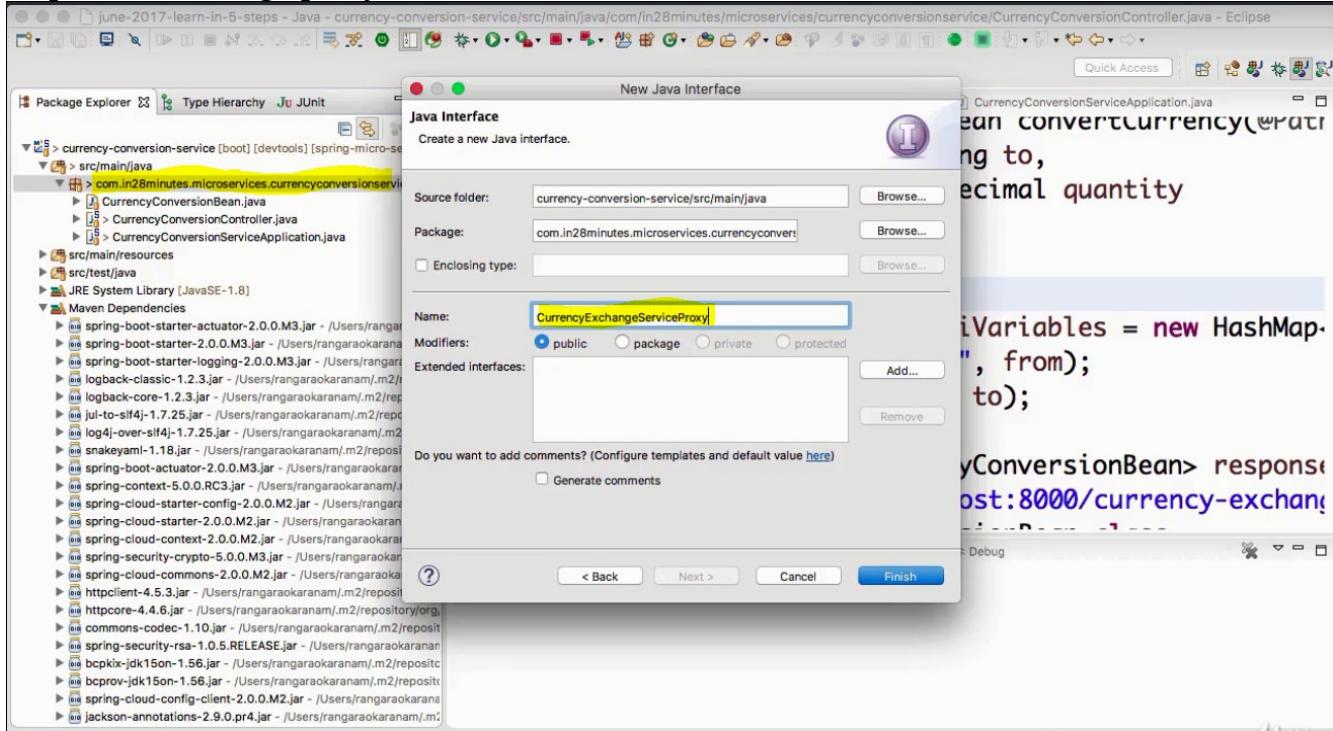
Step2:



```
1 package com.in28minutes.microservices.currencyconversionservice;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.feign.EnableFeignClients;
6
7 @SpringBootApplication
8 @EnableFeignClients("com.in28minutes.microservices.currencyconversionservice")
9 public class CurrencyConversionServiceApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(CurrencyConversionServiceApplication.class, args)
13     }
14 }
15
```

>2.0.0.M4, Imported package is org.springframework.cloud.openfeign.*

Step3: Create a feign proxy



```
1 package com.in28minutes.microservices.currencyconversionservice;
2
3 import org.springframework.cloud.netflix.feign.FeignClient;
4
5 @FeignClient(name="currency-exchange-service", url="localhost:8000")
6 public interface CurrencyExchangeServiceProxy {
7
8 }
```

**>2.0.0.M4, Imported package is
org.springframework.cloud.openfeign.***

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

The screenshot shows the Eclipse IDE interface with the title bar "june-2017-learn-in-5-steps - Java - currency-conversion-service/src/main/java/com/in28minutes/microservices/currencyconversionservice/CurrencyExchangeServiceProxy.java - Eclipse". The code editor displays the following Java code:

```
1 package com.in28minutes.microservices.currencyconversionservice;
2
3 import org.springframework.cloud.netflix.feign.FeignClient;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.PathVariable;
6
7 @FeignClient(name="currency-exchange-service", url="localhost:8000")
8 public interface CurrencyExchangeServiceProxy {
9
10    @GetMapping("/currency-exchange/from/{from}/to/{to}")
11    public CurrencyConversionBean retrieveExchangeValue
12        (@PathVariable("from") String from, @PathVariable("to") String to);
13 }
14
```

Step2: Add the ref in the Controller

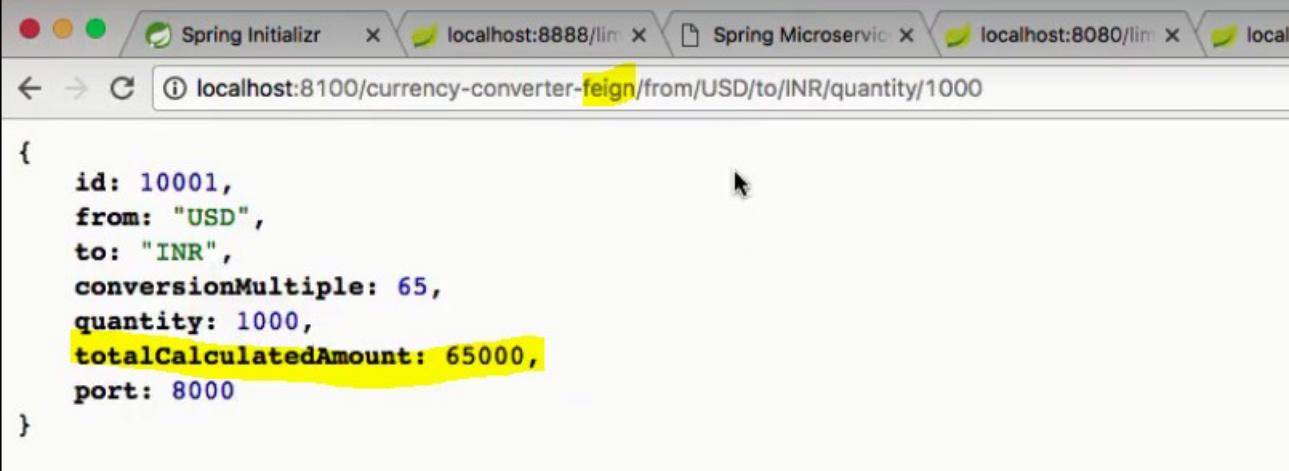
```
14 @RestController
15 public class CurrencyConversionController {
16
17     @Autowired
18     private CurrencyExchangeServiceProxy proxy;
```

Step3: Copy the existing convertCurrency to convertCurrencyFeign

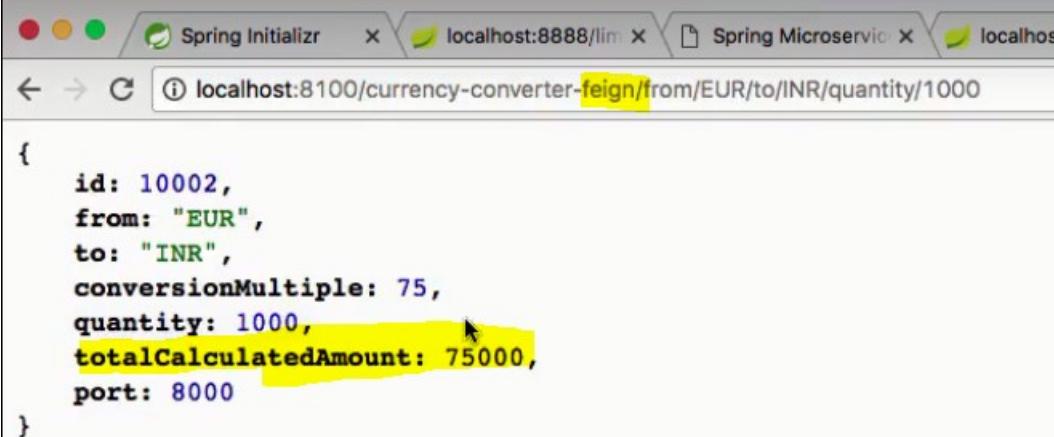
The screenshot shows the Eclipse IDE interface with the title bar "june-2017-learn-in-5-steps - Java - currency-conversion-service/src/main/java/com/in28minutes/microservices/currencyconversionservice/CurrencyConversionController.java - Eclipse". The code editor displays the following Java code, with several lines highlighted in yellow:

```
32
33     CurrencyConversionBean response = responseEntity.getBody();
34
35     return new CurrencyConversionBean(response.getId(), from, to, response.
36         quantity.multiply(response.getConversionMultiple()), response.g
37     }
38
39     @GetMapping("/currency-converter-feign/from/{from}/to/{to}/quantity/{quanti
40     public CurrencyConversionBean convertCurrencyFeign(@PathVariable String fro
41         @PathVariable BigDecimal quantity) {
42
43         CurrencyConversionBean response = proxy.retrieveExchangeValue(from, to)
44
45         return new CurrencyConversionBean(response.getId(), from, to, response.
46             quantity.multiply(response.getConversionMultiple()), response.g
47     }
48
49 }
```

Step4: Restart CurrencyConverterService and test:



```
{  
  id: 10001,  
  from: "USD",  
  to: "INR",  
  conversionMultiple: 65,  
  quantity: 1000,  
  totalCalculatedAmount: 65000,  
  port: 8000  
}
```



```
{  
  id: 10002,  
  from: "EUR",  
  to: "INR",  
  conversionMultiple: 75,  
  quantity: 1000,  
  totalCalculatedAmount: 75000,  
  port: 8000  
}
```