Lab 1

Take the sample Customers and Suppliers code and use it to create a RESTful service that:

1) Responds to GET /MyService/v1/Customers with a text/plain listing of all Customers, one per line
2) Responds to GET /MyService/v1/Customers/001 with a text/plain listing of the single customer with id 001
3) Ensure that GET /MyService/v1/Customers/Fred does not invoke any method; use a regular expression to achieve this
4) Responds to DELETE /MyService/v1/Customers/001 by deleting the single customer with id 001
5) Responds to GET /MyService/v1/Customers?query=name.eq.Fred+Jones with a text/plain representation of the customer with the name Fred Jones
6) In each case, test your service by using POSTMAN in Chrome

Lab 2

Update your previous service:

1. Respond to GET /Myservice/v1/Suppliers/001 with a text/plain description of Suppliers 001
2. Respond to GET /MyService/v1/Customers/001/Suppliers/1 by returning a text/plain representation of the first supplier with a relationship with Customer 001
3. Respond to GET /MyService/Suppliers by returning a text/html <ul> type list that contains <a href=…> elements that are links to each Supplier detail
4. Arrange that the user must be logged in, and in an administrative role, to be able to perform the DELETE operation listed in Lab 1 item 4

Lab 3

Continue to update your service:

1. Respond to GET /MyService/v1/Customers/999 (or any out of range ID value) with a status code of 404 Not Found
2. Modify the method further so that it returns XML or JSON according to the requested Accept type from the client. Modify the @Produces annotation of this method to indicate that both XML and JSON are offered, and that the method no longer offers text/plain.

Lab 4

Continue to update your service, choose one of these new operations to implement. Only implement the second if you have spare time:

1. Create a method that responds to POST /MyService/v1/Customers that accepts either XML or JSON as the body of the message (entity) and creates a new record in the Customers table. Verify that you can see the newly created element in the listing by doing a GET /MyService/v1/Customers. Note that you can create a new random UUID object with the static method UUID.randomUUID(). The insert method should return a status code 201 Created, and return the URI at which the new Customer may be viewed, e.g. /MyService/v1/Customers/0000…9. (The textual format of a UUID is quite long!)
2. Create a method that responds to a PUT /MyService/v1/Customers/000…5/Relationship/000…2 where the second UUID represents the primary key of a Supplier. The method creates a new relationship between the indicated Customer and Supplier. If successful, the method should return the URI that would lead to the particular Supplier through the Customer. Remember that these URIs (/MyService/v1/Customer/000…5/Suppliers/3 take a simple index of the position in that Customer's sequence of relationships, not the UUID of the Supplier in the relationship.

3. Optional:

   Create a Provider that handles hexadecimal numbers, using a content type of application/hex. Implement both MessageBodyWriter<Integer> and MessageBodyReader<Integer>. When checking the class types, take care that Integer.CLASS refers to the class type of an Integer object, and Integer.TYPE describes the class of the primitive int.

   Next, create a new root resource in your service that has a method that consumes text/plain and produces application/hex, and another that processes the inverse content types. Receive the body of the request as an int for the method that consumes application/hex, and return value as a String. Perform this conversion int -> String simply by concatenating "" with the number. For the method that consumes text/plain, receive the entity as a String, convert it to an int using Integer.parseInt, then return the resulting int value through the Response entity. Use the POSTMAN client to configure the Accept and Content-Type headers. You should see that you can send text/plain and Accept application/hexadecimal causing a conversion from decimal to hex. If you send a Content-Type of application/hexadecimal and Accept text/plain, the same method should convert from hex to decimal.

Lab 5

Continue to modify your existing service:

1. Generate the XSD schema file that defines your Customer objects in XML format.

2. Temporarily: Modify the XML output so that the Customer name field is called “cust-name”
3. Temporarily: Modify the XML output so that the Customer id field is not included.
4. Restore your original XML handling
5. Modify the method that handles GET /MyService/v1/Customers so that it returns XML. Modify it so that it accepts a query parameter “elements” which has a value that is a comma separated list of field names within the customer. Arrange to return a modified data set so that only the requested fields are part of the response.