SOA
Web API
Jersey Framework

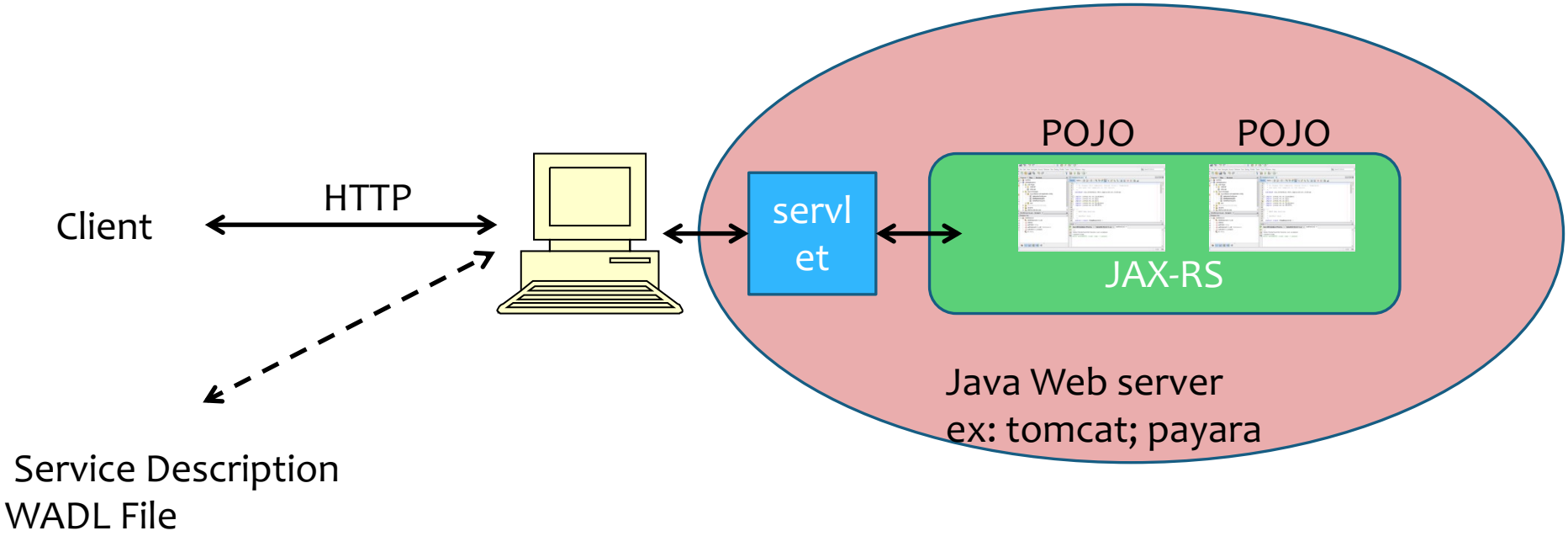
Introduction to Jersey

* Jersey Framework Java
* JAX-RS (JSR-370) implementation and more

* Description
  * Annotations used inside classes
  * Of the life cycle (via servlets)
  * Error management (exceptions)
  * Routing
  * Client side
  * Etc.

Architecture

* Jersey => Web API developpement based on coding of POJO, completed with annotations



Jersey : Web API side

* Web service development:

* Principle:
  * Rest is stateless, simple methods with serialisable object
  * Hateos would be great (we know the available resources and URI)

* Bottom/UP approach
  * Creation of POJO + annotations
  * Compilation then deployment

Jersey : Web API side

Path of the Service

```
@Path("generic")
public class GenericResource {
```

HTTP verb

```
@GET
@Produces("text/html")
```

MIME type returned

```
  public String getResp() {
    try{
    return "<html><body>test<br>coucou\n</body></html";
        } catch (Exception e) {
         throw new WebApplicationException(500);
  }
```

Error Management

Jersey : Web API side

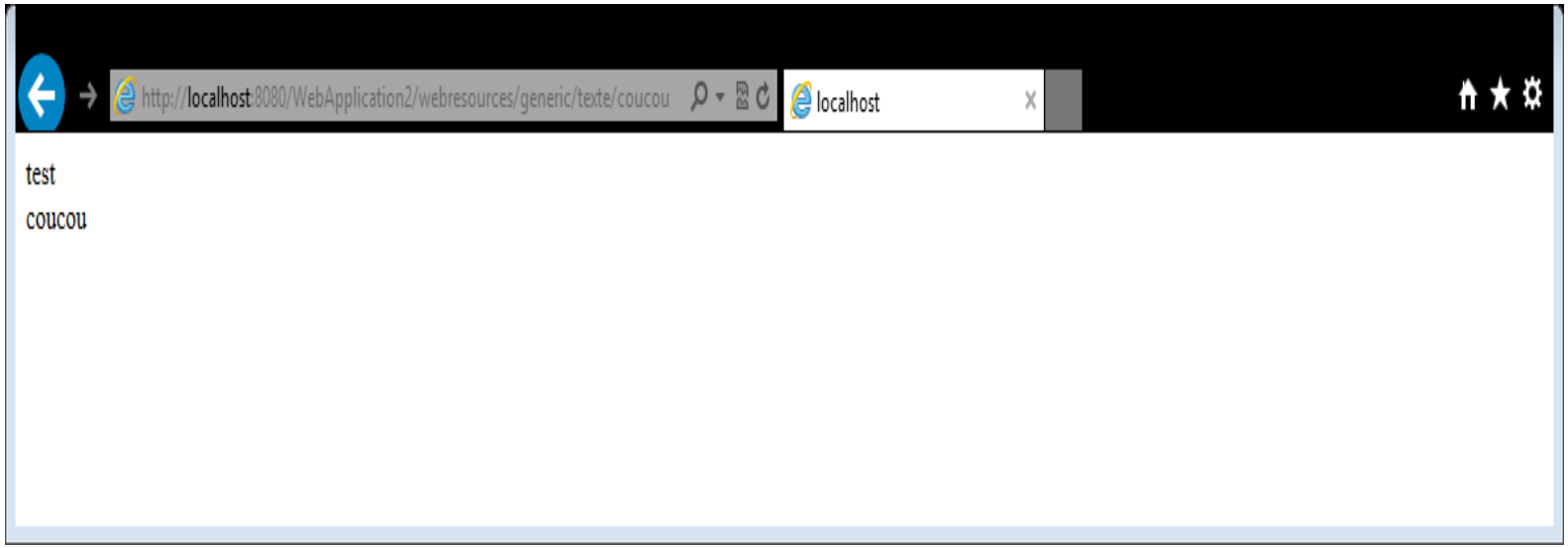
* Deployment configuration file Web.xml
* Mostly hidden by IDE

```
<servlet>
<servlet-name>EssaisServlet</servlet-name>
<servlet-class>bdtest.EssaisServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>EssaisServlet</servlet-name>
<url-pattern>/rest ou autre chose ou rien/EssaisServlet</url-pattern>
</servlet-mapping>
```

Jersey : Web API side

* For the GET verb, you can test with the browser:



* Otherwise : postman, SoapUI, etc. for testing

Jersey : Web API side

Annotations for :

* HTTP verb
* Parameter and parameter type declaration
* Return type declaration
* Serialisation (request and response)
* Header
* Scope (stateless, … or not)

Jersey : Web API side

* Annotation @path:
  * May be used to define a root resource (Service)
    * Path describing a part of the URI
  * Ex: With Tomcat Server & Netbeans
    http://localhost:8080/WebApplication2/webresources/generic

Web App. context

Service context

Jersey : Web API side

* Annotation @Path (again)
  * Annotation of methods
  * Fine grain resource declaration

```
@GET @Path(”appel”)
@Produces("text/html")
  public String getResp() {….}
```

  * http://localhost:8080/WebApplication2/webressources/generic/appel

Jersey : Web API side

* @Path + @PathParam
* Use of URI to send parameter

```
@GET @Path(”appel/{texte}”)
@Produces("text/html")
  public String getResp(@PathParam(”texte") String t) {…}
```

* http://localhost:8080/WebApplication2/webressources/generic/appel/coucou

Jersey : Web API side

* Example :

```
@GET
  @Path("appel/{texte}")
  @Produces("text/html")
  public String getOb(@PathParam("texte")  String texte) {

     return "<html><body>test<br>"+texte+ "\n</body></html";
     //TODO return proper representation object
   //  throw new UnsupportedOperationException();
  }
```

* Usage:

http://localhost:8080/WebApplication2/webressources/generic/appel/coucou

texte

Jersey : Web API side

* Example 2:

```
@GET
  @Path("appel/texte-{texte}-detail-{det}")
  @Produces("text/html")
  public String getOb(@PathParam("texte")  String t,
@PathParam("det")  String d) {

    return "<html><body>"+t+"<br>"+d+ "\n</body></html";
}
```

* usage:

http://localhost:8080/WebApplication2/webressources/generic/appel/texte-coucou-detail-textecourt

La zone texte

@HEAD, @Get, @Post, @Put, @Delete

* Always use the right verb

* In other terms, don’t use GET everywhere, even if that works
  * Does not comply with REST
  * No sense

  * Instead use CRUD:
  * @POST->creation ressource
  * @GET->Read
  * @PUT->Update
  * @DELETE-> Yes, delete

Jersey : Web API side

* @DefaultValue -> default value
* @PathParam -> see previously
* @QueryParam-> parameter transfert URL url?param1=val1&..
* @FormParam: parameter from HTML forms
* @CookieParam: parameter from cookie
* @HeaderParam: parameter from HTTP header
* @context: from complete HTTP request

Jersey : Web API side

Example with @QueryParam

```
@GET
   @Path("appel/")
   @Produces("text/html")
   public String getOb(@DefaultValue("textedefault ") @QueryParam(" texte")
String texte {

      return "<html><body>test<br>"+texte+ "\n</body></html";
}
```

* Appel: http://localhost:8080/WebApplication2/webressources/generic/appel/?texte=coucou

si d'autres paramètres : suivi de &param2=value2 etc.

Jersey : Web API side

Data serialisation (request and response)

* @Consumes used to specify MIME types for requests
* @Produces used to specify MIME types for responses
* Annotations can be added on classes or methods
* Caution : if nothing is given, all the types should be accepted

Jersey : Web API side

```
@GET
  @Path("appel")
  @Produces("text/html")
  public String getHTMLOb(@DefaultValue("textedefault ")
@QueryParam(" texte") String texte {

    return "<html><body>test<br>"+texte+ "\n</body></html";
}

@Path("appel")
  @Produces("text/xml")
  public String getXMLOb(@DefaultValue("textedefault ")
@QueryParam(" texte") String texte {

    return "<?xml version=\"1.0\"?>"+"<contenu>livre vide</contenu>";
}
```

Jersey : Web API side

More detailled, better ?

```
@Path("appel")
   @GET
   @Produces("application/xml")
   public Response getXml(@DefaultValue("textedefault")
@QueryParam("texte") String texte ) {

Response response =
Response.status(200).type(MediaType.TEXT_XML).entity("<?x
ml version=\"1.0\"
?>"+"<contenu>"+texte+"</contenu>").build();
      return response;
   }
```

Jersey : Web API side

* Use your own Business classes :

* Need to specify mapping between XML/JSON and classes

1. Class coding with annotations JAXB *@XmlRootElement, @XmlType*

2. 3. Serialisation performed w.r.t. the annotations @Produces et @Consumes (text/xml, application/xml, application/json et so on)

Jersey : Web API side

Simple example:

```
@XmlRootElement (name = ''texte'')

Public Class Texte {

Protected String contenu;
Public String getContenu(){ return contenu;}
Public void setContenu(String s){this.contenu=s;}
Public String toString(){ return contenu;}
}
```

Jersey : Web API side

```
@PUT
   @Path("update")
   @Consumes(" application/xml")
   public Void uptexte(Texte t) {
            System.out.println(t.getContenu);
}

@GET
   @Path("appel")
   @Produces(" application/xml")
   public Texte gettexte() {
            Texte t=new Texte();
            t.setContenu(''bla bla'');
            return t;
}
```

* JSON : @Produces("application/json")

Jersey : Web API side

Example 2 :

```
@XmlRootElement(name="customer")
@XmlAccessorType(XmlAccessType.FIELD)
//means all the non sattic fields are taken into account
Public class Customer {

@XmlAttribute(required=true) protected int id;
//attribute put inside the XML elt. <id> </id>
@XmlElement(required=true) protected String firstname;
@XmlElement(required=true) protected String lastname;
@XmlElement(required=true) protected Address address;
@XmlElement(required=true) protected String email;
@XmlElement (required=true) protected String phone; public
Customer() { } // Getter and setter methods // ...
}
```

Jersey : Web API side

* Reponse Management; HTTP statut

* Response without error
  * Code 200 with content
  * Code 204 without content

* Response with error
  * from 400 to 599
  * Ex: 404 not found
  * Ex: 405, method not allowed

Jersey : Web API side

Response class as a response

* Offers the poosibility to
* Choose an HTTP code
* Add parameters within the header
* Of course we can also add a body

* Instanciation of a response with the Builder pattern

Jersey : Web API side

Response class as a response

* Examples:

* ResponseBuilder ok() => statut 200

* ResponseBuilder serverError() => statut >400

* ResponseBuilder status(Response.Status): for choosing any status code
  * Status: Responses.NOT_FOUND
  * Response.Status.OK,
  * or any number

Jersey : Web API side

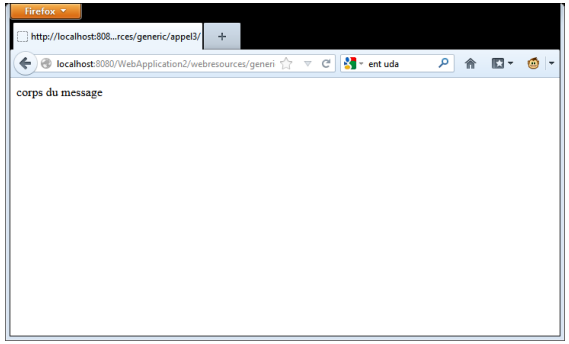
* Méthodes de la classe Builder utiles:
  * Response build() : crée une instance
  * ResponseBuilder entity(Objet o): modifie contenu du corps
  * ResponseBuilder header(String, Object): modifie un paramètre de l'entête HTTP

Jersey : Web API side

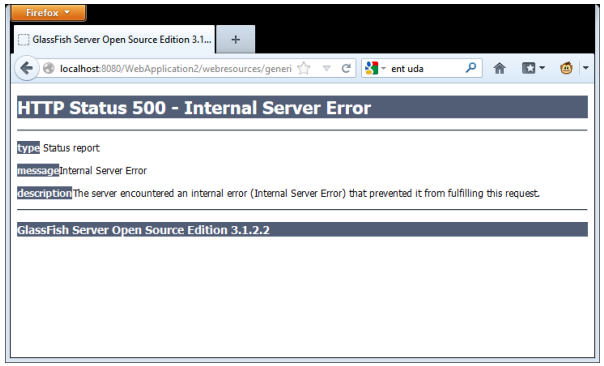
```
@Path("appel")
Public class Exempleresponse {
@Path("response")
@GET
Public Response getText(){
Return Response
        .status(Response.Status.OK)
        .header("param1","test");
        .entity("corps du message");
        .build();
        }
        }
```

```
@GET
Public Response getText2(){
Return Response
        .serverError()
        .build();
        }
        }
```

HTTP Status code 200



HTTP Satus code 500



Jersey : Web API side

Error handling ; Exceptions

* Jersey provides an exception, which is later converted into a response **WebApplicationException (see doc.)**

```
@GET
@Path("/images/{image}")
@Produces("image/*")
public Response getImage(@PathParam("image") String image) {
  File f = new File(image);

  if (!f.exists()) {
    throw new WebApplicationException(404);
  }
  String mt = new MimetypesFileTypeMap().getContentType(f);
  return Response.ok(f, mt).build();
}
```

Jersey : Web API side

Error handling ; Exceptions

* Dev. of a child class of **WebApplicationException**
* Example : Class NotFoundException with status code 404
* For being called as : *throw new NotFoundException("Item, " + itemid + ", is not found");*

Jersey : Web API side

```
1 public class NotFoundException extends WebApplicationException {
2
3   /**
4    * Create a HTTP 404 (Not Found) exception.
5    */
6   public NotFoundException() {
7      super(Responses.notFound().build());
8   }
9
10   /**
11    * Create a HTTP 404 (Not Found) exception.
12    * @param message the String that is the entity of the 404 response.
13    */
14   public NotFoundException(String message) {
15      super(Response.status(Responses.NOT_FOUND).
16            entity(message).type("text/plain").build());
17   }
18
19 }
```

Jersey : Client side

Jersey : Client side

* Coding of Java client apps. from scratch

* Auto Generation of client apps with Netbeans ☺ but limited

Jersey : Client side

Jeysey provides a class Client to perform Web Api calls

* Client client = ClientBuilder.newClient();
* ClientConfig clientConfig = new ClientConfig();
  * Allows to inject some configuration properties (ex: SSL, etc.)

Jersey : Client side

* Class WebTarget
  * To give the HTTP ressource
  * Along with parameters

```
WebTarget webTarget =
client.target("http://example.com/rest");

WebTarget helloworldWebTarget =
resourceWebTarget.path("helloworld");

WebTarget helloworldWebTargetWithQueryParam =
    helloworldWebTarget.queryParam("greeting", "Hi
World!");
```

Jersey : Client side

* Class Invocation
  * Used for calling Web API and get a response

```
Invocation.Builder invocationBuilder =
    helloworldWebTargetWithQueryParam.request(MediaType.TEXT_PLAIN_TYPE);
invocationBuilder.header("some-header", "true");

Response response = invocationBuilder.get();
System.out.println(response.getStatus());
System.out.println(response.readEntity(String.class));
```

Jersey : Client side

* Fluent support

```
Client client = ClientBuilder.newClient(new ClientConfig()
        .register(MyClientResponseFilter.class)
        .register(new AnotherClientFilter()));

String entity = client.target("http://example.com/rest")
        .path("resource/helloworld")
        .queryParam("greeting", "Hi World!")
        .request(MediaType.TEXT_PLAIN_TYPE)
        .header("some-header", "true")
        .get(String.class);
```

Here a String element is retrieved. But others class types can be used

Jersey : Client side

* Fluent support

* Short version:
* (but we loose many options)

```
String responseEntity = ClientBuilder.newClient()
    .target("http://example.com").path("resource/rest")
    .request().get(String.class);
```

Jersey : Client side

* Method get()
* But also methods put(), post() etc.
* Some examples in the next slidees

Jersey : Client side

Example using a template to define the type of response

```
public class Jclientex04 {
  private WebTarget webTarget;
  private Client client;
  private static final String BASE_URI = "http://localhost:8080/restws-war/webresources";

  public Jclientex04() {
    client = javax.ws.rs.client.ClientBuilder.newClient();
    webTarget = client.target(BASE_URI).path("ex04");
  }

  public <T> T getXml(Class<T> responseType) throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path("appel")
                .queryParam("texte", "coucou");

    return resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
  }
```

Jersey : Client side

Calling the previous methods :

```
Jclientex04 c= new Jclientex04();
    String s=c.getXml(String.class);
    System.out.println(s);
```

You can change the type here

Jersey : Client side

Complete example :

Service code

```
public Ex05jsonResource() {
  }

  /**
   * Retrieves representation of an instance of ws.Ex05jsonResource
   * @return an instance of java.lang.String
   */
  @Path("appel")
  @GET
  @Produces("application/json")
  public Texte getJson() {
    Texte t= new Texte();
    t.setContenu("json test");
    return t;

  }
```

Jersey : Client side

```
@Path("appelput")
   @PUT
   @Consumes("application/json")
   @Produces("application/json")
   public Texte putJson(Texte t) {

try{
   t.setContenu(t.getContenu()+"passé par là");
   return t;
} catch (Exception e) {throw new WebApplicationException(404);} //gestion des erreurs
coté Service !
}
}
```

Jersey : Client side

Client App. Code

```
public Jclientex05() {
    client = javax.ws.rs.client.ClientBuilder.newClient();
    webTarget = client.target(BASE_URI).path("ex05");
  }

  public <T> T getJson(Class<T> responseType) throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path("appel");
    return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
  }

  public Response putJson(Texte ent ) throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path("appelput");
    return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).put(Entity.entity(ent,
MediaType.APPLICATION_JSON));
```

Création de clients Jersey

```
//calling the previous method (in main())

try{
 Jclientex05 c2= new Jclientex05();
 Texte t=c2.getJson(Texte.class);
 System.out.println(t.toString());
 }catch(ClientErrorException e){//to do}

Texte tinput=new Texte();
 tinput.setContenu("second test");
 Response r=c2.putJson(tinput);
 if (r.getStatus()==200)
 {
 Texte t3=(Texte) r.readEntity(Texte.class);
 System.out.println(t3.getContenu());
```

Error handling

Either by checking the status

Or with an exception

Jersey : Client side

* Authentication/ encryption

* **SSL**
  * SSL Context

  SSLContext ctx = SSLContext.getInstance("SSL"); ctx.init(null, myTrustManager, null); config.getProperties().put(HTTPSProperties.PROPERTY_HTTPS_PROPERTIES, new HTTPSProperties(hostnameVerifier, ctx));

* **Http Authentication (basicAuth)**
  * HttpAuthenticationFeature

  HttpAuthenticationFeature feature = HttpAuthenticationFeature.basic("user", "superSecretPassword");

Jersey : Client side

* Authentication/ encryption

* **Http Authentication (basicAuth)**
  * HttpAuthenticationFeature

  HttpAuthenticationFeature feature = HttpAuthenticationFeature.basic("user", "superSecretPassword");

* **Other available protocols**
  * OauthV2 etc.
  * https://eclipse-ee4j.github.io/jersey.github.io/apidocs/2.29/jersey/org/glassfish/jersey/client/oauth2/OAuth2ClientSupport.html

Jersey : Client side

* Authentication/ encryption

Oauth v2 overview