CSC 615 TAKE HOME FINAL EXAM PROJECT

PROFESSOR GODFREY C. MUGANDA

Your task is to write a JAX-RS client to work with the JAX-RS `Candidates` service we have been using an example during the course.

First, you should download and install the Paraya Glassfish server from `www.paraya.fish`. The code you will need to write does not work if you use the regular Glassfish server due to bugs in the Oracle version of Glassfish.

The client you write will exchange information with the service using the JSON media type to represent objects of the class `Candidate`, as well as lists of Candidate objects.

Here is the `Candidate` class

```
@XmlRootElement
public class Candidate
{
    public String name;
    public int age;
    public String party;
    public Candidate()
    {

    }
    public Candidate(String name, int age, String party)
    {
        this.name = name;
        this.age = age;
        this.party = party;
    }
}
```

1. The Service Part

Use the following for your service class:

```
@Path("Pres")
public class PresResource
{

    List<Candidate> candidates = new ArrayList<>();
    public PresResource()
    {
       Candidate c = new Candidate();
       c.age = 34;
       c.party = "Libertarian";
       c.name = "James Q. Candid";
```

```
        candidates.add(c);
        c = new Candidate();
        c.age = 54;
        c.party = "Sectarian";
        c.name = "Joyce P. Goodenough";
        candidates.add(c);
    }

    /**
     * Retrieves representation of an instance of
     * Predential.PresResource
     * @return an instance of java.lang.String
     */
    @GET
    @Path("/all")
    @Produces(MediaType.APPLICATION_XML)
    public List<Candidate>  getAllXML()
    {
      return candidates;
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/all")
    public List<Candidate>  getAllJSON()
    {
      return candidates;
    }

    @GET
    @Produces(MediaType.APPLICATION_XML)
    @Path("{id}")
    public Candidate  getXML(@PathParam("id") int index)
    {
      return candidates.get(index);
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("{id}")
    public Candidate  getJSON(@PathParam("id") int index)
    {
      return candidates.get(index);
    }

    /**
     * PUT method for updating or creating an instance of PresResource
     * @param content representation for the resource
     */
    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    public void putJSON(Candidate c)
```

```
    {
        candidates.add(c);
    }

    @PUT
    @Consumes(MediaType.APPLICATION_XML)
    public void putXML(Candidate c)
    {
        candidates.add(c);
    }

    @PUT
    @Path("/all")
    @Consumes(MediaType.APPLICATION_XML)
    public void putAllXML(List<Candidate> c)
    {
        candidates.addAll(c);
    }

    @PUT
    @Path("/all")
    @Consumes(MediaType.APPLICATION_JSON)
    public void putAllJSON(List<Candidate> c)
    {
        candidates.addAll(c);
    }
}
```

Notice that the service can handle input and output in both XML and JSON.

The XML handling is built into Jersey: you will have to write a class that implements a message body reader and message body writer for JSON. Here is a shell

```
@Provider
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class JsonCandidateMarshaller implements
                    MessageBodyReader, MessageBodyWriter
{

}
```

Once you have everything in place you should be able to use the web service tester to test most of the web methods.

2. The Client Part

To create a client, start by creating a Java SE project, and then use Netbeans to add a RESTful client to the project.

Next, you need to add a copy of the `Candidate` class and a copy of the `JsonCandidateMarshaller` to the client you have created. You also have to register the JsonCandidateMarshaller class with the client: you can do that like this:

```
   client.register(JsonCandidateMarshaller.class);
```

Here is a part of the modified code:

```
public class PresCandidateClient
{

    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI
         = "http://localhost:28193/CandidateService/webresources";

    public PresCandidateClient()
    {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        client.register(JsonCandidateMarshaller.class);
        webTarget = client.target(BASE_URI).path("Pres");
    }
```

Use the following shell for your client

```
public class CandidatesClient
{
    public static void main(String[] args)
    {
        PresCandidateClient client = new PresCandidateClient();

        Candidate c = client.getXML(Candidate.class, "0");
        System.out.printf("%s, %s, %s\n", c.name, c.age, c.party);

        c = client.getJSON(Candidate.class, "1");
        System.out.printf("%s, %s, %s\n", c.name, c.age, c.party);

        Candidate c1 = new Candidate();
        c1.age = 34;
        c1.name = "Paul Young";
        c1.party = "Sesame Street Party";
        client.putXML(c1);

        c = client.getJSON(Candidate.class, "2");
        System.out.printf("%s, %s, %s\n", c.name, c.age, c.party);

        // Get the entire list in JSON format in the form of
        // a Response object,
        // Read the entity as an object, treat is as an InputStream
        // and print the contents of that stream

        // Get the entire list as a Response again,
        // treat it as an InputStream and
        // use  a JsonReader on the stream to read a JsonArray and
        // convert it to an array of Candidate and print  all
        // candidates.
        // ----------
```

```
    }
}
```

Notice that the code creates a client, and then invokes methods on it. The service starts out with a list of two candidates: and the client starts by retrieving the candidate at index 0 in XML and printing it, an then retrieving the candidate at index 1 and printing it. Next, the client creates a third candidate and adds it to the service by sending an XML representation of that candidate. Next, the client retrieves the candidate at index 2 and prints it.

Next, retrieve the entire list of candidates from the service in JSON format and print the JSON response to the console. Finally, retrieve the entire list of candidates in JSON format again, get the response stream as before, and then use a `JsonReader` to read the contents. Convert the contents to an array of `Candidate` and print the candidates.

To do this, you may want to modify the client code generated by Netbeans so that the methods for returning an entire list in JSON or XML return a `Response` object.

Here is a sample output

```
James Q. Candid, 34, Libertarian
Joyce P. Goodenough, 54, Sectarian
Paul Young, 34, Sesame Street Party

[{"name":"James Q. Candid","party":"Libertarian","age":34},
 {"name":"Joyce P. Goodenough","party":"Sectarian","age":54},
 {"name":"Paul Young","party":"Sesame Street Party","age":34}]

Here is the entire list:
James Q. Candid, 34, Libertarian
Joyce P. Goodenough, 54, Sectarian
Paul Young, 34, Sesame Street Party
```