Génie logiciel

Philippe Dugerdil

28.11.2019

Some more annotations

Annotations in the service method's parameter list

Pathname as service parameter

- Rather than forcing a subpath to match a string, one can get the value of the subpath in a parameter
- @Path("/{*urlVar*}")
  - The String parameter expression that matches the HTTP path can contain variables enclosed in braces. Here *unlVar* is the variable name that will be bound to a URL segment in a request.
  - Example:
    - If the class' @Path value is "/rooms" and method's @Path is "/{**numb**}",
    - The path **/rooms/123** would lead to the matching of **numb** with "123".

Inputting a URL segment value in a method

- @PathParam("myVar")
  - To be placed before the method's parameter in which to inject the value of the matched URI segment

- To get the value of a path variable in a methods we can use :
- `myMethod(@PathParam("myVar") String s)`

Type casting

Parameter Injection Type Casting

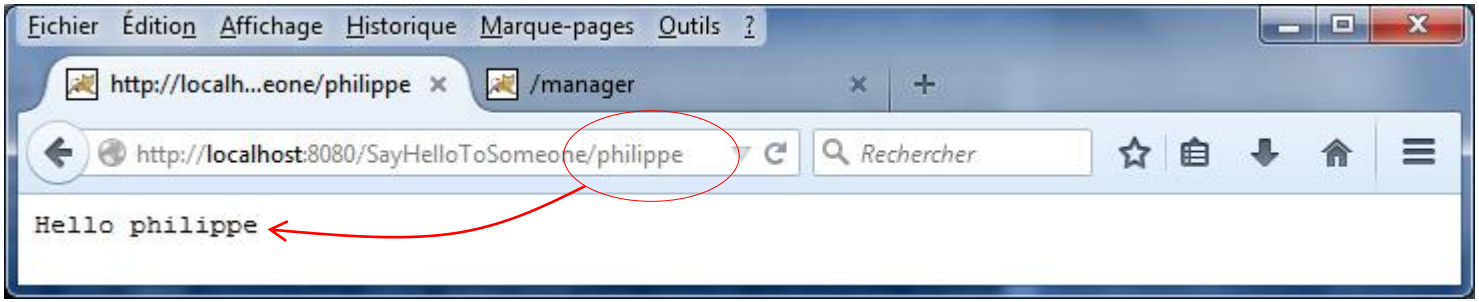
The value corresponding to the annotation is cast to the type of the associated method parameter if either:

1. It is a primitive type
2. It is a java class with a constructor having a single String parameter
3. It is a Java class that has a static method *valueOf(String)* which returns an instance of the class
4. It is a List<T>, Set<T>, SortedSet<T> where T satisfies the rule 2 or 3 above or is a String.

If a conversion fails the server will return a 4xx error

Example project: “SayHelloToSomeone”

```
import javax.ws.rs.*;
@Path("/")
public class SayHello {

@GET
@Path("{name}")
@Produces(MediaType.TEXT_PLAIN)
    public String sayHelloTXT(@PathParam("name") String name) {
        return "Hello " + name;
    }
}
```



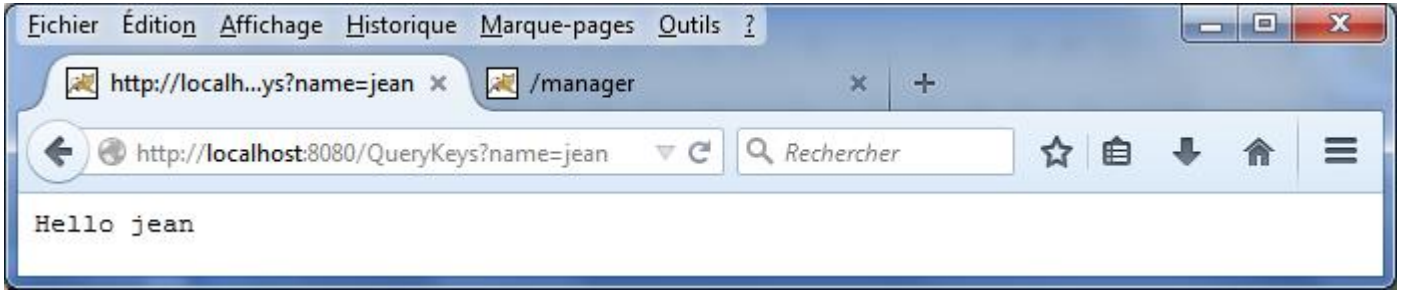
@QueryParam

- In HTTP queries, the query parameters in the URL are separated from the URL by "?". Each parameter is a pair *queryKey*=*value*. Parameters are separated by "&".
- @QueryParam("*queryKey*")
  - To be placed before the Java method's parameter that will get the value associated to *queryKey* in the URL.
  - With: …? start=10 we retrieve the value 10 using @QueryParam("start").
  - Default values: if the parameter is absent from the query, JAX RS injects **null** for object types and **0** for primitive types.

Example project: “QueryKeys”

```
import javax.ws.rs.*;

@Path("/")
public class KeyDisplay {

@GET
@Produces(MediaType.TEXT_PLAIN)
    public String sayHelloTXT(@QueryParam("name") String name) {
        return "Hello " + name;
    }
}
```

Type casting



@Context

- Injects HTTP context (environment parameters) information into a method's parameter.
- Depending on the type of the parameter prefixed by the @Context annotation, the information retrieved will be different.
  - This could be: UriInfo, HttpHeaders, HttpServletRequest, HttpServletResponse…

- Exemple: getting information about the URI: **UriInfo**

This type determine what will be injected

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public String getUriInfo(@Context UriInfo uriInfo) {  …}
```

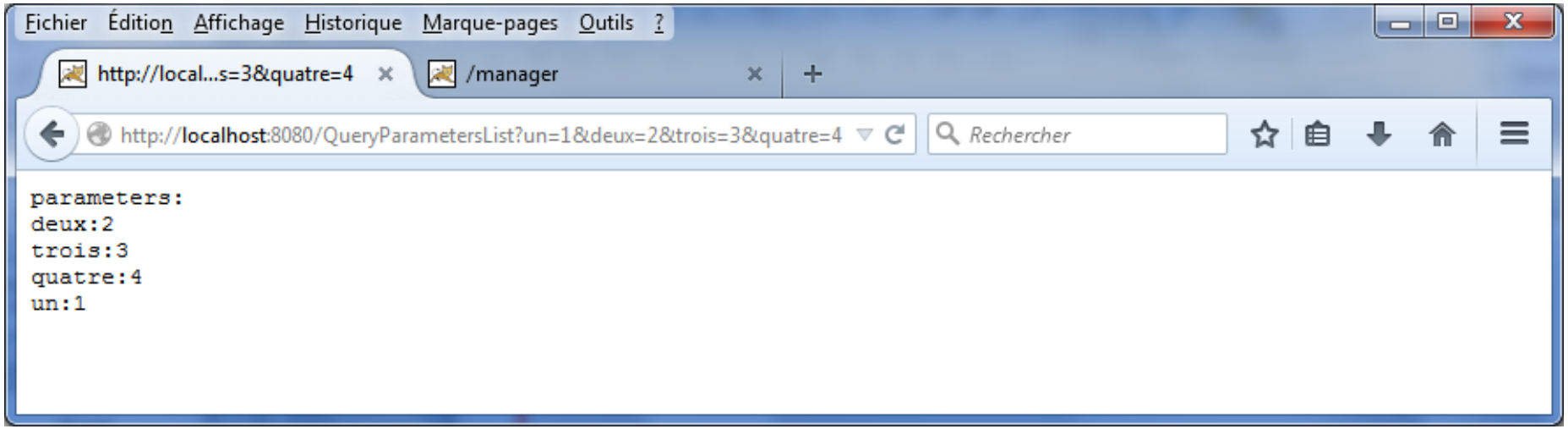
Type: UriInfo

Method Summary	
java.net.URI	getAbsolutePath() Get the absolute path of the request.
UriBuilder	getAbsolutePathBuilder() Get the absolute path of the request in the form of a UriBuilder.
java.net.URI	getBaseUri() Get the base URI of the application.
UriBuilder	getBaseUriBuilder() Get the base URI of the application in the form of a UriBuilder.
java.util.List<java.lang.Object>	getMatchedResources() Get a read-only list of the currently matched resource class instances.
java.util.List<java.lang.String>	getMatchedURIs() Get a read-only list of URIs for matched resources.
java.util.List<java.lang.String>	getMatchedURIs(boolean decode) Get a read-only list of URIs for matched resources.
java.lang.String	getPath() Get the path of the current request relative to the base URI as a string.
java.lang.String	getPath(boolean decode) Get the path of the current request relative to the base URI as a string.
MultivaluedMap<java.lang.String,java.lang.String>	getPathParameters() Get the values of any embedded URI template parameters.
MultivaluedMap<java.lang.String,java.lang.String>	getPathParameters(boolean decode) Get the values of any embedded URI template parameters.
java.util.List<PathSegment>	getPathSegments() Get the path of the current request relative to the base URI as a list of PathSegment.
java.util.List<PathSegment>	getPathSegments(boolean decode) Get the path of the current request relative to the base URI as a list of PathSegment.
MultivaluedMap<java.lang.String,java.lang.String>	getQueryParameters() Get the URI query parameters of the current request.
MultivaluedMap<java.lang.String,java.lang.String>	getQueryParameters(boolean decode) Get the URI query parameters of the current request.
java.net.URI	getRequestUri() Get the absolute request URI including any query parameters.
UriBuilder	getRequestUriBuilder() Get the absolute request URI in the form of a UriBuilder.

Project QueryParameterList

```
import java.util.Iterator;
import javax.ws.rs.*;
import javax.ws.rs.core.*;

@Path("/")
public class ParameterList {
@GET
@Produces(MediaType.TEXT_PLAIN)
   public String queryParamList(@Context UriInfo uriInfo) {
      MultivaluedMap<String,String> queryList = uriInfo.getQueryParameters();
      Iterator<String> it = queryList.keySet().iterator();
      StringBuffer sb = new StringBuffer();
      sb.append("parameters: \n");
      while(it.hasNext()){
         String key = it.next();
         sb.append(key).append(":").append(queryList.getFirst(key)).
                                                          append("\n");}
      return sb.toString();}}
```

Result



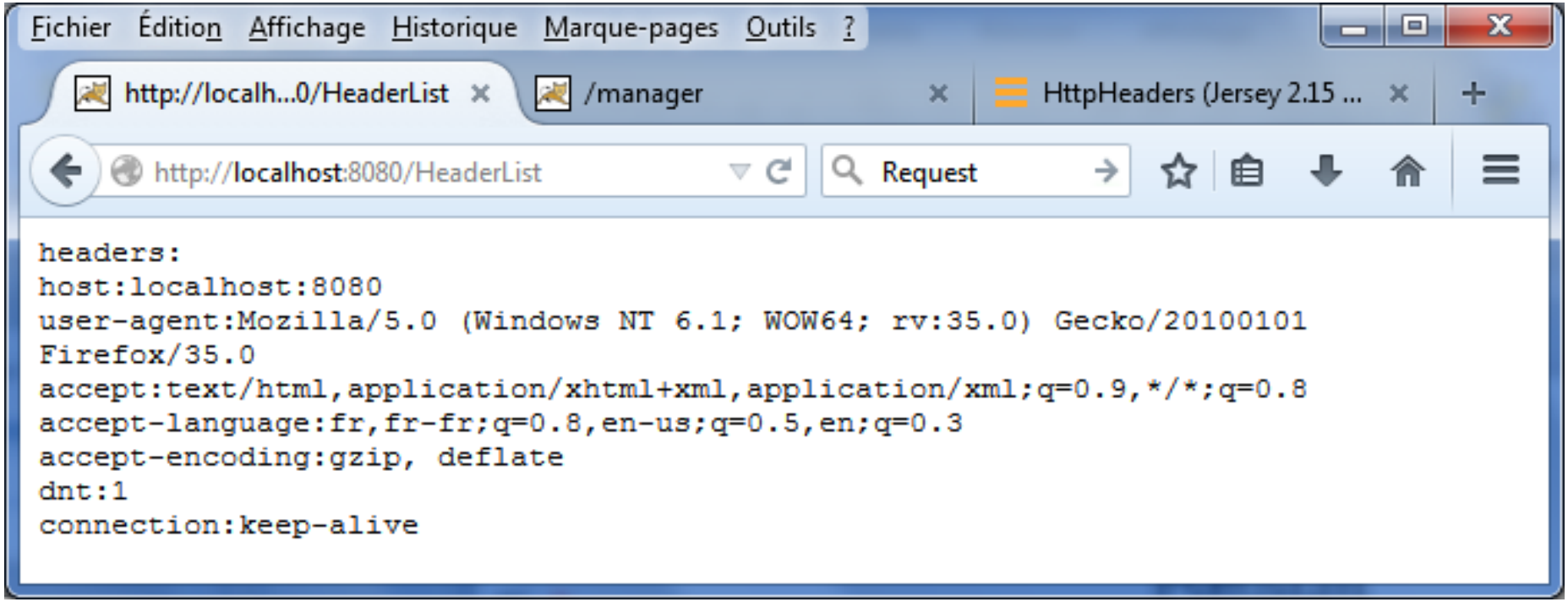
Type: HttpHeaders

Method Summary	
java.util.List<java.util.Locale>	getAcceptableLanguages() Get a list of languages that are acceptable for the response.
java.util.List<MediaType>	getAcceptableMediaTypes() Get a list of media types that are acceptable for the response.
java.util.Map<java.lang.String,Cookie>	getCookies() Get any cookies that accompanied the request.
java.util.Locale	getLanguage() Get the language of the request entity
MediaType	getMediaType() Get the media type of the request entity
java.util.List<java.lang.String>	getRequestHeader(java.lang.String name) Get the values of a HTTP request header.
MultivaluedMap<java.lang.String,java.lang.String>	getRequestHeaders() Get the values of HTTP request headers.

Project HeaderList

```
import java.util.Iterator;
import javax.ws.rs.*;
import javax.ws.rs.core.*;

@Path("/")
public class HeaderList {
@GET
@Produces(MediaType.TEXT_PLAIN)
   public String headerList(@Context HttpHeaders headers) {
      MultivaluedMap<String,String> headerList = headers.getRequestHeaders();
      Iterator<String> it = headerList.keySet().iterator();
      StringBuffer sb = new StringBuffer();
      sb.append("headers: \n");
      while(it.hasNext()){
         String key = it.next();
         sb.append(key).append(":").append(headerList.getFirst(key)).
                                                          append("\n");}
   return sb.toString();}}
```

Result



Type: HttpServletResponse

Method Summary	
void	addCookie(Cookie cookie) Adds the specified cookie to the response.
void	addDateHeader(java.lang.String name, long date) Adds a response header with the given name and date-value.
void	addHeader(java.lang.String name, java.lang.String value) Adds a response header with the given name and value.
void	addIntHeader(java.lang.String name, int value) Adds a response header with the given name and integer value.
boolean	containsHeader(java.lang.String name) Returns a boolean indicating whether the named response header has already been set.
java.lang.String	encodeRedirectURL(java.lang.String url) Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
java.lang.String	encodeURL(java.lang.String url) Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
java.lang.String	getHeader(java.lang.String name) Gets the value of the response header with the given name.
java.util.Collection<java.lang.String>	getHeaderNames() Gets the names of the headers of this response.
java.util.Collection<java.lang.String>	getHeaders(java.lang.String name) Gets the values of the response header with the given name.
int	getStatus() Gets the current status code of this response.
void	sendError(int sc) Sends an error response to the client using the specified status code and clears the buffer.
void	sendError(int sc, java.lang.String msg) Sends an error response to the client using the specified status and clears the buffer.
void	sendRedirect(java.lang.String location) Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer.
void	setDateHeader(java.lang.String name, long date) Sets a response header with the given name and date-value.
void	setHeader(java.lang.String name, java.lang.String value) Sets a response header with the given name and value.
void	setIntHeader(java.lang.String name, int value) Sets a response header with the given name and integer value.
void	setStatus(int sc) Sets the status code for this response.

Example: Setting HTTP status code

If the parameter prefixed by @Context is of type HttpServletResponse, on can set return parameters such as the return code or the response headers

```
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import javax.servlet.http.*;
@Path("/")
public class TestHeaders {
  @POST
  @Consumes(MediaType.TEXT_PLAIN)
  @Produces(MediaType.TEXT_PLAIN)
  public String addRoom(String input, @Context HttpServletResponse response){
    response.setStatus(HttpServletResponse.SC_CREATED);
    response.setHeader(HttpHeaders.LOCATION, "The new URL");
    try { response.flushBuffer(); }catch(Exception e){}
  return "hello";}
}
```

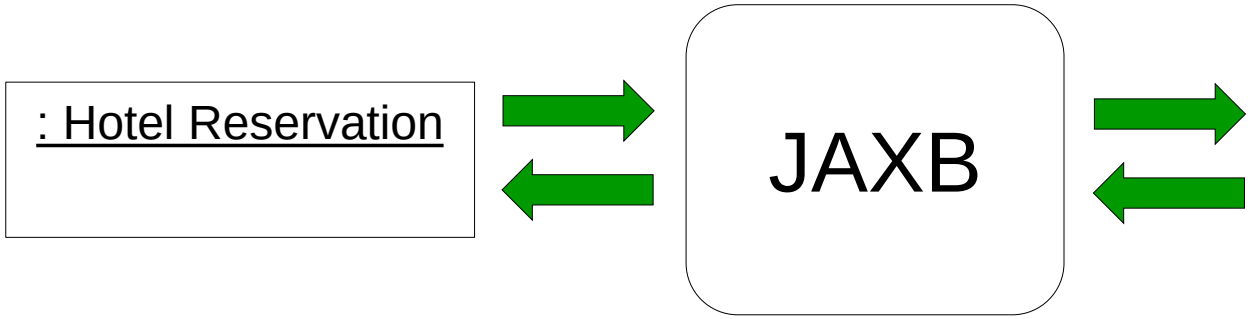
Status code: 201

Makes sure everything is outputted

Frequently payloads are coded either in XML or JSON

But writing the XML or JSON format “by hand” is tedious

Working with objects: JAXB

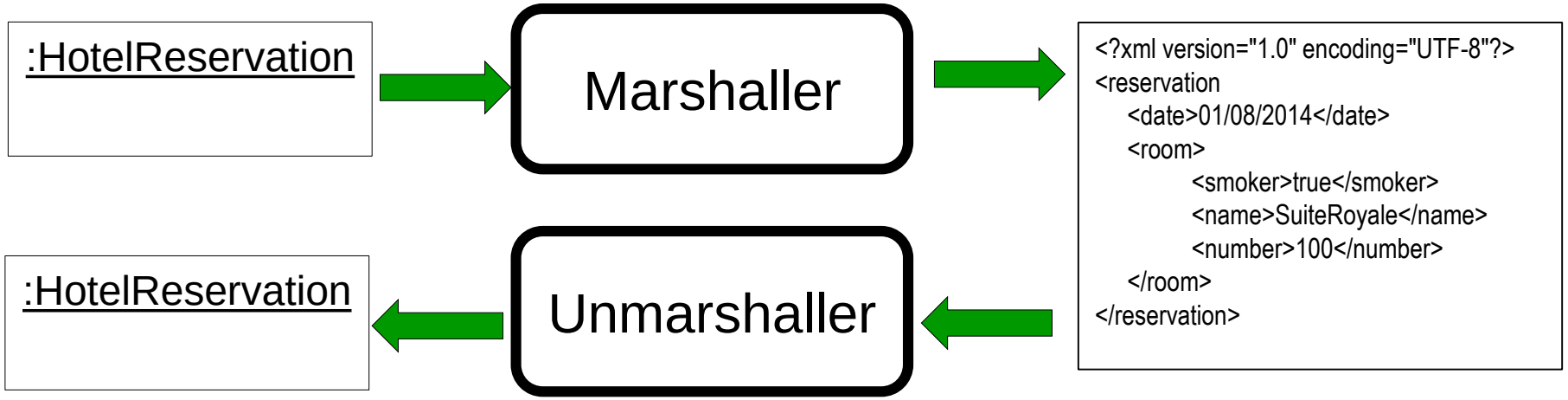



```
<?xml version="1.0" encoding="UTF-8"?>
<reservation
    <date>01/08/2014</date>
    <room>
        <smoker>true</smoker>

        <name>SuiteRoyale</name>
        <number>100</number>
    </room>
</reservation>
```

- JAXB is the standard library to transform objects from/to XML
  - javax.xml.bind.* (marshall / unmarshall)
- like JAX-RS, JAXB works with annotations
  - javax.xml.bind.annotation.*; (annotations in classes)
- JAXB is pre installed on Tomcat

Mashalling and Unmarshalling

The object to convert from/to XML must contain instructions to the Mashaller as JAXB annotations

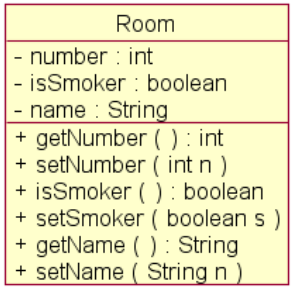


Annotation: @XmlRootElement

- @XmlRootElement
  - Location: before the class declaration.
    - ✓ The annotated class must have the empty constructor
  - This is the key annotation that tells JAXB that the class can be converted from/to XML
    - ✓ Only the root of a tree of objects needs it.
  - Default conversion:
    - ✓ Includes all the fields having public pairs of getter and setter as well as all the public fields as elements of the XML structure
  - If an attribute references an instance of some class, the latter will also be converted as subelement under the attribute elements

Example: simplest JAXB annotated class

```
import javax.xml.bind.annotation.*;

@XmlRootElement
public class Room {
  private int number;
  private boolean isSmoker;
  private String name;
  Room(){}
  Room(int number, boolean isSmoker, String name){
    this.isSmoker=isSmoker;
    this.number=number;
    this.name=name;}
  public int getNumber() {return number;}
  public void setNumber(int number) {this.number = number;}
  public boolean isSmoker() {return isSmoker;}
  public void setSmoker(boolean isSmoker) {this.isSmoker = isSmoker;}
  public String getName() {return name;}
  public void setName(String name) {this.name = name;}
}
```



JAX RS handlers for JAXB : output

JAXB annotated classes can become the return type of a method associated with some GET operation or the input parameter of a method associated with some PUT or POST operations

Example:

That's all we need to tell the server to marshall the annotated Room to XML

```
@GET
@Produces(MediaType.APPLICATION_XML)
public Room getRoom() {
        //Return an instance of Room
        //Room must be annotated with JAXB annotations
}
```

JAX RS handlers for JAXB : input

- POST Example

That's all we need to tell the server to unmarshall the XML to an annotated Room

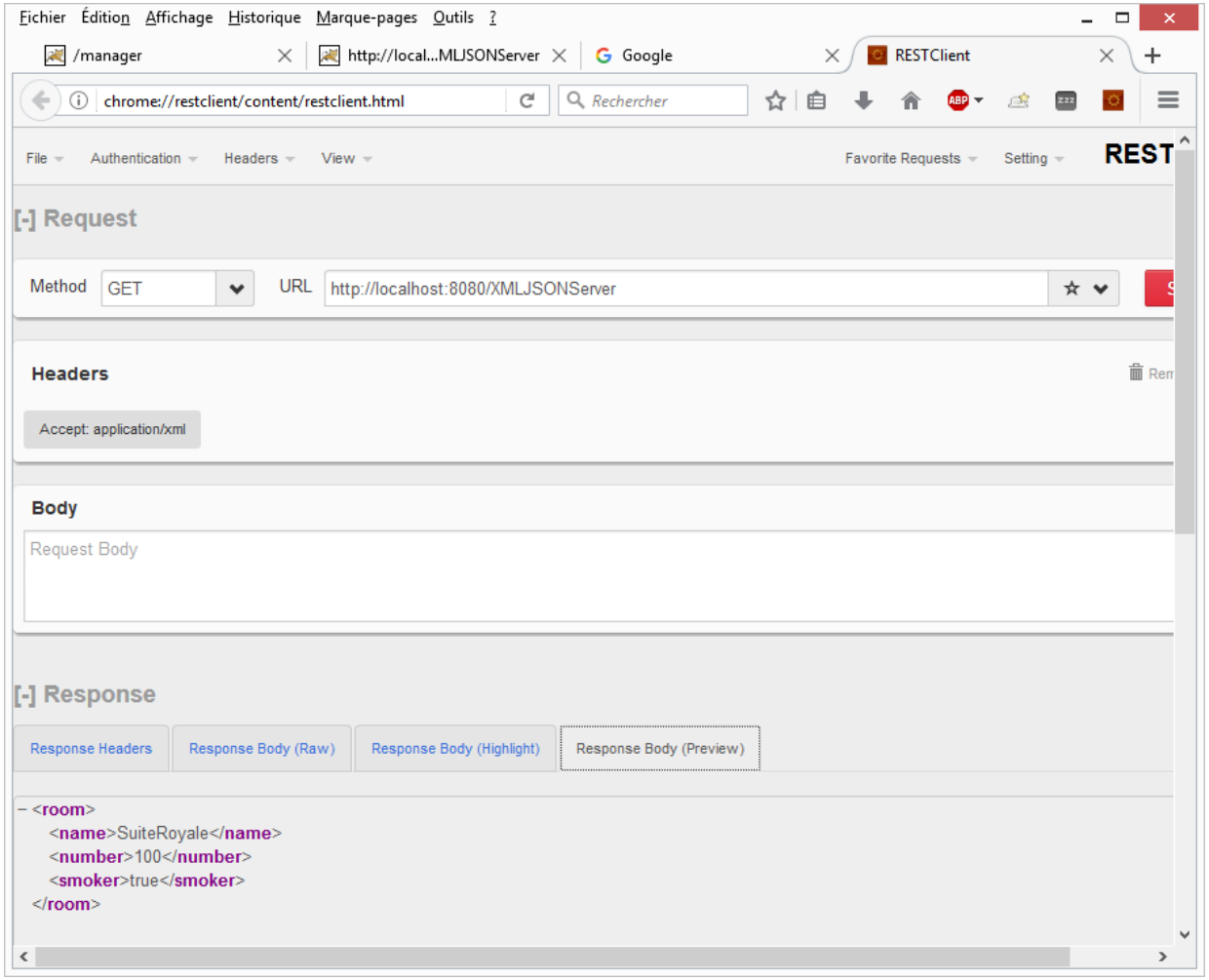
```
@POST
@Consumes(MediaType.APPLICATION_XML)
public void createRoom(Room room) {
    //the Room instance has been reconstructed from the XML payload
    //by JAX-RS, provided that Room is properly JAXB-annotated
}
```

Exemple

Service

```
import javax.ws.rs.*;
import javax.ws.rs.core.*;

@Path("/")
public class Service {

@GET
@Produces(MediaType.APPLICATION_JSON)
public Room returnObjectJSON() {
   return new Room(100,true,"SuiteRoyale");
   }

@GET
@Produces(MediaType.APPLICATION_XML)
public Room returnObjectXML() {
   return new Room(100,true,"SuiteRoyale");
   }}
```

Query XML



Query JSON

Fichier Édition Affichage Historique Marque-pages Outils ?

/manager | http://loca...JSONServer | Google | RESTClient

chrome://restclient/content/restclient.html

File Authentication Headers View Favorite Requests Setting

[-] Request

Method GET URL http://localhost:8080/XMLJSONServer

Headers

Accept: application/json

Body

Request Body

[-] Response

Response Headers | Response Body (Raw) | Response Body (Highlight) | Response Body (Preview)


```
{
 "name": "SuiteRoyale",
 "number": "100",
 "smoker": "true"
}
```

Summary: only 4 annotations

```
import javax.xml.bind.annotation.*;

@XmlRootElement
public class Room {
   private int number;
   private boolean isSmoker;
   private String name;
   Room(){}
   Room(int number, boolean isSmoker, String name){
      this.isSmoker=isSmoker;
      this.number=number;
      this.name=name;}
   public int getNumber() {return number;}
   public void setNumber(int number) {this.number = number;}
   public boolean isSmoker() {return isSmoker;}
   public void setSmoker(boolean isSmoker) {this.isSmoker = isSmoker;}
   public String getName() {return name;}
   public void setName(String name) {this.name = name;}
}
```

JAXB annotated class

```
mport javax.ws.rs.*;
mport javax.ws.rs.core.*;

Path("/")
ublic class Service {

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  public Room returnObject() {
  return new Room(100,true,"SuiteRoyale")}
```

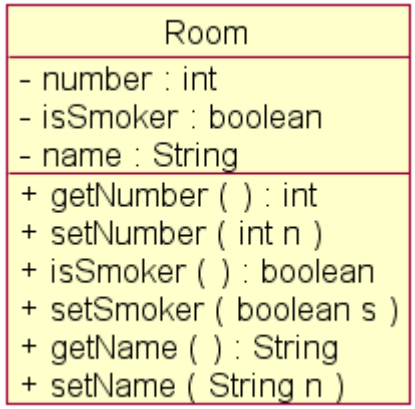
JAX RS annotated class

4 annotations (@XmlRootElement, @Path, @GET, @Produces) : that's all what is needed to create a GET service that returns the XML (JSON) representation of a simple object (Room).

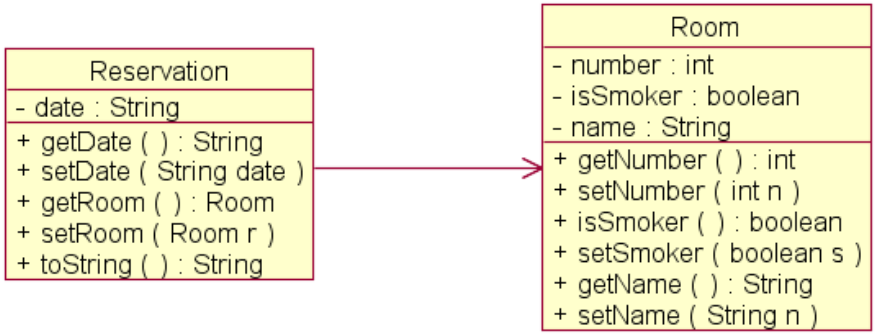
Explanation of the conversion

```
import javax.xml.bind.annotation.*;
@XmlRootElement
public class Room {
    private int number;
    private boolean isSmoker;
    private String name;
    Room(){}
    public int getNumber() {return number;}
    public void setNumber(int number) {this.number = number;}
    public boolean isSmoker() {return isSmoker;}
    public void setSmoker(boolean isSmoker) {this.isSmoker = isSmoker;}
    public String getName() {return name;}
    public void setName(String name) {this.name = name;}}
```

Included because of



Tree structure: Reservation class



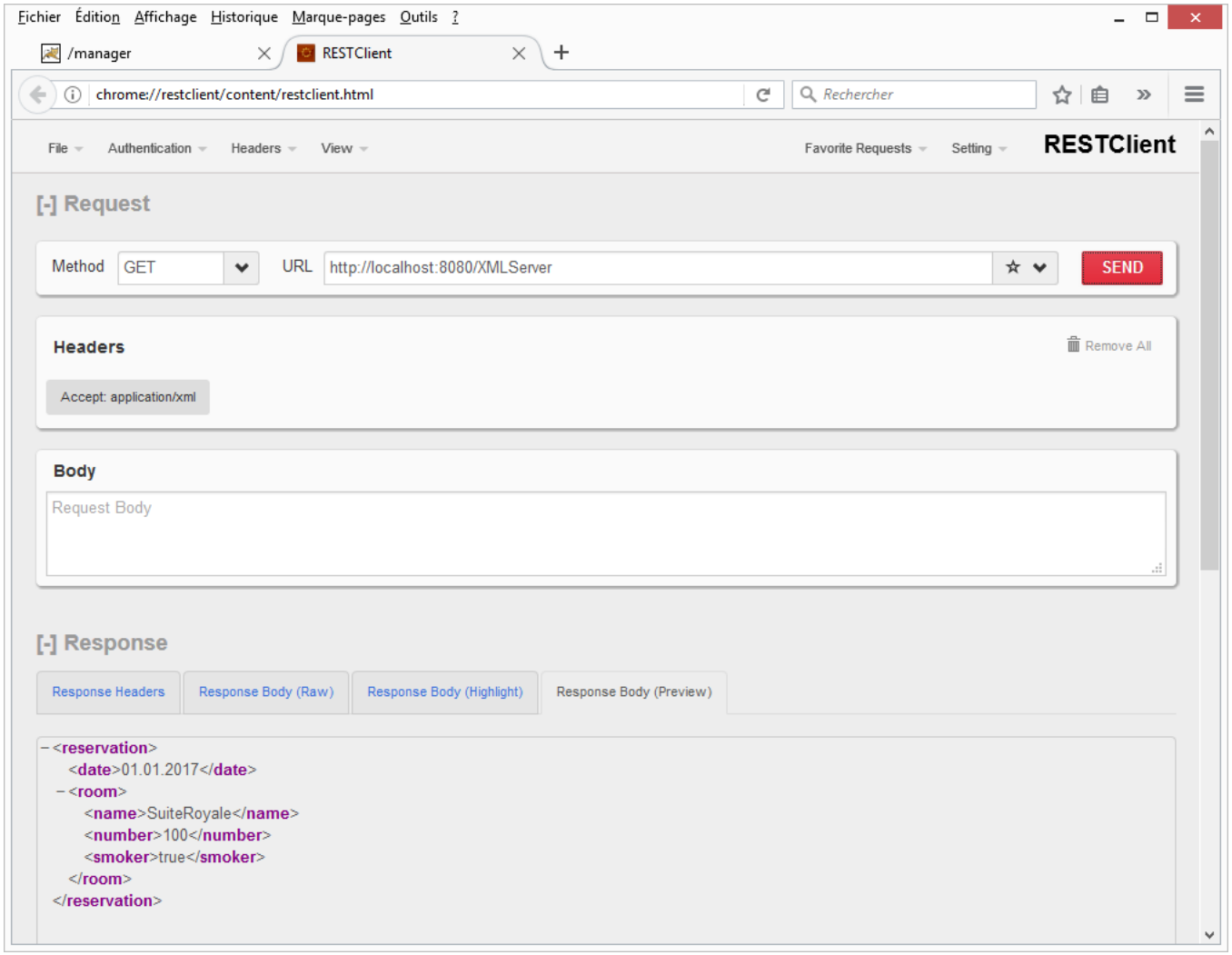

```
@XmlRootElement
public class Reservation {
    private Room room;
    private String date;
    Reservation() {}
    public Room getRoom() {return room;}
    public void setRoom(Room room) {this.room = room;}
    public String getDate() {return date;}
    public void setDate(String date) {this.date = date;}
    public String toString(){ return date + "\nroom : " + room;}}
```

NB: if Room is not going to be converted alone, it does not need the @XmlRootElement annotation

Service

```
import javax.ws.rs.*;
import javax.ws.rs.core.*;

@Path("/")
public class Service {

@GET
@Produces(MediaType.APPLICATION_JSON)
public Reservation returnObjectJSON() {
return new Reservation("01.01.2017", new Room(100,true,"SuiteRoyale"));
}

@GET
@Produces(MediaType.APPLICATION_XML)
public Reservation returnObjectXML() {
return new Reservation("01.01.2017", new Room(100,true,"SuiteRoyale"));
}}
```

Result: XML



Result: JSON

Fichier Édition Affichage Historique Marque-pages Outils ?

/manager | RESTClient

chrome://restclient/content/restclient.html

File Authentication Headers View Favorite Requests Setting **RESTClient**

[-] Request

Method GET URL http://localhost:8080/XMLServer SEND

Headers Remove All

Accept: application/json

Body

Request Body

[-] Response

Response Headers | Response Body (Raw) | Response Body (Highlight) | Response Body (Preview)


```
{
  "date": "01.01.2017",
  "room": {
    "name": "SuiteRoyale",
    "number": "100",
    "smoker": "true"
  }
}
```

Fine tuning the conversion

@XmlAccessorType(XmlAccessType.*keyword*)

- Tells what field to include in the XML structure.
- Location: before the class declaration.
- Keyword
  - **FIELD** : Every non static field unless annotated by **XmlTransient**.
  - **NONE** : None of the fields unless they are specifically annotated with some of the JAXB annotations.
  - **PROPERTY**: Every field having a getter/setter pair unless annotated by XmlTransient.
  - **PUBLIC_MEMBER**: Every public getter/setter pair and every public field unless annotated by XmlTransient.
    - ✓ This is de default conversion

Selecting the fields to include

- @XmlAccessorType(XmlAccessType.NONE) tells the system not to include any field but the ones identified with:
  - @XMLElement or @XMLElement(name = "*someName*")
    - ✓ Includes the field following the annotation as **an element** in the XML structure. The name parameter is optional. If not given, the name of the field is taken.
  - @XMLAttribute or @ XMLAttribute(name = "*someName*")
    - ✓ Include the field following the annotation as **an attribute of the root element** in the XML structure. The name parameter is optional. If not given, the name of the field is taken.

Use the *name* attribute to get an element name that is independent from the name of the field

Room

```java
@XmlRootElement
@XmlAccessorType(XmlAccessType.NONE)
public class Room {

    @XmlAttribute
    private int number;
    @XmlAttribute
    private boolean isSmoker;
    @XmlAttribute
    private String name;

    Room(){}
    Room(int number, boolean isSmoker, String name){
        this.isSmoker=isSmoker;
        this.number=number;
        this.name=name;
    }

    public int getNumber() {return number;}
    public void setNumber(int number) {this.number = number;}

    public boolean isSmoker() {return isSmoker;}
    public void setSmoker(boolean isSmoker) {this.isSmoker = isSmoker;}

    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
}
```

Result XML : attributes

Fichier Édition Affichage Historique Marque-pages Outils ?

/manager | RESTClient

chrome://restclient/content/restclient.html

File Authentication Headers View Favorite Requests Setting **RESTClient**

[-] Request

Method GET URL http://localhost:8080/XMLAccessorTypeServer/attribute SEND

Headers Remove All

Accept: application/xml

Body

Request Body

[-] Response

Response Headers | Response Body (Raw) | Response Body (Highlight) | Response Body (Preview)

```
<room number="100" isSmoker="true" name="SuiteRoyale"/>
```

PUT, POST & JAXB

The representation of the object to be created on the server must be sent in the payload of the request

Send: <room><name>SuiteRoyale</name><number>100</number><smoker>true</smoker></room>

But not:

```
<room>
  <name>SuiteRoyale</name>
  <number>100</number>
  <smoker>true</smoker>
</room>
```

The payload must not be formatted with CR/LF. Send the payload in the same format as what you receive from the GET service

Service

```
import javax.ws.rs.*;

@Path("/")
public class Service {

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void createObjectJSON(Room r) {
        System.out.println("JSON");
        System.out.println(r);
    }

    @POST
    @Consumes(MediaType.APPLICATION_XML)
    public void createObjectXML(Room r) {
        System.out.println("XML");
        System.out.println(r);
    }
}
```

The received Room object will be printed on the Tomcat console, after “XML” or “JSON” depending on the header

Room

```
@XmlRootElement
public class Room {

    private int number;
    private boolean isSmoker;
    private String name;
    Room(){}
    Room(int number, boolean isSmoker, String name){
        this.isSmoker=isSmoker;
        this.number=number;
        this.name=name;
    }

    public int getNumber() {return number;}
    public void setNumber(int number) {this.number = number;}

    public boolean isSmoker() {return isSmoker;}
    public void setSmoker(boolean isSmoker) {this.isSmoker = isSmoker;}

    public String getName() {return name;}
    public void setName(String name) {this.name = name;}

    public String toString(){
        return name +" "+number+" "+isSmoker;
    }
}
```

Result

Fichier Édition Affichage Historique Marque-pages Outils ?

/manager | RESTClient | Dugerdil Philippe (HES) - ...

chrome://restclient/content/restclient.html

File Authentication Headers View Favorite Requests Setting **RESTClient**

[-] Request

Method POST URL http://localhost:8080/XMLJSONPostServer SEND

Headers Remove All

Content-Type: application/XML

Body

```
<room><name>SuiteRoyale</name><number>100</number><smoker>true</smoker></room>
```

[-] Response

Response Headers | Response Body (Raw) | Response Body (Highlight) | Response Body (Preview)

```
1.  Status Code      : 204
2.  Date             : Thu, 30 Mar 2017 13:37:37 GMT
```

Tomcat's console

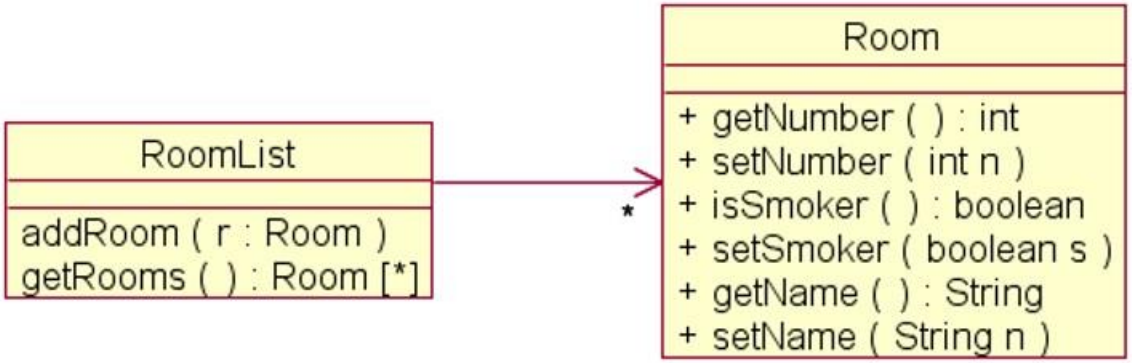
Tomcat

```
30-Mar-2017 15:23:53.105 INFO [http-nio-8080-exec-28] com.sun.jersey.api.core.Sc
anningResourceConfig.logClasses Root resource classes found:
  class classes.Service
30-Mar-2017 15:23:53.105 INFO [http-nio-8080-exec-28] com.sun.jersey.api.core.Sc
anningResourceConfig.init No provider classes found.
30-Mar-2017 15:23:53.107 INFO [http-nio-8080-exec-28] com.sun.jersey.server.impl
.application.WebApplicationImpl._initiate Initiating Jersey application, version
 'Jersey: 1.18 11/22/2013 03:05 AM'
30-Mar-2017 15:23:53.268 INFO [http-nio-8080-exec-28] org.apache.catalina.startu
p.HostConfig.deployWAR Deployment of web application archive C:\Program Files\ap
ache-tomcat-8.5.4\webapps\XMLJSONPostServer.war has finished in 403 ms
XML
SuiteRoyale 100 true
```

JAXB and Collections

JAXB can work with collections. The collection items will be placed one after the other in a substructure with name = name of the collection's field

```
import java.util.*;
import javax.xml.bind.annotation.*;
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class RoomList {
    private Collection<Room> rooms = new ArrayList<Room>();
    public void addRoom(Room r){rooms.add(r);}
    public Collection<Room> getRooms(){ return rooms;}
}
```


RoomList
addRoom (r : Room)
getRooms ( ) : Room [*]
*
Room
+ getNumber ( ) : int
+ setNumber (int n)
+ isSmoker ( ) : boolean
+ setSmoker (boolean s)
+ getName ( ) : String
+ setName (String n)

XML

Fichier Édition Affichage Historique Marque-pages Outils ?

/manager | RESTClient

chrome://restclient/content/restclient.html

File Authentication Headers View Favorite Requests Setting **RESTClient**

[-] Request

Method GET URL http://localhost:8080/XMLJSONCollection SEND

Headers Remove All

Accept: application/xml

Body

Request Body

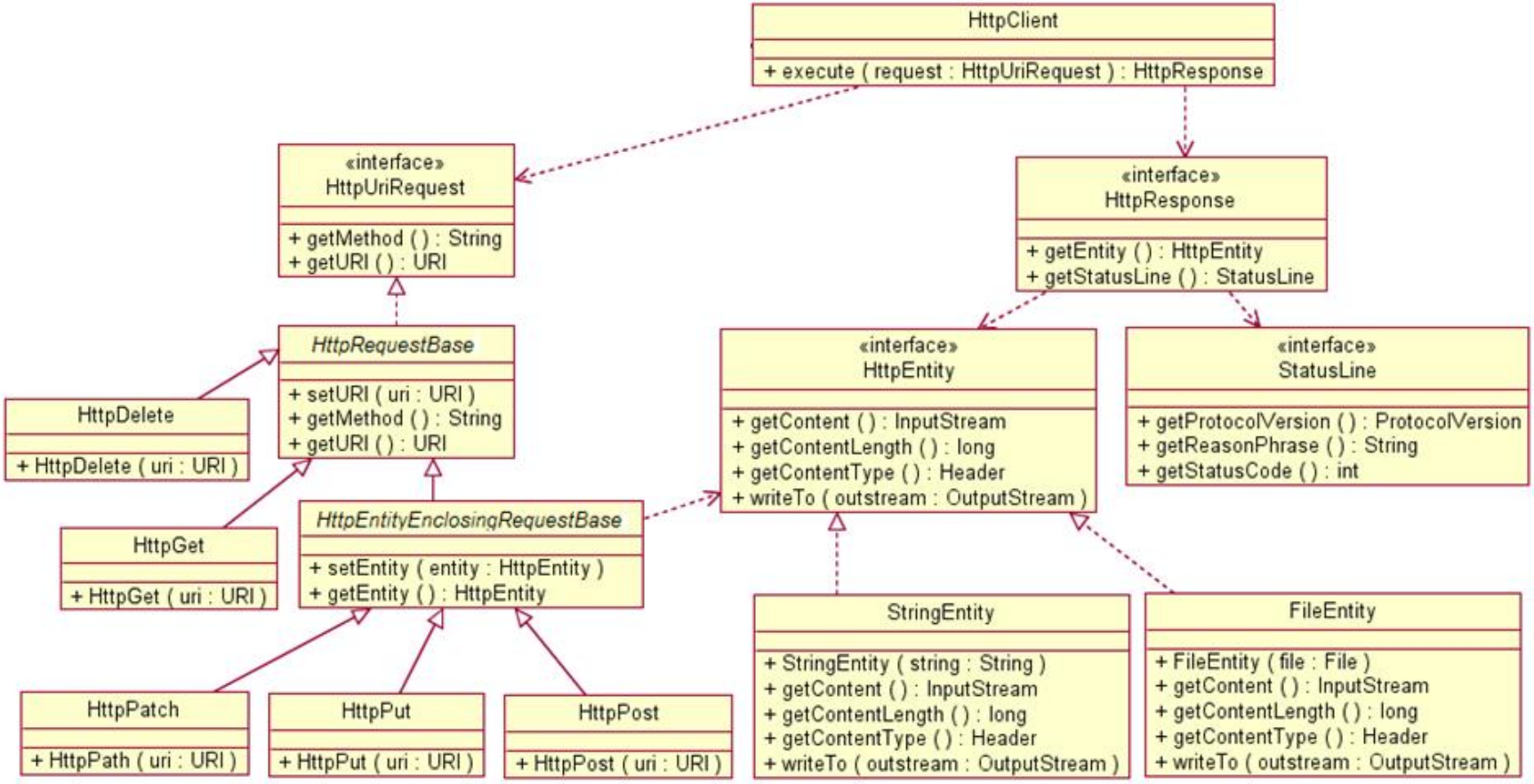
[-] Response

Response Headers | Response Body (Raw) | Response Body (Highlight) | Response Body (Preview)


```
- <roomList>
  - <rooms>
      <name>SuiteRoyale</name>
      <number>100</number>
      <smoker>true</smoker>
    </rooms>
  - <rooms>
      <name>Suite</name>
      <number>101</number>
      <smoker>false</smoker>
    </rooms>
  - <rooms>
      <name>Suite</name>
      <number>102</number>
      <smoker>false</smoker>
    </rooms>
  - <rooms>
      <name>normale</name>
      <number>103</number>
      <smoker>true</smoker>
    </rooms>
  </roomList>
```

Java as the REST client

Main HTTP Client classes



Steps to issue a query

1. The issuer of any query is an instance of HttpClient
   - Instantiated using a factory class : HttpClients.createDefault() or HttpClientBuilder.create().build();
2. Each kind of HTTP query is represented by its own class: HttpGet, HttpPut, HttpPost, HttpDelete
   - For HttpPut, HttpPost one must add the Payload which is an instance of HttpEntity
3. Then ask the HttpClient to **execute** the query
   - This returns an instance of HttpResponse

Processing the response to a query

1. The query response is an instance of HttpResponse
   - The code is obtained with getStatusLine().getStatusCode()
   - Type of payload is obtained with getEntity().getContentType()
     - ✓ Returns a Header (use getValue() to get the value of the header. This returns null if Content-Type is unknown)
   - The payload is obtained with: getEntity().getContents().
     - Returns an InputStream
2. Use the Stream objects and protocol to read the contents.

Required libraries

- Go to http://hc.apache.org
  - Dowload the HttpClient latest GA version
  - Unzip package. In our context we only need the following lib :
    - ✓ httpclient-x.y.jar
    - ✓ httpcore-x.y.jar
    - ✓ commons-logging-x.y.jar

    [x,y : version numbers]
- These lib must be added to the path of Java clients apps

Example: invoking HelloWorld

```
public class HttpClientHelloWorld {

public static void main(String[] args) throws Exception {
    String url = "http://localhost:8080/HelloWorld/helloworld";
    HttpClient client = HttpClients.createDefault();
    HttpGet request = new HttpGet(url);
    HttpResponse response = client.execute(request);
    System.out.println(response.getStatusLine().getStatusCode());
    BufferedReader rd = new BufferedReader(
        new InputStreamReader(response.getEntity().getContent()));
    String line;
    while ((line = rd.readLine()) != null)
        System.out.println(line);
    }
}
```

That's all we need To issue a GET query

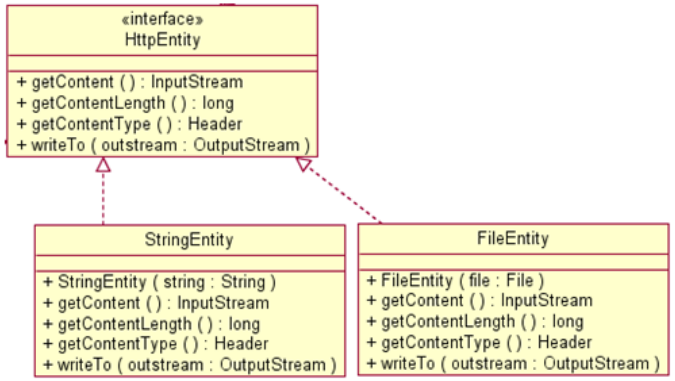
Console

```
200
Hello World
```

Setting the Payload on the client side

The payload is an HttpEntity

- Set the HttpPut payload using
  - setEntity(HttpEntity entity)
  - Classes implementing the HttpEntity protocol :
    - StringEntity
    - FileEntity



- Set the proper header to set the type of the payload
  - GET query should declare what it accepts, example:

```
request.setHeader(HttpHeaders.ACCEPT,MediaType.APPLICATION_XML);
```

  - PUT query should declare what it sends (and accepts), example:

```
request.setHeader(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN);
```

Simple example: Echo service

```
import javax.ws.rs.*;

@Path("/")
public class Echo {

   @PUT
   @Consumes(MediaType.TEXT_PLAIN)
   public void writeOnConsole(String input) {
       System.out.println(input);
   }
}
```

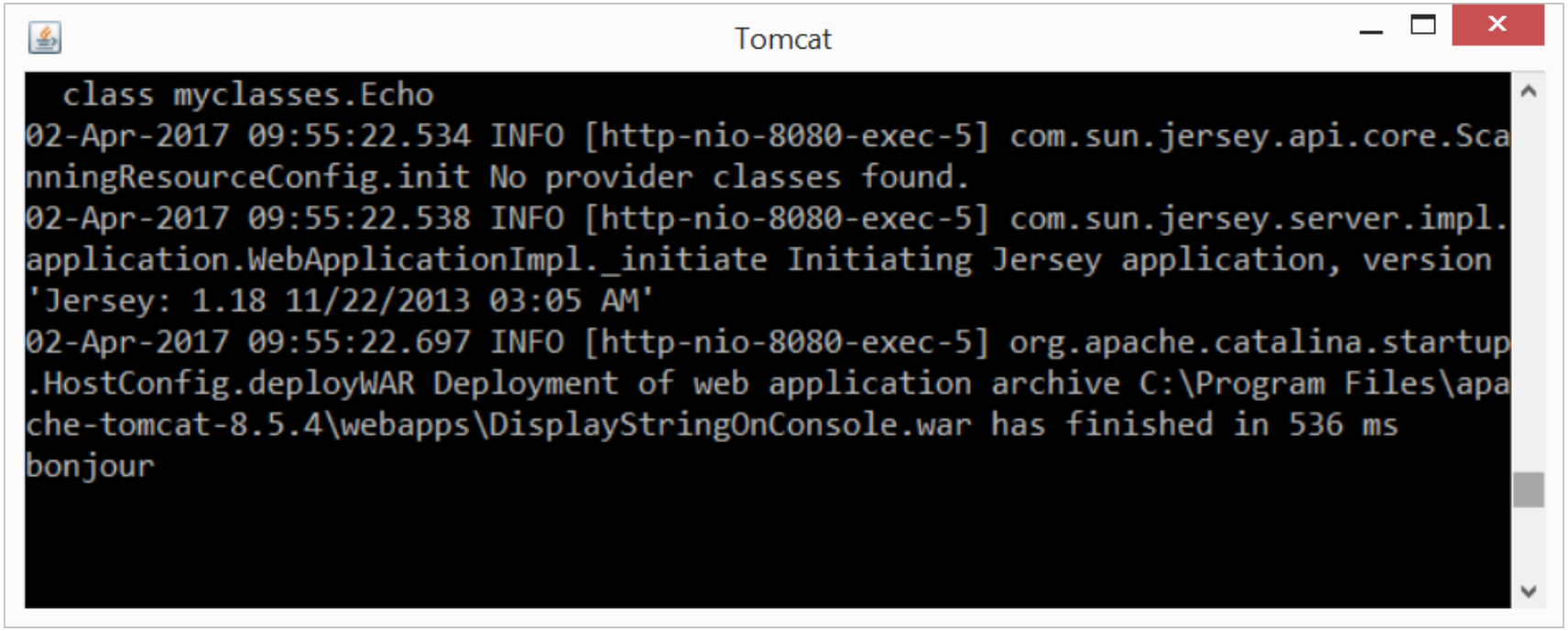
Client code

```
import javax.swing.JOptionPane;


public class EchoClient {

    public static void main(String[] args) throws Exception {
        String url = "http://localhost:8080/DisplayStringOnConsole";
        String message = JOptionPane.showInputDialog("???");

        HttpClient client = HttpClientBuilder.create().build();
        HttpPut request = new HttpPut(url);
        request.setEntity(new StringEntity(message));
        request.setHeader(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN);
        HttpResponse response = client.execute(request);

        System.out.println(response.getStatusLine().getStatusCode());
        }
}
```

Markers | Properties | Servers | Data Source Ex... | Snippets | Problems | Console | Mig

```
<terminated> EchoClient (1) [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (2 avr. 2017 à 10:11:37)
204
```

Result



Contents match between the client and the service

Content match example

“Accept” header to select the method on the server side

```
@Path(“typematch/")

public class Helloworld {
@GET
@Path("{name}")
@Produces(MediaType.TEXT_PLAIN)
   public String getHelloTXT(@PathParam("name") String name) {
      return "Hello World in plain text! " + name; }

@GET
@Path("{name}")
@Produces(MediaType.APPLICATION_XML)
   public String getHelloXML(@PathParam("name") String name) {
      return "<response> Hello World in XML! " + name + " </response>";}
```

Example: HTTP GET, client side

```
String url = "http://localhost:8080/FirstRSProject/typematch/bon-travail";
HttpClient client = HttpClients.createDefault();
HttpGet request = new HttpGet(url);
request.setHeader(HttpHeaders.ACCEPT, MediaType.TEXT_PLAIN);
HttpResponse response = client.execute(request);
System.out.println(response.getStatusLine().getStatusCode());
BufferedReader br = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));
String line = br.readLine();
while(line != null){
        System.out.println(line);
        line = br.readLine();}
```

```
Problems  @ Javadoc  Declaration  Console
<terminated> MyGETContentNegociationClient [Java Applicati
200
Hello World in plain text! bon-travail
```

Example: HTTP GET, client side

```
String url = "http://localhost:8080/FirstRSProject/typematch/bon-travail";
HttpClient client = HttpClients.createDefault();
HttpGet request = new HttpGet(url);
request.setHeader(HttpHeaders.ACCEPT, MediaType.APPLICATION_XML);
HttpResponse response = client.execute(request);
System.out.println(response.getStatusLine().getStatusCode());
BufferedReader br = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));
String line = br.readLine();
while(line != null){
        System.out.println(line);
        line = br.readLine();}
```

```
Problems  @ Javadoc  Declaration  Console
<terminated> MyGETContentNegociationClient [Java Application] C:\Prc
200
<response> Hello World in XML! bon-travail </response>
```