HU Extension Assignment 08 E-90 Cloud Computing

Handed out: 10/23/2015 Due by 11:59 PM on Friday, 10/30/2015

FINAL

Place all of your narratives and illustrations in a single Word or PDF document named E90_LastNameFirstNameHW08.docx [.pdf]. Use this assignment as the initial template. Please implement code solutions as a separate class in one (Java, C#, Ruby, Python,..) project. Add your steps and your code below problem statements used for that problem. Upload your homework file and your working code (e.g., filename.java) into your Assignment 8 folder. Do not include executables. Please also do not zip your files.

The following requires installation and configuration of Apache Tomcat on your Windows or Mac server as well as in Eclipse or IDE which are necessary configurations for this problem set.

Download Tomcat's Binary distribution from http://tomcat.apache.org/download-80.cgi
Select zip file for your system (Windows or MAC). Install Tomcat by expanding the zip file to: c:\ e.g. `c:\apache_tomcat_8.0.28`. Create the environmental variable `CATALINA_HOME` with that value. You must have environmental variable `JAVA_HOME` set to the installation of your Java6/7/8 JDK not JRE. You start Tomcat by running `startup.bat` file in `Tomcat bin` directory. You stop Tomcat by running `shutdown.bat`.

Download the Apache Tomcat 8.0 package to my Mac OS. Here I download and extract it to my local path "/usr/local/apache-tomcat-8.0.28". I also create a symbolic link to it. Since the default CATALINA_HOME is set to "Library/Tomcat" as default. I don't need to set it again.

```
hqiu@bos-mpdei>> sudo ln -s /usr/local/apache-tomcat-8.0.28 /Library/Tomcat
```

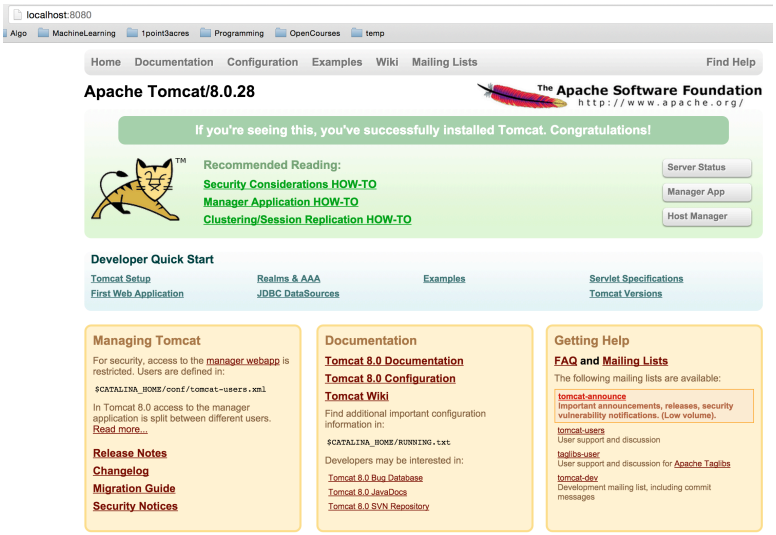
Give the executable permission to the scripts.

```
hqiu@bos-mpdei>> sudo chmod a+x /Library/Tomcat/bin/*.sh
```

Start the service and check from the browser. Shut down the service after the check. We will start the Tomcat service through Eclipse later.

```
hqiu@bos-mpdei>> /Library/Tomcat/bin/startup.sh
Using CATALINA_BASE:   /Library/Tomcat
Using CATALINA_HOME:   /Library/Tomcat
Using CATALINA_TMPDIR: /Library/Tomcat/temp
Using JRE_HOME:        /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home
Using CLASSPATH:       /Library/Tomcat/bin/bootstrap.jar:/Library/Tomcat/bin/tomcat-juli.jar
Tomcat started.
hqiu@bos-mpdei>> /Library/Tomcat/bin/shutdown.sh
Using CATALINA_BASE:   /Library/Tomcat
Using CATALINA_HOME:   /Library/Tomcat
Using CATALINA_TMPDIR: /Library/Tomcat/temp
Using JRE_HOME:        /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home
Using CLASSPATH:       /Library/Tomcat/bin/bootstrap.jar:/Library/Tomcat/bin/tomcat-juli.jar
```

Check from the browser. We can reach Tomcat in http://localhost:8080.



Go to http://jersey.java.net/download.html and download file called `jersey-archive1.19.jar` file with Oracle implementation of JAX-RS. The name of the hyper link is: `Jersey 1.19 JAR bundle.` Once you start building your Eclipse Dynamic Web Projects, you will add that `jar` to your Build Path. You will also copy that `jar` file (Jersey 1.19 bundle) to your `WebContent\WEB-INF\lib` directory under your Eclipse project. If you decide to modify and compile class `CustomerResource` or class `CustomerResourceClient` from the command line, you will need to have some of those `jars` in your `CLASSPATH,` as well. Your client class is contained in the file `CustomerResourceClient.java.`
Note: File `jax_rs_code.zip` contains java sources of classes discussed in the lecture on RESTFul technology. Files from this zip (src folder, web.xml) will need to be moved to your project.

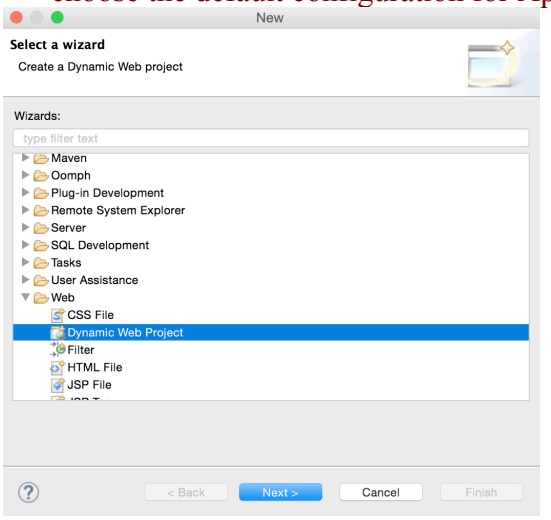
If you are a C# programmer and you are familiar with WCF, you are welcome to run this exercise in C#. In that case, you will not use Java Jersey API but rather WCF classes. You are similarly welcome to do this assignment in any other language of your choice.

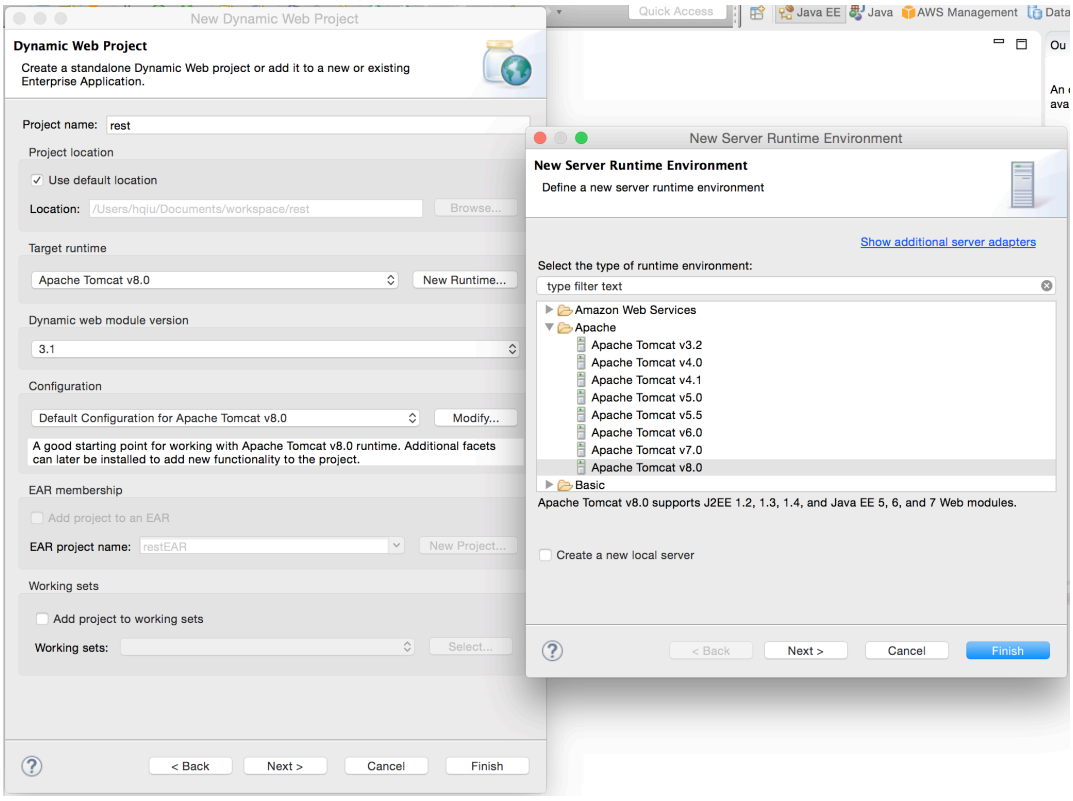
Problem 1:
Open Eclipse in a new workspace. Go to File -> New -> Dynamic Web Project. Call it "rest". This name is arbitrary, however, please do not change it or if you want to change it then modify all URL-s in the client code. Those URLs point to an application called "rest" on the localhost and port 8080. Expand attached ZIP file jax_rs_code.zip. Copy packages `com.rest.client` and `com.rest.ws` from directory `rest` of the expanded ZIP file to the `src` directory under `Java Resources` folder of your project. Similarly, move file `web.xml` to the `WebContet\WEB-INF` directory of your Eclipse project. If your `WEB_INF` directory has a `lib` subdirectory, populate it with the Jersey bundled jar mentioned above (OR if you downloaded the Jersey zip then copy all the libraries). Build your project and fix any errors you might encounter. Run your project on your Tomcat server. You run the project by right clicking on the project and selecting `Run As -> Run on Server`. If your Eclipse is not aware of your

Tomcat, please add new server. If your Tomcat is version 8, select Tomcat-8 as the server type. Subsequently, run your client class `CustomerResourceClient` as a Java application. Show your output. Submit working version of all classes, even if you did not modify a thing.

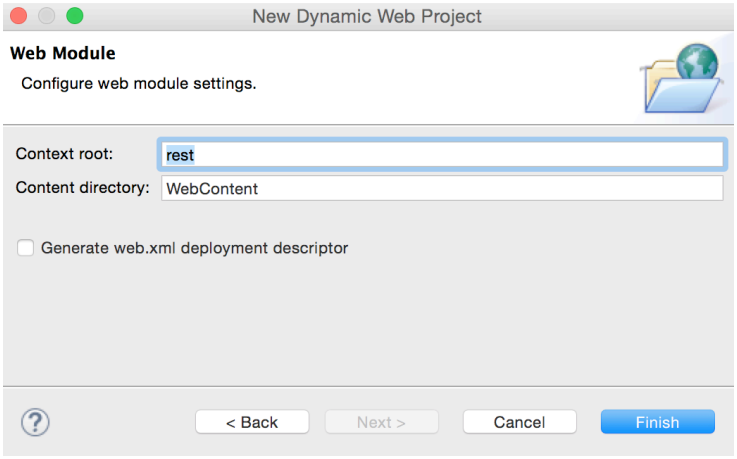
Points: 30

1) Go to "File->New", choose "Dynamic Web Project". Set the "Target runtime" to "Apache Tomcat 8.0". We can create the runtime and set the path of Tomcat through "New Runtime". Set the "Dynamic web module version" to "3.1" and choose the default configuration for Apache Tomcat v8.0".

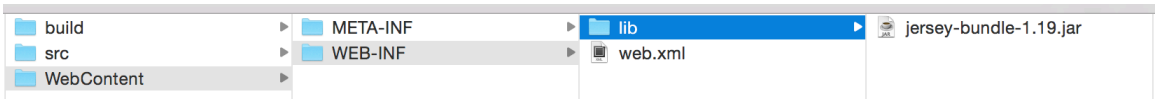


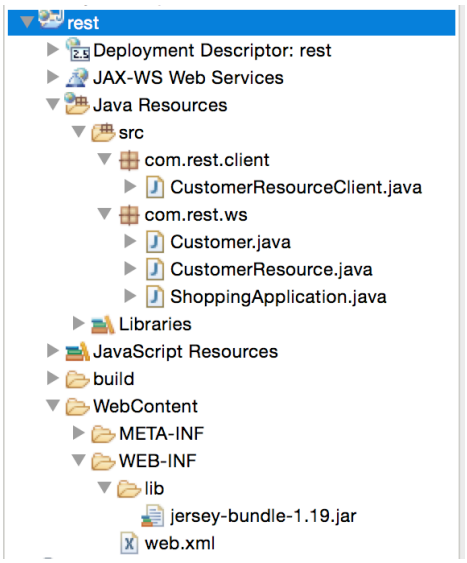


Note that the project name is “rest”. Do not change it if we don’t want to change the URLs in the copied source file. The URLs point to the application called “rest”.

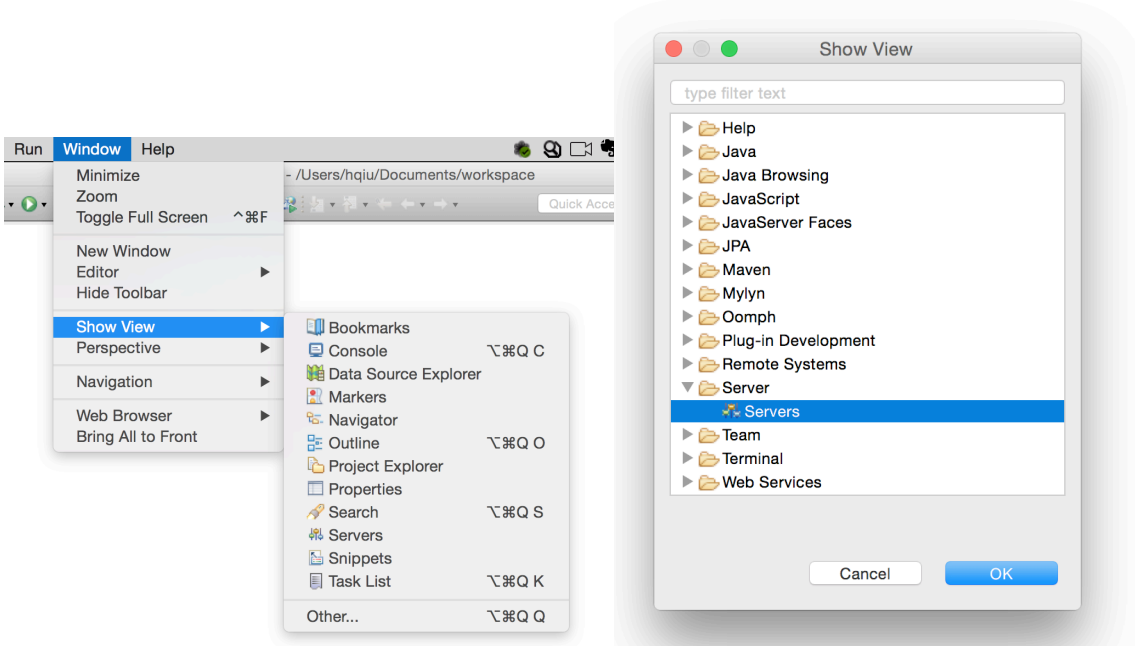


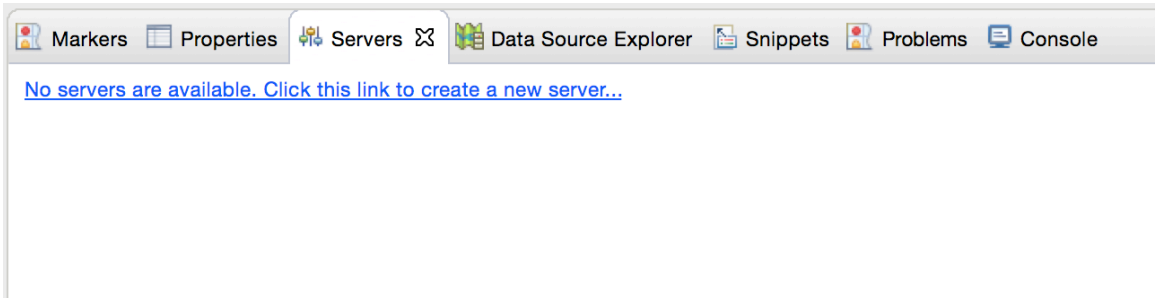
2) Copy the src folder from the example package to the current workspace. So it should have the same structure as the “src” folder from “jax_rs_code.zip”. Similarly, copy “web.xml” to “WebContent/WEB-INF”. Put the Jersey bundled jar “jersey-bundle-1.19.jar” into the “WebContent/WEB-INF/lib” directory. The structure looks like below.



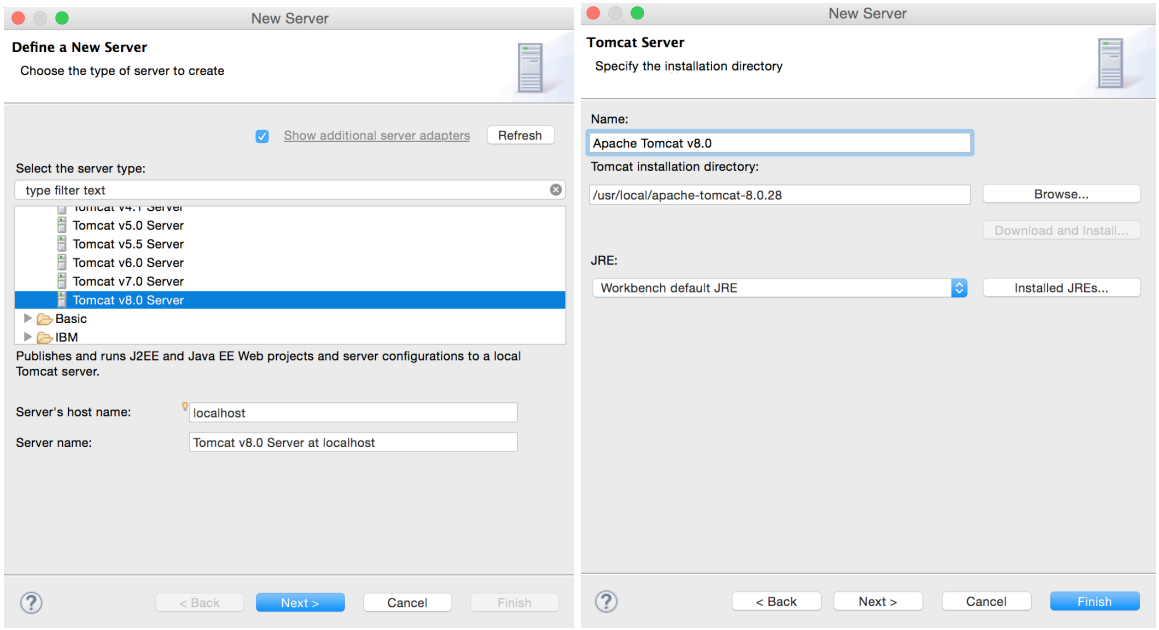


3) Create a Tomcat server through Eclipse. Goto “Window->Show View->Other”. Select “Server/servers”. There will be a “server” view at the bottom of the window.

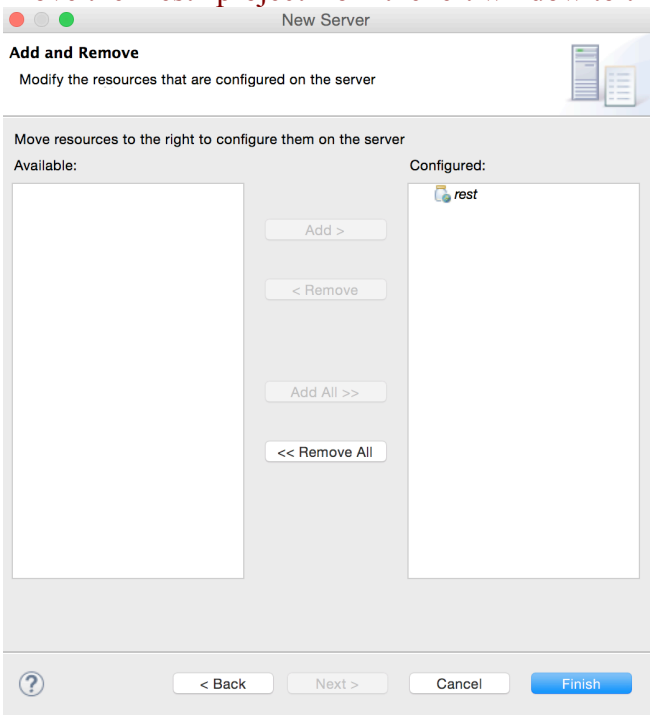




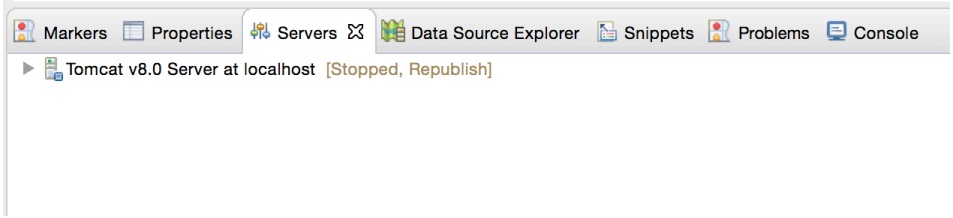
Click the link to create a new server. Choose "Tomcat v8.0 Server".



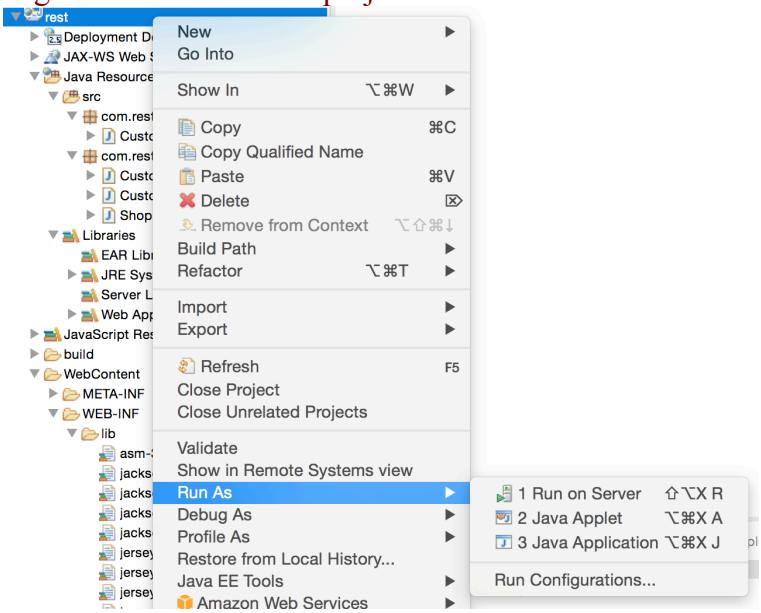
Move the “rest” project from the left window to the right to configure it on the server.

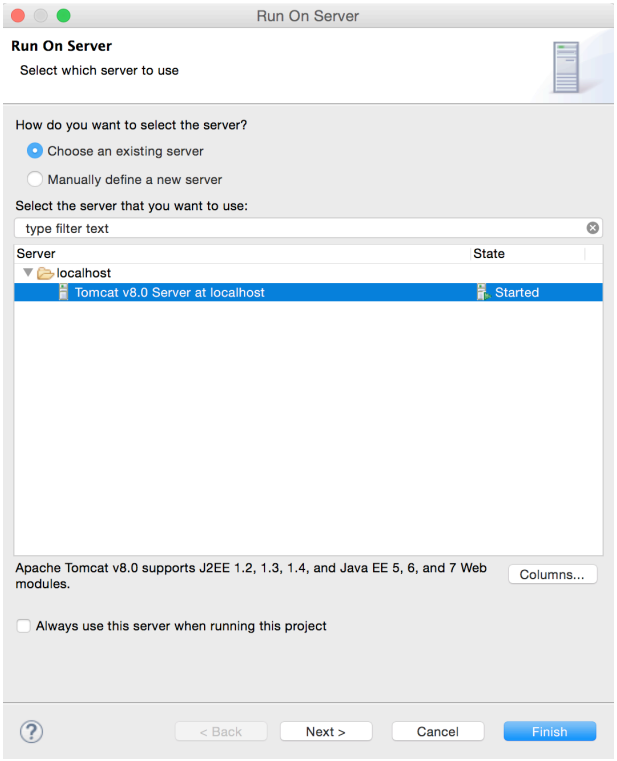


The “rest” project is synchronized on the Tomcat Server.

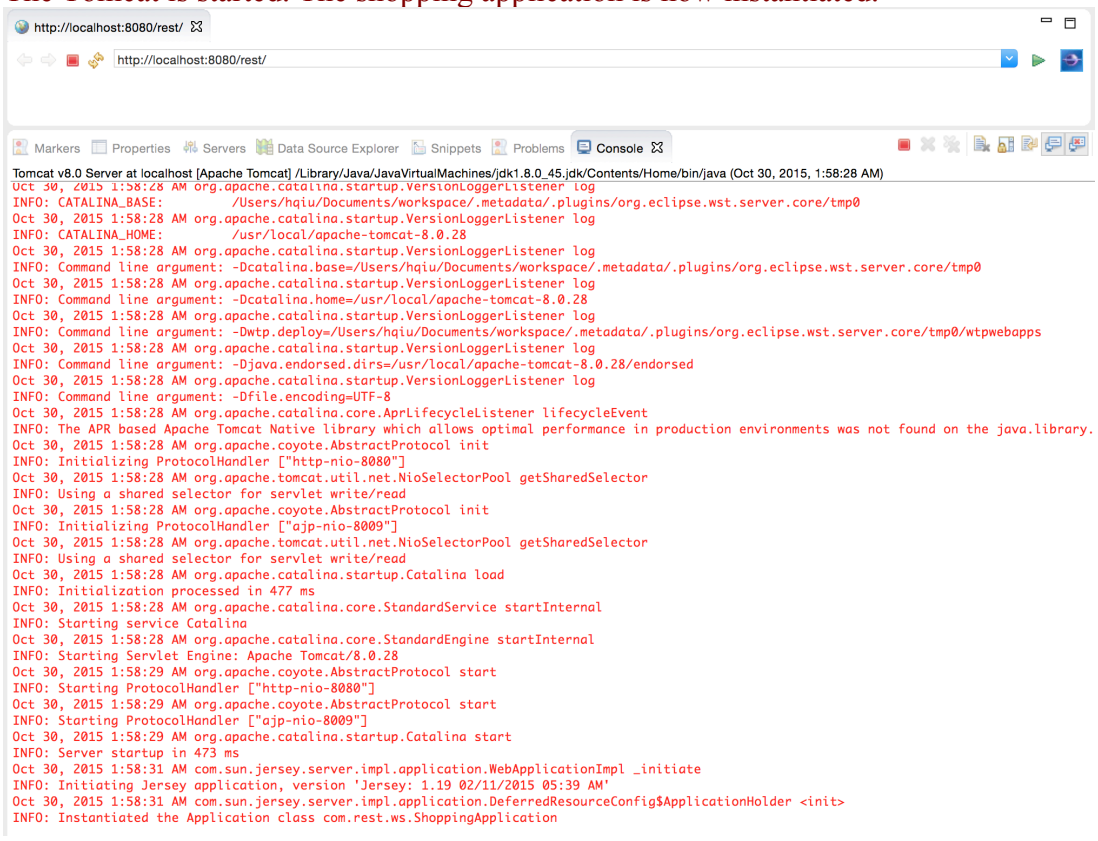


Right click on the “rest” project and select “Run As-> Run on Server”.

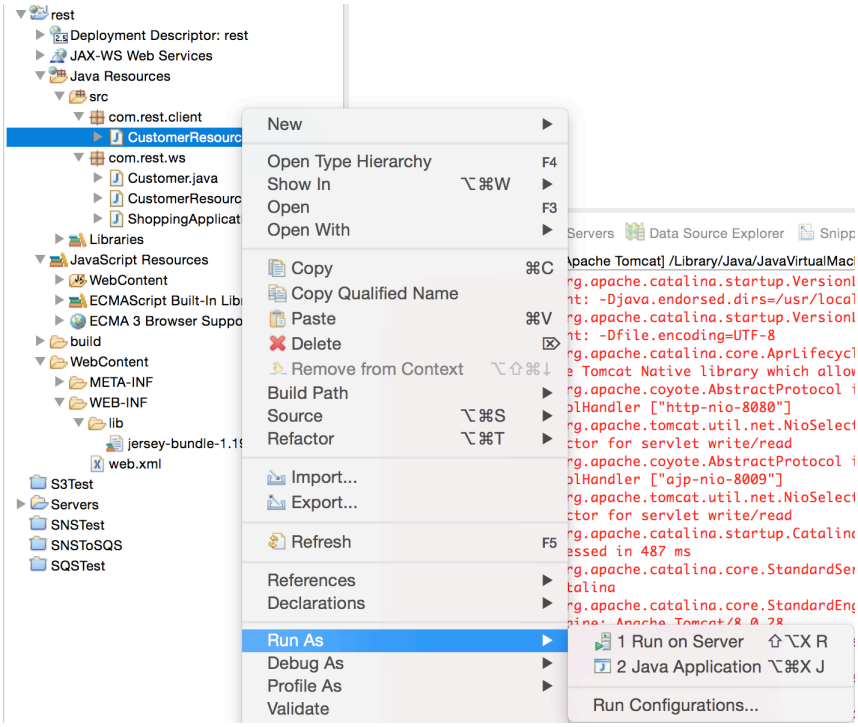




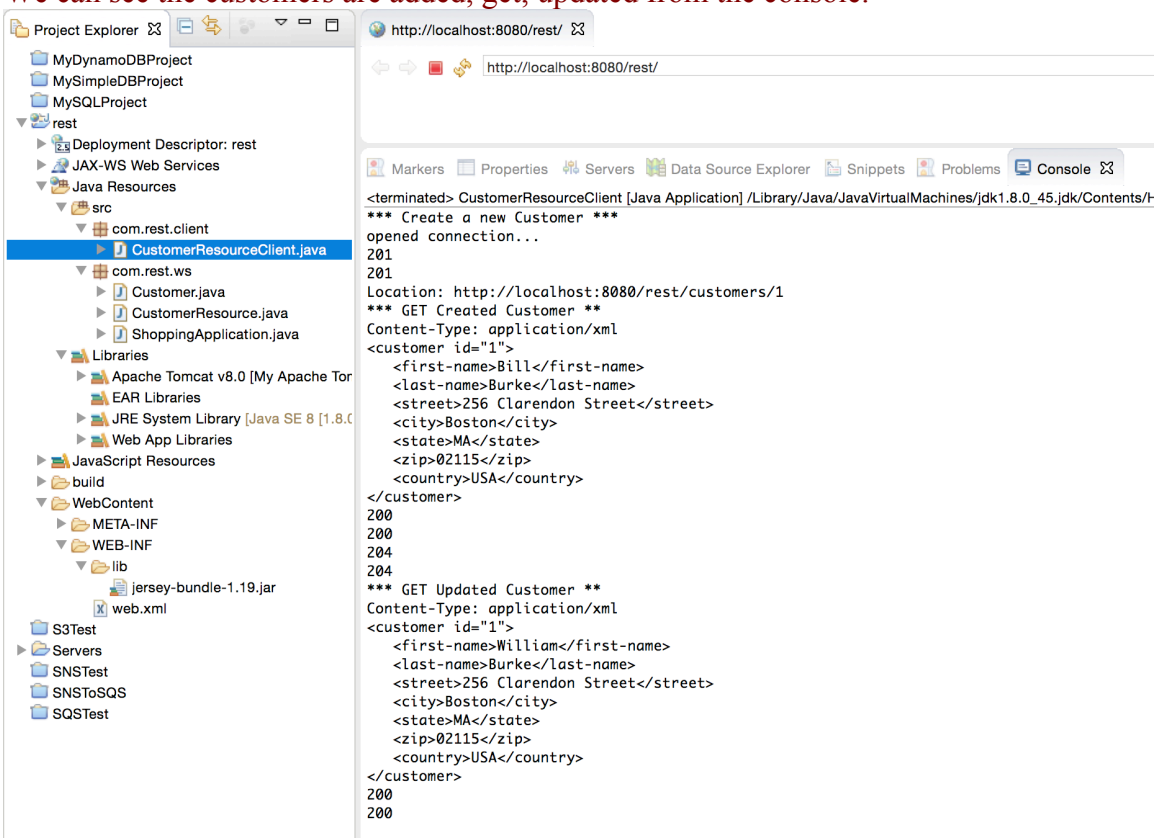
The Tomcat is started. The shopping application is now instantiated.



4) Run the client “CustomerResourceClient” as a Java Application.



We can see the customers are added, get, updated from the console.

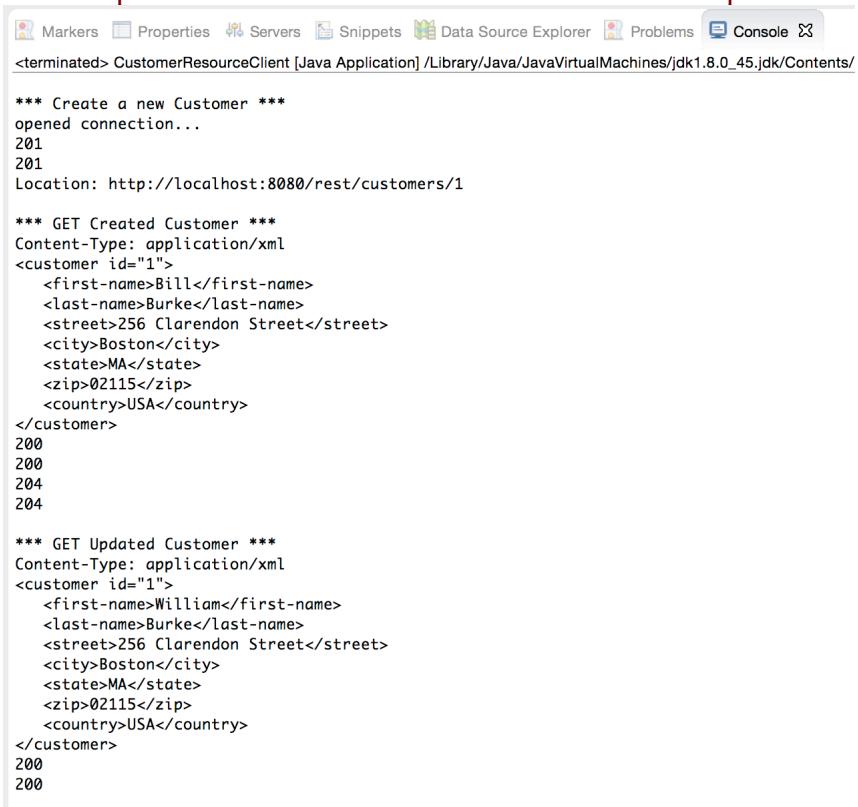


In this problem I haven't modified any file. But I will rename "CustomerResourceClient" class to "CustomerResourceClient_P1" and "CustomerResource" class to "CustomerResource_P1". I will leave the original names for problem 2.

Problem 2:
Modify your `CustomerResource` class so that it could support a request that one particular `Customer` gets deleted. Also, add necessary methods that would allow your resource class to return a specified number of customers (2 or 3 is fine), starting with a particular customer id. To test new resource class you will have to modify your original client class `CustomerResourceClient` so that it creates 3 or 4 new customers before you could start requesting multiples of them back or before you could delete any of them. Code for creating 4 customers could be just four copy-pasted segments of existing code. Change customer names and perhaps their addresses. Run your code. Show your output. Submit the code you modified.
Points: 30

The "CustomerResourceClient.java" is the client side code for this problem. The "CustomerResource.java" is the server side code for this problem.

1) The original example code first sends a "POST" request to insert a record. Then it sends a "GET" request to query the customer information and a "PUT" request to update the customer information. It checks the updated information in the end.



2) We need to change both the server and client sides code. First insert four new customers to the database. We only need to update client side code for this step. Similarly, send "POST" requests to create customer information.

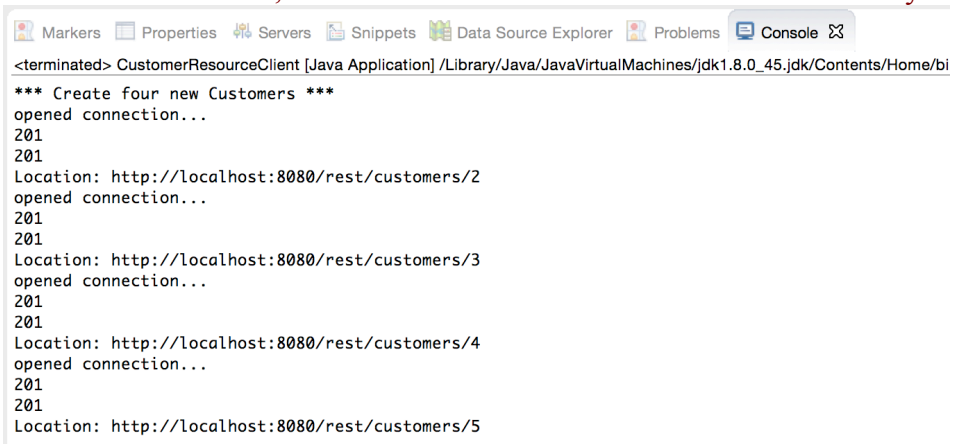
Client side code:

```
// Create four new customers
System.out.println("\n*** Create four new Customers ***");
ArrayList<String> newCustomers = new ArrayList<String>();
newCustomers.add(new String("<customer>" + "<first-name>Tom</first-name>" + "<last-name>Burke</last-name>"
        + "<street>256 Clarendon Street</street>" + "<city>Boston</city>" + "<state>MA</state>"
        + "<zip>02115</zip>" + "<country>USA</country>" + "</customer>"));
newCustomers.add(new String("<customer>" + "<first-name>Jason</first-name>" + "<last-name>Burke</last-name>"
        + "<street>248 Clarendon Street</street>" + "<city>Cambridge</city>" + "<state>MA</state>"
        + "<zip>02478</zip>" + "<country>USA</country>" + "</customer>"));
newCustomers.add(new String("<customer>" + "<first-name>Richael</first-name>" + "<last-name>Burke</last-name>"
        + "<street>123 Clarendon Street</street>" + "<city>Newton</city>" + "<state>MA</state>"
        + "<zip>01827</zip>" + "<country>USA</country>" + "</customer>"));
newCustomers.add(new String("<customer>" + "<first-name>Jane</first-name>" + "<last-name>Burke</last-name>"
        + "<street>234 Clarendon Street</street>" + "<city>Watertown</city>" + "<state>MA</state>"
        + "<zip>02837</zip>" + "<country>USA</country>" + "</customer>"));

for (String customer : newCustomers) {
    postUrl = new URL("http://localhost:8080/rest/customers");
    connection = (HttpURLConnection) postUrl.openConnection();
    System.out.println("opened connection...");
    connection.setDoOutput(true);
    connection.setInstanceFollowRedirects(false);
    connection.setRequestMethod("POST");
    connection.setRequestProperty("Content-Type", "application/xml");
    os = connection.getOutputStream();
    os.write(customer.getBytes());
    os.flush();
    System.out.println(HttpURLConnection.HTTP_CREATED);
    System.out.println(connection.getResponseCode());
    System.out.println("Location: " + connection.getHeaderField("Location"));
    connection.disconnect();
}
```

Server side code:

```
@POST
@Consumes("application/xml")
public Response createCustomer(InputStream is) {
    Customer customer = readCustomer(is);
    customer.setId(idCounter.incrementAndGet());
    customerDB.put(customer.getId(), customer);
    System.out.println("Created customer " + customer.getId());
    return Response.created(URI.create("/" + customer.getId())).build();

}
```

Results from console, "201" shows the customer is created successfully.



I add the following "GET" request to get and display all customers' information. It can help check if we have created or deleted a customer successfully. It will calculate the total number of customers and display all of them.

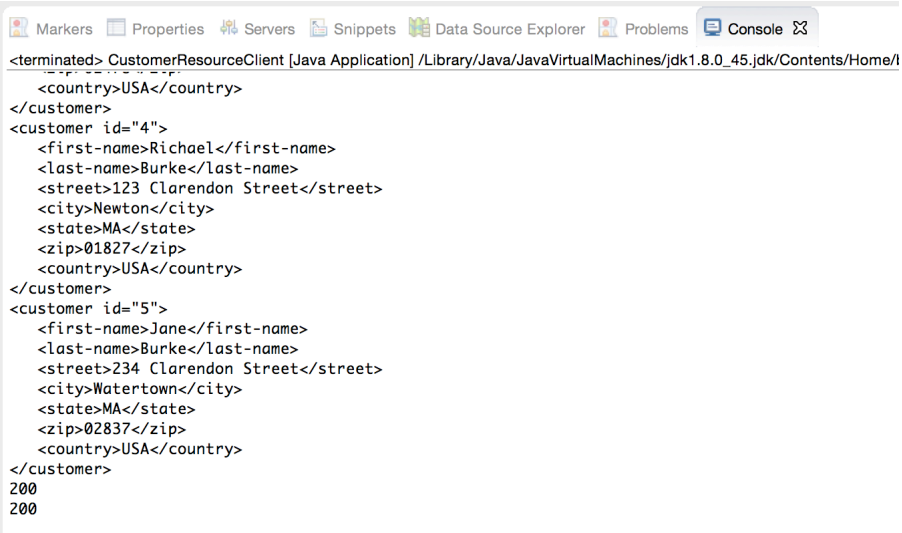
CustomerResource.java:

```
@GET
@Produces("application/xml")
public StreamingOutput getAllCustomers() {
    return new StreamingOutput() {
        public void write(OutputStream outputStream) throws IOException, WebApplicationException {
            int customersToReturn = customerDB.size();
            outputAllCustomers(outputStream, customersToReturn);
        }
    };
}

protected void outputAllCustomers(OutputStream os, int customersToReturn) throws IOException {
    PrintStream writer = new PrintStream(os);
    Collection<Customer> customers = customerDB.values();
    Iterator<Customer> iter = customers.iterator();
    int count = 0;
    while (iter.hasNext() && count < customersToReturn) {
        Customer cust = iter.next();
        writer.println("<customer id=\"" + cust.getId() + "\">");
        writer.println("   <first-name>" + cust.getFirstName() + "</first-name>");
        writer.println("   <last-name>" + cust.getLastName() + "</last-name>");
        writer.println("   <street>" + cust.getStreet() + "</street>");
        writer.println("   <city>" + cust.getCity() + "</city>");
        writer.println("   <state>" + cust.getState() + "</state>");
        writer.println("   <zip>" + cust.getZip() + "</zip>");
        writer.println("   <country>" + cust.getCountry() + "</country>");
        writer.println("</customer>");
        count++;
    }
}
```

CustomerResourceClient.java:

```
    // Get all the customers
    System.out.println("\n*** GET All the Customers ***");
    getUrl = new URL("http://localhost:8080/rest/customers");
    connection = (HttpURLConnection) getUrl.openConnection();
    connection.setRequestMethod("GET");
    System.out.println("Content-Type: " + connection.getContentType());

    reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));

    line = reader.readLine();
    while (line != null) {
        System.out.println(line);
        line = reader.readLine();
    }
    System.out.println(HttpURLConnection.HTTP_OK);
    System.out.println(connection.getResponseCode());
    connection.disconnect();
```

Results from console:



It shows all five customers' information.

3) Add another "GET" method to take two input arguments. It will return the specified customers based on a startingId and numOfCustomersToReturn. It will first get the list of all selected customers and then display them together.

CustomerResource.java:

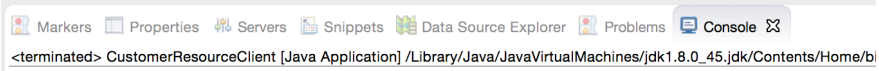
```
@GET
@Path("{startingId}/{numOfCustomersToReturn}")
@Produces("application/xml")
public StreamingOutput getSpecifiedCustomers(@PathParam("startingId") int startingId,
        @PathParam("numOfCustomersToReturn") int customersToReturn) {
    ArrayList <Customer> customers = new ArrayList<Customer>();
    for (int id = startingId; id < startingId + customersToReturn; id++) {
        Customer customer = customerDB.get(id);
        if (customer == null) {
            throw new WebApplicationException(Response.Status.NOT_FOUND);
        }
        customers.add(customer);
    }

    return new StreamingOutput() {
        public void write(OutputStream outputStream) throws IOException, WebApplicationException {
            outputSpecifiedCustomers(outputStream, customers);
        }
    };
}

protected void outputSpecifiedCustomers(OutputStream os, ArrayList<Customer> custs) throws IOException {
    PrintStream writer = new PrintStream(os);
    for (Customer cust : custs) {
        writer.println("<customer id=\"" + cust.getId() + "\">");
        writer.println("   <first-name>" + cust.getFirstName() + "</first-name>");
        writer.println("   <last-name>" + cust.getLastName() + "</last-name>");
        writer.println("   <street>" + cust.getStreet() + "</street>");
        writer.println("   <city>" + cust.getCity() + "</city>");
        writer.println("   <state>" + cust.getState() + "</state>");
        writer.println("   <zip>" + cust.getZip() + "</zip>");
        writer.println("   <country>" + cust.getCountry() + "</country>");
        writer.println("</customer>");
    }
}
```

CustomerResourceClient.java:

```
    // Get specified customers
    System.out.println("\n*** GET Specified Customers **");
    int startingId = 2;
    int numOfCustomersToReturn = 3;
    getUrl = new URL("http://localhost:8080/rest/customers/" + startingId + "/" + numOfCustomersToReturn);
    connection = (HttpURLConnection) getUrl.openConnection();
    connection.setRequestMethod("GET");
    System.out.println("Content-Type: " + connection.getContentType());

    reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));

    line = reader.readLine();
    while (line != null) {
        System.out.println(line);
        line = reader.readLine();
    }
    System.out.println(HttpURLConnection.HTTP_OK);
    System.out.println(connection.getResponseCode());
    connection.disconnect();
```

Note that the request should give two input parameters after “rest/customers”. The format should be corresponding to the method definition “{startingId}/{numOfCustomersToReturn}”.

Results from console:




```
*** GET Specified Customers **
Content-Type: application/xml
<customer id="2">
   <first-name>Tom</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="3">
   <first-name>Jason</first-name>
   <last-name>Burke</last-name>
   <street>248 Clarendon Street</street>
   <city>Cambridge</city>
   <state>MA</state>
   <zip>02478</zip>
   <country>USA</country>
</customer>
<customer id="4">
   <first-name>Richael</first-name>
   <last-name>Burke</last-name>
   <street>123 Clarendon Street</street>
   <city>Newton</city>
   <state>MA</state>
   <zip>01827</zip>
   <country>USA</country>
</customer>
200
200
```

4) Delete a customer based on the ID. Add a “DELETE” method/request. Here I return the response code without content. So the response code is “204”. We can also give a “200 OK” response. But we need to give a corresponding response content like “GET” request.

CustomerResource.java:

```
@DELETE
@Path("{id}")
public void deleteCustomer(@PathParam("id") int id) {
    final Customer customer = customerDB.get(id);
    if (customer == null) {
        throw new WebApplicationException(Response.Status.NOT_FOUND);
    }
    customerDB.remove(customer.getId(), customer);
    System.out.println("Deleted customer " + customer.getId());
}
```

CustomerResourceClient.java:

```
// Delete one particular customer
System.out.println("\n*** Delete One Particular Customer ***");
getUrl = new URL("http://localhost:8080/rest/customers/3");
connection = (HttpURLConnection) getUrl.openConnection();
connection.setRequestMethod("DELETE");
System.out.println("Content-Type: " + connection.getContentType());
System.out.println(HttpURLConnection.HTTP_NO_CONTENT);
System.out.println(connection.getResponseCode());
connection.disconnect();
```

Results from console:

```
*** Delete One Particular Customer ***
Content-Type: application/xml
204
204
```

I tried returning the response with content here. But problem 3 & 4 will still use return the response with no content.

CustomerResource.java:

```
@DELETE
@Path("{id}")
public StreamingOutput deleteCustomer(@PathParam("id") int id) {
    final Customer customer = customerDB.get(id);
    if (customer == null) {
        throw new WebApplicationException(Response.Status.NOT_FOUND);
    }
    customerDB.remove(customer.getId(), customer);
    System.out.println("Deleted customer " + customer.getId());

    return new StreamingOutput() {
        public void write(OutputStream outputStream) throws IOException, WebApplicationException {
            outputCustomer(outputStream, customer);
        }
    };
}
```

CustomerResourceClient.java:

```
// Delete one particular customer
System.out.println("\n*** Delete One Particular Customer ***");
getUrl = new URL("http://localhost:8080/rest/customers/3");
connection = (HttpURLConnection) getUrl.openConnection();
connection.setRequestMethod("DELETE");
System.out.println("Content-Type: " + connection.getContentType());

reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));

line = reader.readLine();
while (line != null) {
    System.out.println(line);
    line = reader.readLine();
}

System.out.println(HttpURLConnection.HTTP_OK);
System.out.println(connection.getResponseCode());
connection.disconnect();
```

Results from console:

```
*** Delete One Particular Customer ***
Content-Type: text/html
<customer id="3">
   <first-name>Jason</first-name>
   <last-name>Burke</last-name>
   <street>248 Clarendon Street</street>
   <city>Cambridge</city>
   <state>MA</state>
   <zip>02478</zip>
   <country>USA</country>
</customer>
200
200
```

Show all the customers again. We can see that the customer with ID equals 3 is deleted.

```
// Get all the customers
System.out.println("\n*** GET All the Customers ***");
getUrl = new URL("http://localhost:8080/rest/customers");
connection = (HttpURLConnection) getUrl.openConnection();
connection.setRequestMethod("GET");
System.out.println("Content-Type: " + connection.getContentType());

reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));

line = reader.readLine();
while (line != null) {
    System.out.println(line);
    line = reader.readLine();
}
System.out.println(HttpURLConnection.HTTP_OK);
System.out.println(connection.getResponseCode());
connection.disconnect();
```

Markers Properties Servers Snippets Data Source Explorer Problems Console

```
<terminated> CustomerResourceClient [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java
*** Delete One Particular Customer ***
Content-Type: application/xml
204
204

*** GET All the Customers ***
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="2">
   <first-name>Tom</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="4">
   <first-name>Richael</first-name>
   <last-name>Burke</last-name>
   <street>123 Clarendon Street</street>
   <city>Newton</city>
   <state>MA</state>
   <zip>01827</zip>
   <country>USA</country>
</customer>
<customer id="5">
   <first-name>Jane</first-name>
   <last-name>Burke</last-name>
   <street>234 Clarendon Street</street>
   <city>Watertown</city>
   <state>MA</state>
   <zip>02837</zip>
   <country>USA</country>
```

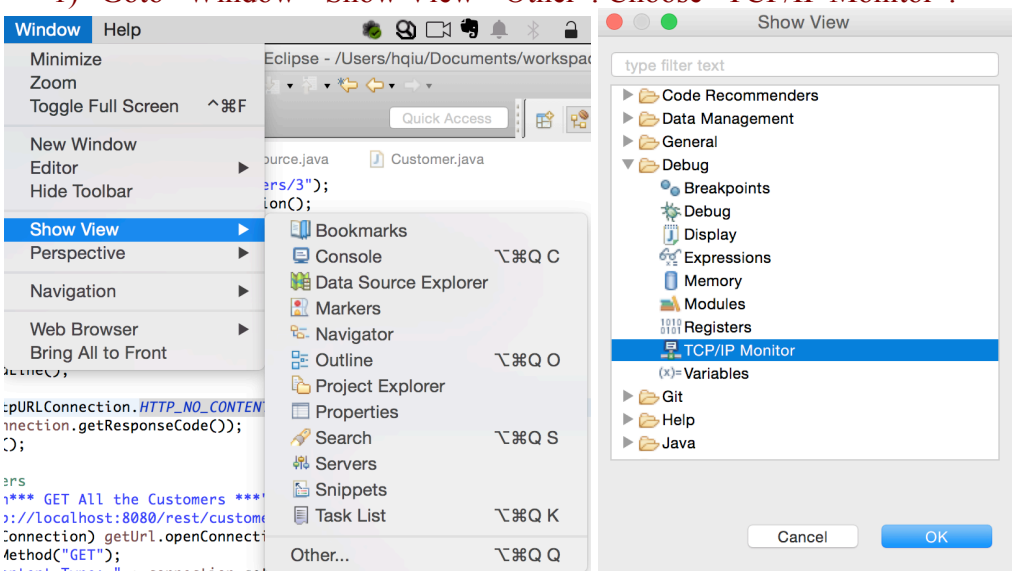
Problem 3:

Capture HTTP requests and responses for your `GET, PUT, POST` and `DELETE` method calls from Problem 2. Use TCPMon or some other HTTP monitoring tool to convince yourself that network traffic is indeed as advertised, meaning that you are sending HTTP messages with expected content and receiving HTTP messages with a similar content. Show your HTTP output `(GET, PUT, POST, DELETE)`.

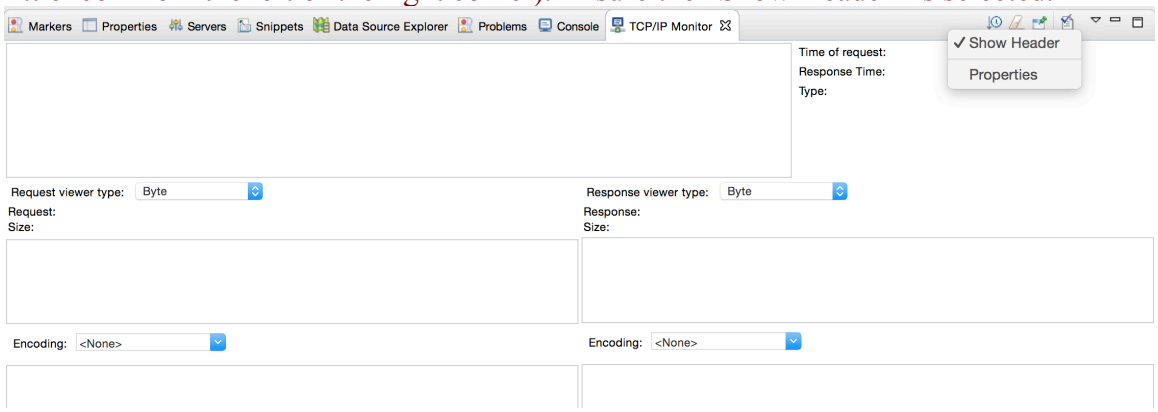
Points: 10

In this problem, I only changed file “CustomerResourceClient.java” and renamed it to “CustomerResourceClient_P3.java”. I use the Eclipse “TCP/IP Monitor” tool to monitor the traffic for this problem.

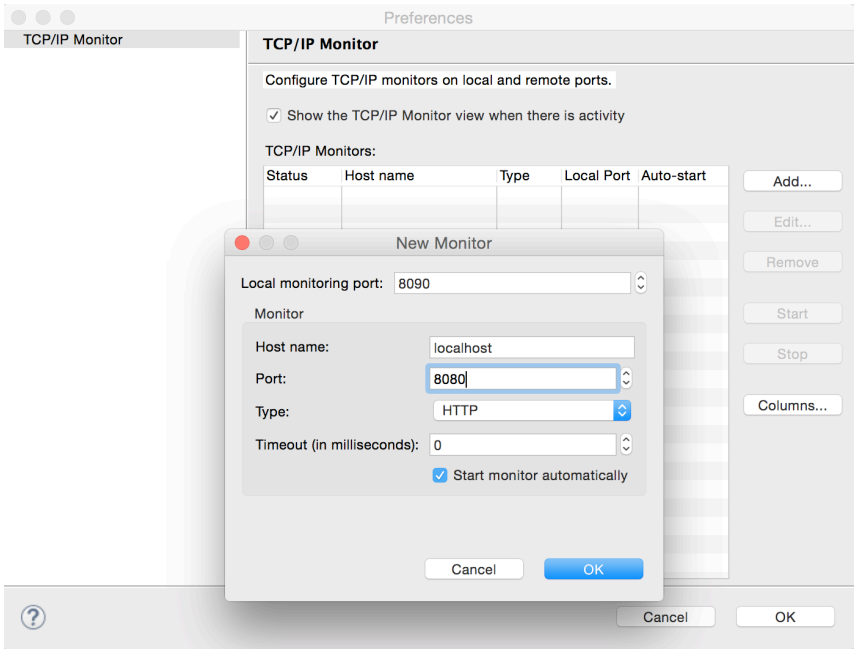
1) Goto “Window->Show View->Other”. Choose “TCP/IP Monitor”.

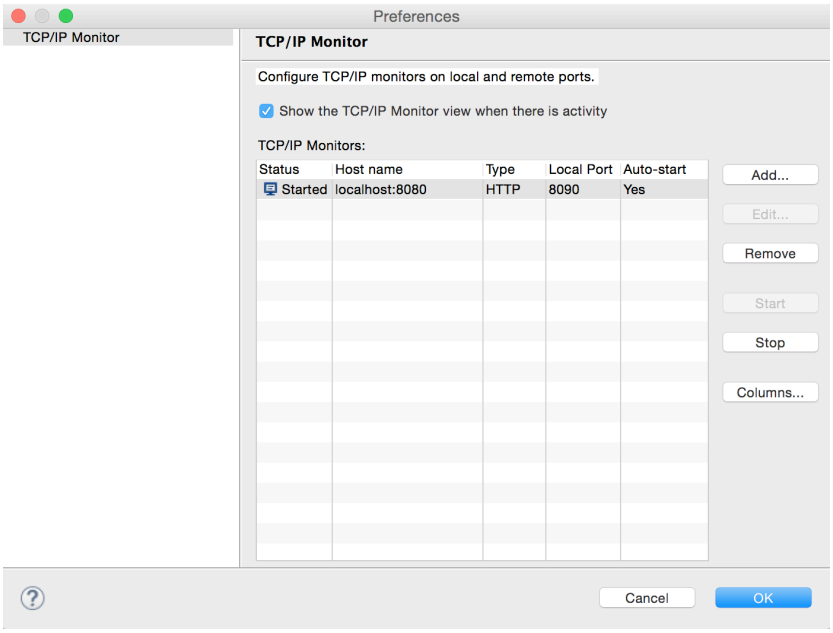


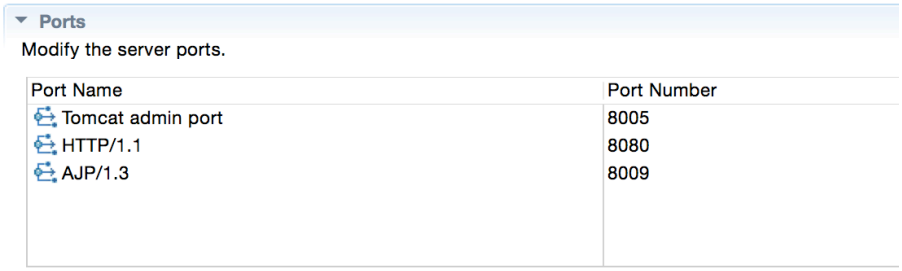
The monitoring window will appear at the bottom. Select upside down triangle (the third little icon from the left of the right corner). Ensure the “Show Header” is selected.



Next select the “Properties” from the same drop down. See below. New wizard will appear. The “local monitoring port” should be a port that was not currently used (8090 here). The port of the monitor is the Apache Tomcat port 8080.



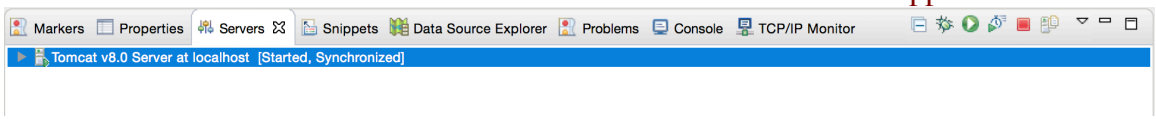




2) Edit the CustomerResourceClient.java to set the http to: 8090 instead of 8080 in all the http links.

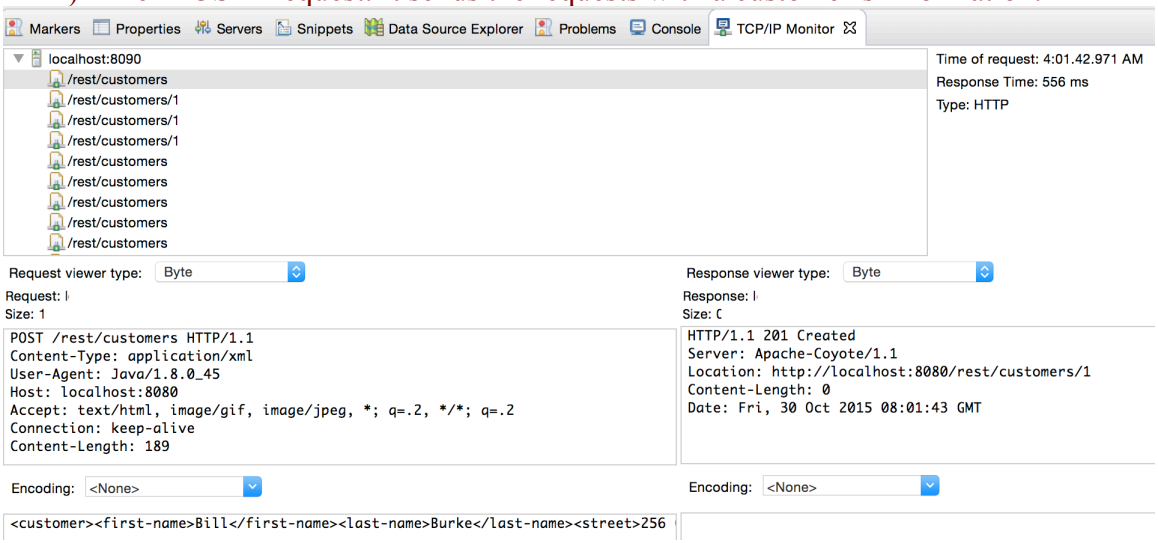
e.g., getUrl = new URL("http://localhost:8090/rest/customers/1");

3) Run the "rest" on server. Run the "CustomerResourceClient" as Java application.

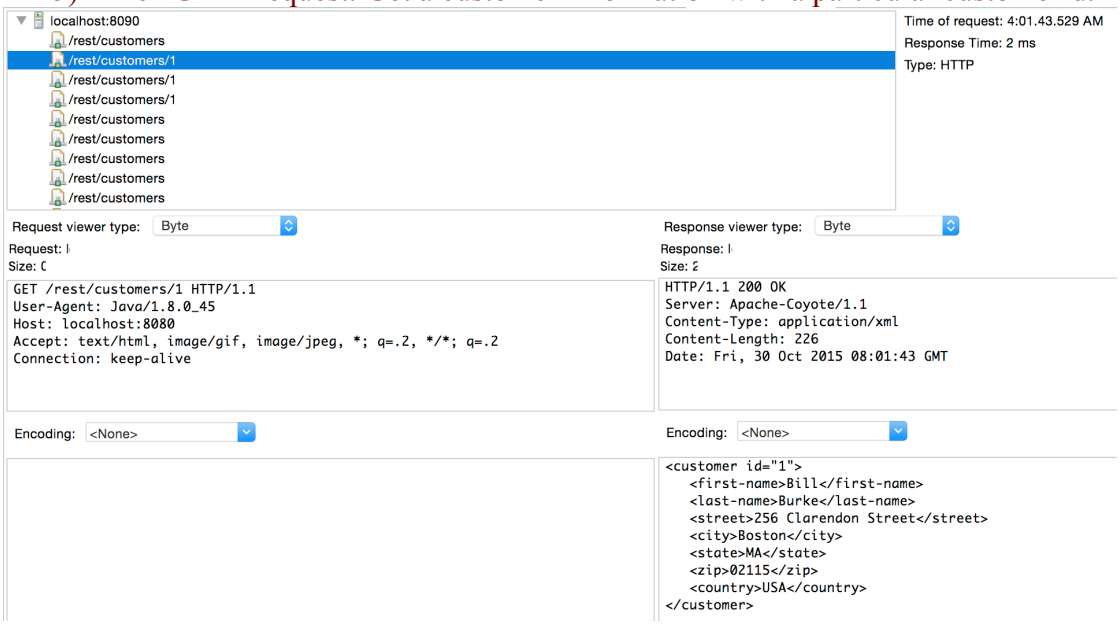


Click the green button with a triangle in the center to start/restart the server.

4) The "POST" request. It sends the requests with a customer's information.

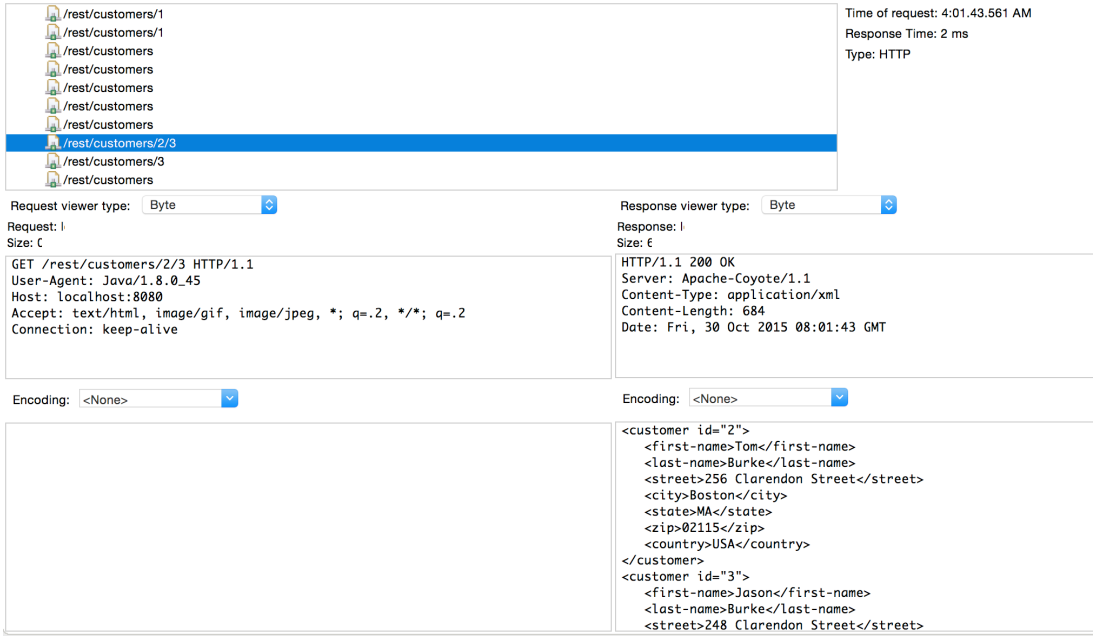


5) The "GET" request. Get a customer information with a particular customerId.

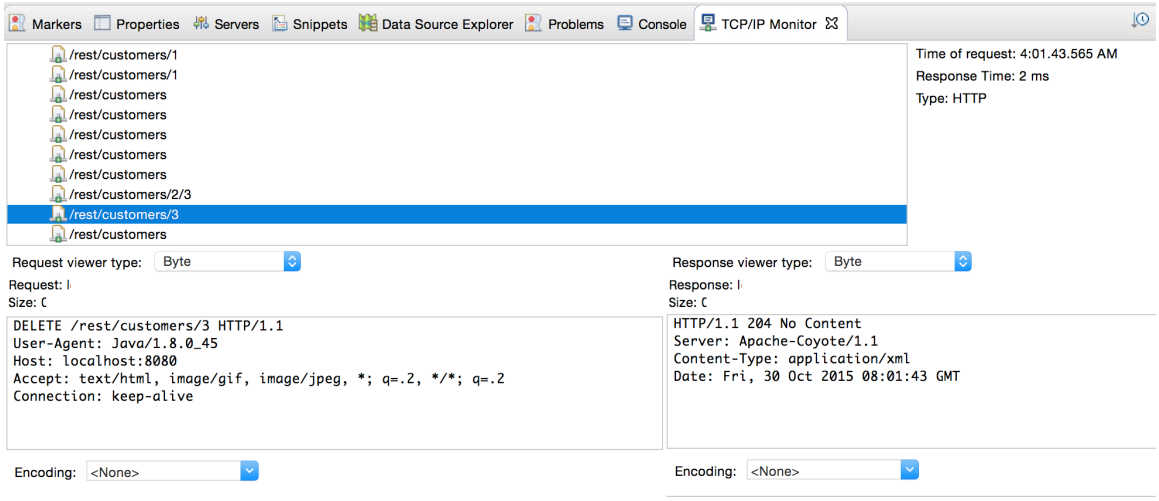


6) The "PUT" request. Update a customer's old information with a particular customerId and the customer's new information. The return response is "204" with no content.

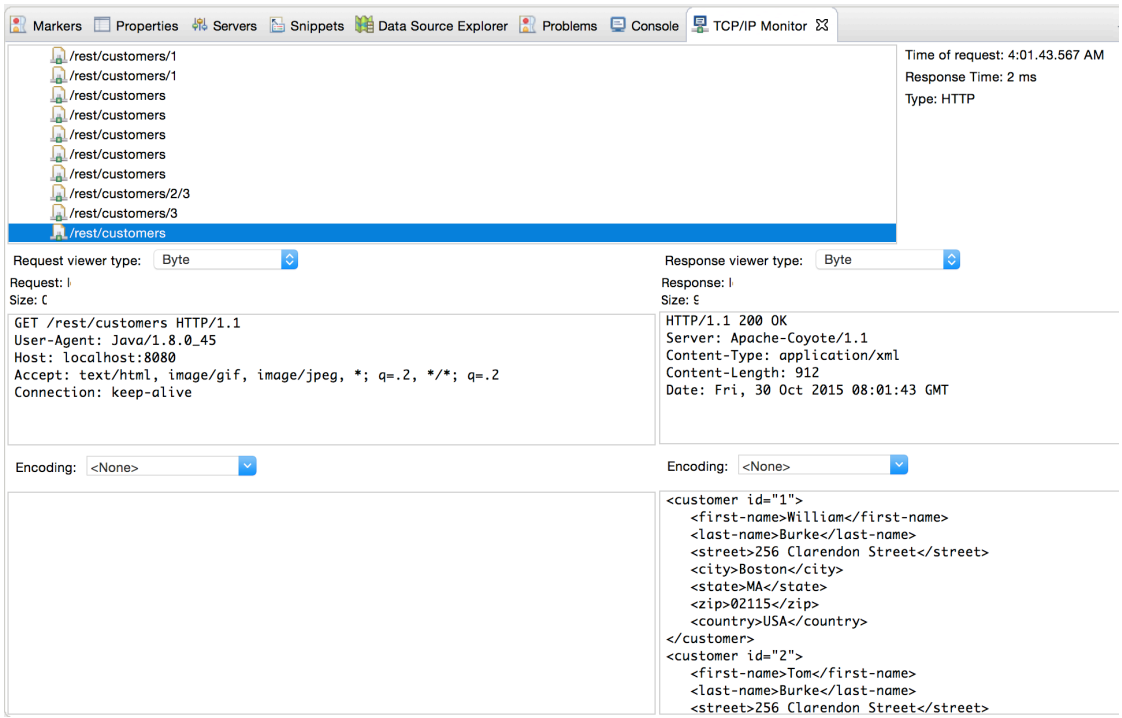
7) The "GET" request. Get a specified number of customers with a starting customerId and numOfCustomersToReturn. The response is "200 OK" with a list of customers' information.



8) The "DELETE" request. Delete a customer's information with customerId. The response is "204" with no content.



9) The "GET" request without input arguments. It will request information for all the customers back.

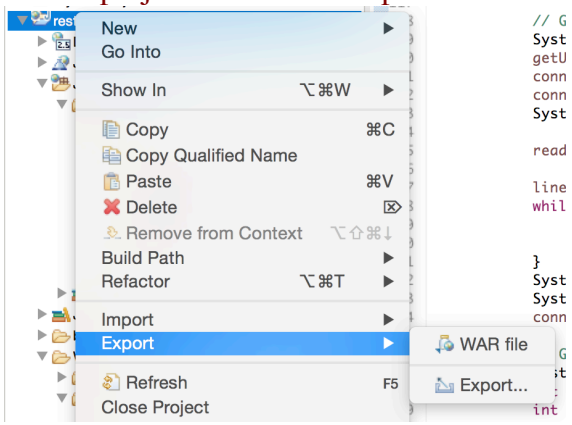


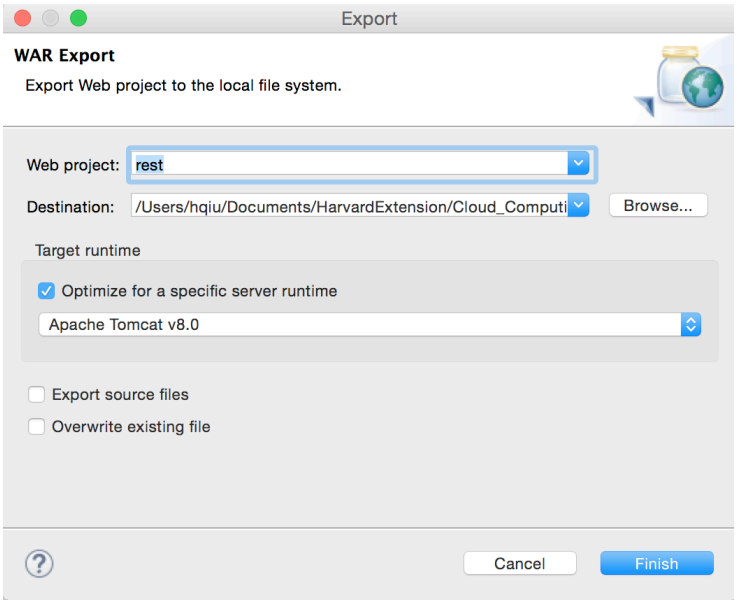
Problem 4:
Launch 2 AWS EC2 instances (Windows or Linux) in the same availability zone using Amazon Console or AWS CLI. Install Java JDK and Tomcat on one AWS Instance and startup Tomcat. Make sure you have security group set up (SSH, HTTP, etc.) and inbound rules. This is your Server and serves as your host of your RESTful web service. You can also use an AWS EC2 Instance that has Tomcat pre-installed (make sure Java JDK is installed and Tomcat is running. On your Server check that you can reach Tomcat in a browser: http://localhost:8080/). Assign a private IP address to your Server by creating an Elastic IP. You can do this in AWS Console or AWS CLI. Export your project (CustomerResource class) from problem 2 in Eclipse (as rest.war file) and copy it

to your Server's Tomcat \webapps directory. Make sure it has proper executable permissions. Restart your Tomcat server.
Launch a second AWS EC2 instance (Windows or Linux similar to your first EC2 instance with security group then install Java JDK.
On your laptop in Eclipse - add the private IP address of your Server to your CustomerResourceClient class for post, get, delete customer and recompile.
Copy your .jar folder to similar folder structure as you used in your package to your client (2nd AWS EC2 Instance). . Run your class: java … You should see customers being created, queried, … Show your output. Submit the code you have modified.
Points: 30

In this problem, I only need to change the CustomerResourceClient.java to set the http to the server private IP address instead of localhost in all the http links. e.g., getUrl = new URL("http://172.31.60.219:8080/rest/customers/1"); The java file for this problem is CustomerResourceClient_P4.java.

1) Export the project (CustomerResource class) to "rest.war" file. Right click "rest" project and select "Export->WAR file".



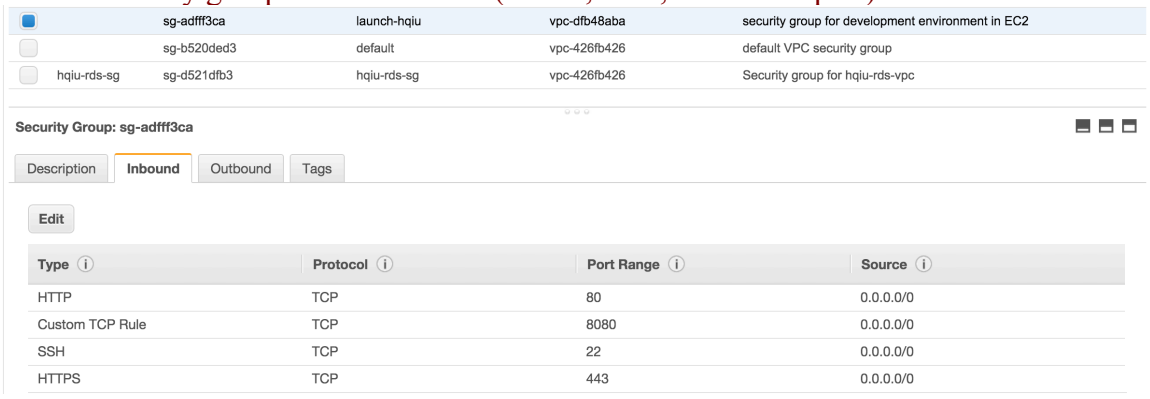


We can also do it through the command line. It's the same.

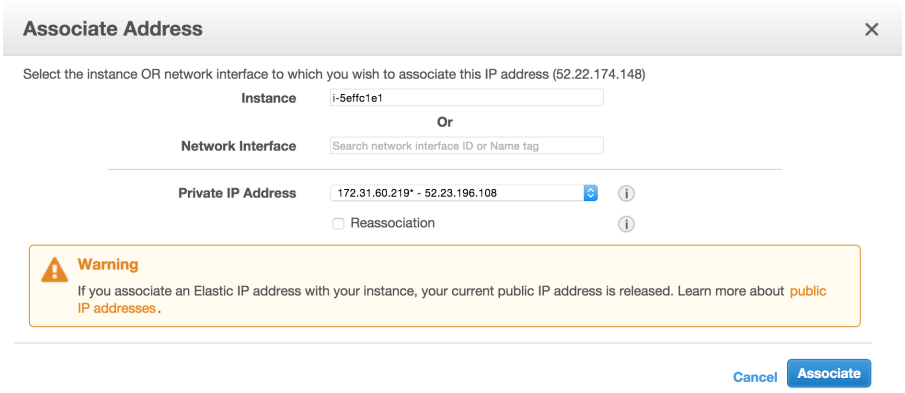
```
hqiu@bos-mpdei>> cd rest/
hqiu@bos-mpdei>> ls
WebContent      build           src
hqiu@bos-mpdei>> jar -cvf rest.war *
added manifest
adding: WebContent/(in = 0) (out= 0)(stored 0%)
adding: WebContent/META-INF/(in = 0) (out= 0)(stored 0%)
adding: WebContent/META-INF/MANIFEST.MF(in = 39) (out= 41)(deflated -5%)
adding: WebContent/WEB-INF/(in = 0) (out= 0)(stored 0%)
adding: WebContent/WEB-INF/lib/(in = 0) (out= 0)(stored 0%)
adding: WebContent/WEB-INF/lib/jersey-bundle-1.19.jar(in = 1623690) (out= 1415640)(deflated 12%)
adding: WebContent/WEB-INF/web.xml(in = 824) (out= 352)(deflated 57%)
adding: build/(in = 0) (out= 0)(stored 0%)
adding: build/classes/(in = 0) (out= 0)(stored 0%)
adding: build/classes/com/(in = 0) (out= 0)(stored 0%)
adding: build/classes/com/rest/(in = 0) (out= 0)(stored 0%)
adding: build/classes/com/rest/client/(in = 0) (out= 0)(stored 0%)
adding: build/classes/com/rest/client/CustomerResourceClient.class(in = 6101) (out= 2657)(deflated 56%)
adding: build/classes/com/rest/client/CustomerResourceClient_P1.class(in = 3454) (out= 1789)(deflated 48%)
adding: build/classes/com/rest/client/CustomerResourceClient_P3.class(in = 6110) (out= 2662)(deflated 56%)
adding: build/classes/com/rest/ws/(in = 0) (out= 0)(stored 0%)
adding: build/classes/com/rest/ws/.DS_Store(in = 6148) (out= 178)(deflated 97%)
adding: build/classes/com/rest/ws/Customer.class(in = 1850) (out= 663)(deflated 64%)
adding: build/classes/com/rest/ws/CustomerResource$1.class(in = 1072) (out= 575)(deflated 46%)
adding: build/classes/com/rest/ws/CustomerResource$2.class(in = 1010) (out= 522)(deflated 48%)
adding: build/classes/com/rest/ws/CustomerResource$3.class(in = 1029) (out= 544)(deflated 47%)
adding: build/classes/com/rest/ws/CustomerResource.class(in = 9630) (out= 4230)(deflated 56%)
adding: build/classes/com/rest/ws/ShoppingApplication.class(in = 949) (out= 494)(deflated 47%)
adding: src/(in = 0) (out= 0)(stored 0%)
adding: src/com/(in = 0) (out= 0)(stored 0%)
adding: src/com/rest/(in = 0) (out= 0)(stored 0%)
adding: src/com/rest/client/(in = 0) (out= 0)(stored 0%)
adding: src/com/rest/client/CustomerResourceClient.java(in = 8095) (out= 1249)(deflated 84%)
adding: src/com/rest/client/CustomerResourceClient_P1.java(in = 3489) (out= 874)(deflated 74%)
adding: src/com/rest/client/CustomerResourceClient_P3.java(in = 8098) (out= 1252)(deflated 84%)
adding: src/com/rest/ws/(in = 0) (out= 0)(stored 0%)
adding: src/com/rest/ws/.DS_Store(in = 6148) (out= 178)(deflated 97%)
adding: src/com/rest/ws/Customer.java(in = 1291) (out= 322)(deflated 75%)
adding: src/com/rest/ws/CustomerResource.java(in = 8005) (out= 1692)(deflated 78%)
adding: src/com/rest/ws/ShoppingApplication.java(in = 530) (out= 241)(deflated 54%)
```

2) Create two AWS EC2 Linux instances from the console. I choose the AMI with Tomcat v8.0 and JDK 1.8 installed.

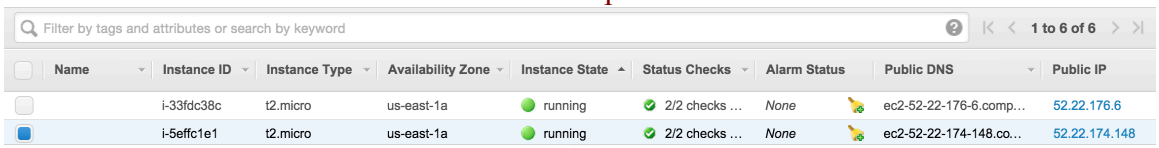
Set the security group for the instance (HTTP, SSH, TCP 8080 port).



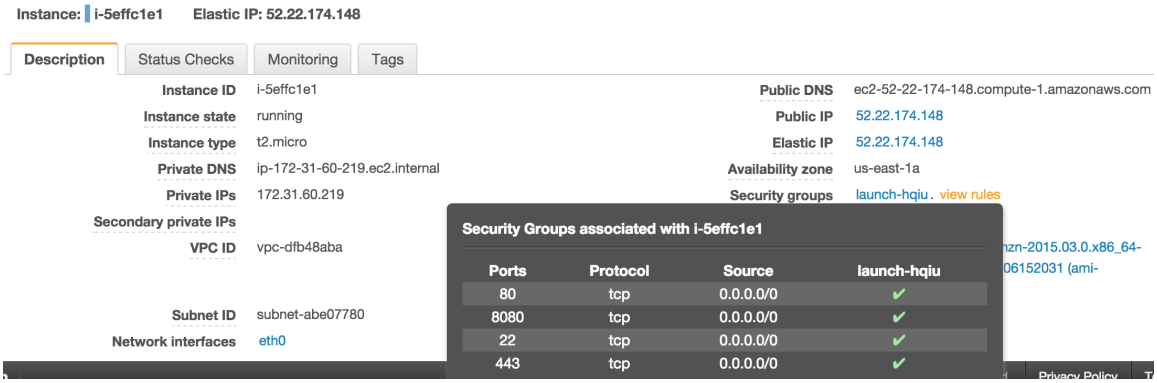
After initiating two instances, allocate two elastic IPs and associate the elastic IPs with the instances.



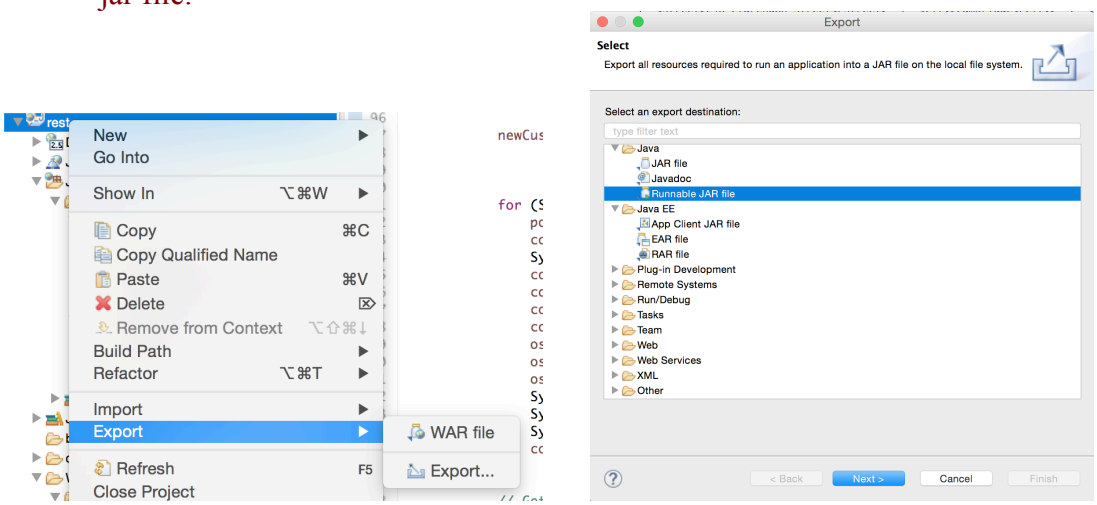
The information of the two instances and their public IPs.



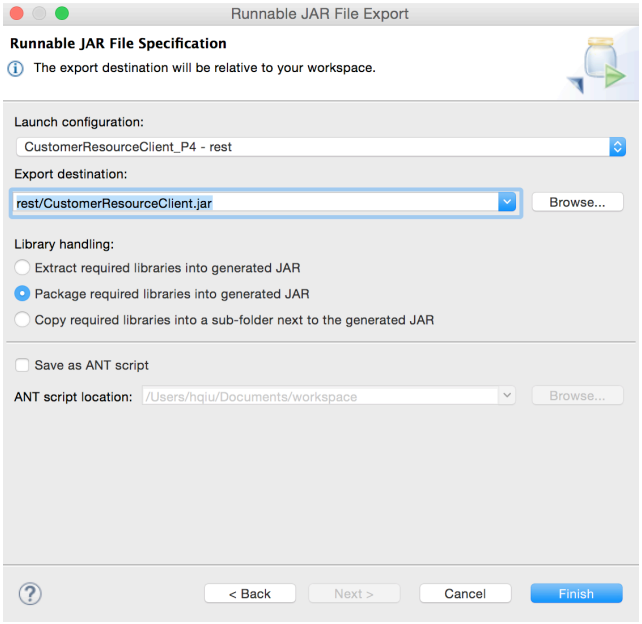
The private IP address of server "52.22.174.148" is "172.31.60.219".



3) Add the private IP address of the server to the "CustomerResourceClient". getUrl = new URL("http://172.31.60.219:8080/rest/customers/1"); Compile the runnable jar file.



Note to set the launch configuration file to “CustomerResourceClient_P4 – rest” to set it as the main method. Then we don’t need to change the Manifest file.



4) Start the Apache Tomcat service on the server.

```
hqiu@bos-mpdei>> ssh -i "ec2hqiu.pem" ec2-user@52.22.174.148
The authenticity of host '52.22.174.148 (52.22.174.148)' can't be established.
RSA key fingerprint is 65:a1:b2:88:62:33:1e:70:9d:dc:ed:ee:ed:ec:f6:16.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.22.174.148' (RSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2015.03-release-notes/
No packages needed for security; 115 packages available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2015.09 is available.
[ec2-user@ip-172-31-60-219 ~]$ java -version
openjdk version "1.8.0_45"
OpenJDK Runtime Environment (build 1.8.0_45-b13)
OpenJDK 64-Bit Server VM (build 25.45-b02, mixed mode)
```

The Tomcat is stopped as default. It also lacks the directory to write the log message.

```
[ec2-user@ip-172-31-60-219 tomcat8]$ sudo service tomcat8 status
tomcat8 is stopped
[ec2-user@ip-172-31-60-219 tomcat8]$ sudo service tomcat8 start
Starting tomcat8: touch: cannot touch ‘/usr/share/tomcat8/logs/tomcat8-initd.log’: No such file or directory
[ec2-user@ip-172-31-60-219 tomcat8]$                       [FAILED]
```

Create the directory and start the service.

```
[ec2-user@ip-172-31-60-219 share]$ sudo mkdir /var/log/tomcat8
[ec2-user@ip-172-31-60-219 share]$ sudo service tomcat8 start
[ec2-user@ip-172-31-60-219 share]$                         [  OK  ]
```

Even we’ve already started the service. Actually it’s still not running. We can check the port using “sudo fuser –v –n tcp 8080”. There’s no results. We need to install the

"tomcat8-webapps" to make it runnable.

```
[ec2-user@ip-172-31-60-219 ~]$ sudo service tomcat8 start
[ec2-user@ip-172-31-60-219 ~]$                             [  OK  ]
[ec2-user@ip-172-31-60-219 ~]$ sudo fuser -v -n tcp 8080
[ec2-user@ip-172-31-60-219 ~]$ sudo yum install tomcat8-webapps tomcat8-docs-webapp tomcat8-admin-webapps
Loaded plugins: priorities, update-motd, upgrade-helper
Resolving Dependencies
--> Running transaction check
---> Package tomcat8-admin-webapps.noarch 0:8.0.23-1.54.amzn1 will be installed
--> Processing Dependency: tomcat8 = 8.0.23-1.54.amzn1 for package: tomcat8-admin-webapps-8.0.23-1.54.amzn1.noarch
---> Package tomcat8-docs-webapp.noarch 0:8.0.23-1.54.amzn1 will be installed
---> Package tomcat8-webapps.noarch 0:8.0.23-1.54.amzn1 will be installed
--> Processing Dependency: jakarta-taglibs-standard >= 1.1 for package: tomcat8-webapps-8.0.23-1.54.amzn1.noarch
--> Running transaction check
---> Package jakarta-taglibs-standard.noarch 0:1.1.1-11.7.9.amzn1 will be installed
--> Processing Dependency: xalan-j2 >= 2.6.0 for package: jakarta-taglibs-standard-1.1.1-11.7.9.amzn1.noarch
--> Processing Dependency: apache-tomcat-apis for package: jakarta-taglibs-standard-1.1.1-11.7.9.amzn1.noarch
---> Package tomcat8.noarch 0:8.0.20-1.53.amzn1 will be updated
---> Package tomcat8.noarch 0:8.0.23-1.54.amzn1 will be an update
--> Processing Dependency: tomcat8-lib = 8.0.23-1.54.amzn1 for package: tomcat8-8.0.23-1.54.amzn1.noarch
--> Running transaction check
---> Package apache-tomcat-apis.noarch 0:0.1-1.6.amzn1 will be installed
---> Package tomcat8-lib.noarch 0:8.0.20-1.53.amzn1 will be updated
---> Package tomcat8-lib.noarch 0:8.0.23-1.54.amzn1 will be an update
```

After installing the "tomcat8-webapps" related packages, restart the Tomcat service. This time we can see that the Tomcat is running on port 8080.

```
[ec2-user@ip-172-31-60-219 ~]$ sudo service tomcat8 start
[ec2-user@ip-172-31-60-219 ~]$                             [  OK  ]
[ec2-user@ip-172-31-60-219 ~]$ sudo fuser -v -n tcp 8080
                     USER        PID ACCESS COMMAND
8080/tcp:            tomcat     2343 F.... java
```

5) Copy the "rest.war" file to the Ec2 server under "/var/lib/tomcat8/webapps" and the "CustomerResourceClient.jar" to the EC2 client under any directory. Change the directory permission for file transfer. Restart the Tomcat server service.

```
[ec2-user@ip-172-31-60-219 share]$ cd /var/lib/tomcat8/
[ec2-user@ip-172-31-60-219 tomcat8]$ ls -lart
total 12
drwxrwxr-x  2 root tomcat 4096 May 13 19:46 webapps
drwxr-xr-x  3 root tomcat 4096 Jun 15 20:35 .
drwxr-xr-x 18 root root   4096 Jun 15 20:35 ..
[ec2-user@ip-172-31-60-219 tomcat8]$ sudo chmod 777 webapps/
```

```
hqiu@bos-mpdei>> scp -i "ec2hqiu.pem" ~/Documents/workspace/rest/rest.war ec2-user@52.22.174.148:/var/lib/tomcat8/webapps
rest.war                                                                                    100% 1408KB   1.4MB/s   00:00
hqiu@bos-mpdei>> scp -i "ec2hqiu.pem" ~/Documents/workspace/rest/CustomerResourceClient.jar ec2-user@52.22.176.6:/home/ec2-user/restClient
CustomerResourceClient.jar                                                                  100% 8151KB   8.0MB/s   00:00
```

The restarted Tomcat server will take "rest.war" and generate a "rest" folder. The "Shopping Application" instance is running.

```
[ec2-user@ip-172-31-60-219 ~]$ sudo service tomcat8 restart
[ec2-user@ip-172-31-60-219 ~]$ _                           [  OK  ]
```

```
[ec2-user@ip-172-31-60-219 home]$ cd /var/lib/tomcat8/webapps/
[ec2-user@ip-172-31-60-219 webapps]$ ls
docs  examples  host-manager  manager  rest  rest.war  ROOT  sample
```

6) On the client side, directly run the jar file. It got the same results as run it locally.

Create a new customer using “POST”. “GET” the customer information and update it.

```
[ec2-user@ip-172-31-49-81 rest]$ java -jar CustomerResourceClient.jar

*** Create a new Customer ***
opened connection...
201
201
Location: http://172.31.60.219:8080/rest/customers/1

*** GET Created Customer ***
Content-Type: application/xml
<customer id="1">
   <first-name>Bill</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
204
204

*** GET Updated Customer ***
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
```

Create another four new customers.

```
*** Create four new Customers ***
opened connection...
201
201
Location: http://172.31.60.219:8080/rest/customers/2
opened connection...
201
201
Location: http://172.31.60.219:8080/rest/customers/3
opened connection...
201
201
Location: http://172.31.60.219:8080/rest/customers/4
opened connection...
201
201
Location: http://172.31.60.219:8080/rest/customers/5
```

Show all the customers after the "POST" requests.

```
*** GET All the Customers ***
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="2">
   <first-name>Tom</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="3">
   <first-name>Jason</first-name>
   <last-name>Burke</last-name>
   <street>248 Clarendon Street</street>
   <city>Cambridge</city>
   <state>MA</state>
```

```
<customer id="4">
   <first-name>Richael</first-name>
   <last-name>Burke</last-name>
   <street>123 Clarendon Street</street>
   <city>Newton</city>
   <state>MA</state>
   <zip>01827</zip>
   <country>USA</country>
</customer>
<customer id="5">
   <first-name>Jane</first-name>
   <last-name>Burke</last-name>
   <street>234 Clarendon Street</street>
   <city>Watertown</city>
   <state>MA</state>
   <zip>02837</zip>
   <country>USA</country>
</customer>
200
200
```

"GET" specified customers. Delete a particular customer and show the remaining customers' information.

```
*** GET Specified Customers **
Content-Type: application/xml
<customer id="2">
   <first-name>Tom</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="3">
   <first-name>Jason</first-name>
   <last-name>Burke</last-name>
   <street>248 Clarendon Street</street>
   <city>Cambridge</city>
   <state>MA</state>
   <zip>02478</zip>
   <country>USA</country>
</customer>
<customer id="4">
   <first-name>Richael</first-name>
   <last-name>Burke</last-name>
   <street>123 Clarendon Street</street>
   <city>Newton</city>
   <state>MA</state>
   <zip>01827</zip>
   <country>USA</country>
</customer>
200
200
```

```
*** Delete One Particular Customer ***
Content-Type: null
204
204

*** GET All the Customers ***
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="2">
   <first-name>Tom</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="4">
   <first-name>Richael</first-name>
   <last-name>Burke</last-name>
   <street>123 Clarendon Street</street>
   <city>Newton</city>
   <state>MA</state>
   <zip>01827</zip>
   <country>USA</country>
</customer>
<customer id="5">
   <first-name>Jane</first-name>
   <last-name>Burke</last-name>
   <street>234 Clarendon Street</street>
   <city>Watertown</city>
   <state>MA</state>
   <zip>02837</zip>
   <country>USA</country>
</customer>
```