**JAX-RS Annotations**

1. **@Path**

* It identifies a particular “Response Method” in a “Root Resource Class”
* It can be specified at “Class” or “Method” level
* Declaration at Class Level is Mandatory. However declaration at Method Level is Optional
* If its not present at Method Level then always First Method gets executed.
* Avoid using spaces in Path Name. Instead use underscore( \_ ) or hyphen ( - ) while using a long resource name.

For example,

Use “/create\_employee” instead of

“/create employee”

* Use lowercase letters in Path Name
* A @Path value may or may not begin with a ‘/’ , it makes no difference.
* Likewise, by default, a @Path value may or may not end in a ‘/’, it makes no difference.
* Thus request URLs that end or do not end in a ‘/’ will both be matched.

1. **@<ResourceMethodDesignators>**

(@GET, @POST, @PUT, @DELETE, @HEAD, and @OPTIONS )

* This annotations are used with Java Methods & they are called as “Resource Method Designator Annotations”
* The Java method annotated with @GET will process HTTP GET requests
* The Java method annotated with @POST will process HTTP POST requests.
* The Java method annotated with @PUT will process HTTP PUT requests.
* The Java method annotated with @DELETE will process HTTP DELETE requests.
* The Java method annotated with @HEAD will process HTTP HEAD requests.
* The Java method annotated with @OPTINS will process HTTP OPTIONS requests
* NOTE: There is NO @TRACE and @CONNECT annotation
* Its Mandatory Information & every Resource Method should have ONLY ONE Resource Method Designator.

NOTE: For Resource Methods @Path is Optional, however,

@<Resource\_Method\_Designators> is Mandatory (ONLY ONE)

1. **@<\*>Param (Parameter Annotations)**

**( @PathParam, @QueryParam, @FormParam, @HeaderParam, @CookieParam and @MatrixParam)**

* Parameters of a “resource method” may be annotated with parameter-based annotations to extract information from a request
* Usually, these annotations are used on the input arguments of a “Response Methods”
* **A. @PathParam**
* It represents the parameter of the URI path

Syntax: {variable\_name}

Ex: @Path(“/users/{username}”)

* This annotation allows us to extract values from extract a path parameter from the path component of the request URL
* It can be used with Regular Expressions

Syntax: {variable\_name : regular\_expression }

Ex: @Path(“{id : \\d+}”)

//It supports digit only

* **B. @QueryParam**
* This annotation allows us to extract values from URL Query Parameters
* **C. @FormParam**
* This annotation allows us to extract values from “posted” form data
* This annotation is used to access “application/x-www-form-urlencoded” request bodies.
* In other words, whenever we submit the form which has metod = “post” then request header will have

“Content-Type: application/x-www-form-urlencoded”

* It should not be used with @GET
* D. **@HeaderParam**
* This annotation allows us to extract values from HTTP request headers.
* **E. @CookiesParam**
* This annotation allows us to extract values from HTTP request cookies.
* **F. @MartrixParam**
* Matrix parameters are a set of “name=value” in URI path

For Ex: /users/Praveen;userid=abcd

URI can consist of N number of Matrix parameters but they should be separated by a semi colon “;”

* They are optional & they can be present between “Path” & “Query String” in URI
* This annotation allows us to extract values from URI matrix parameters.
* All these Parameter Annotations refer various part of an HTTP request. These parts are represented as a string of characters within the HTTP requeset.
* So we can get them as a String values or else JAX-RS can convert this String data into any Java type that we want, provided that it matches one of the following criteria:

1. Be a primitive type (byte, short, int, long, float, double, char & Boolean)
2. Be a

* List<T>
* Set<T> or
* SortedSet<T>

resulting collection which is read-only

1. Have a Class Name which has constructor that accepts a single string argument

Example:

@Path(“/{regno : \\d+}”)

public String getStudentData(

@PathParam(“regno”) long regno,

@QueryParam(“result”) char passFlag,

@QueryParam(“courseid”) SortedSet<Byte> cid,

@FormParam(“fees”) double amount,

@CookieParam(“schoolNM”) String name,

@CookieParam(“schoolLocation”) String loc,

@MatrixParam(“pincode”) int code,

@HeaderParam(“actice”) boolean state)

{

.

.

}

1. **@DefaultValue**

* Assigns a default value to a parameter ( Parameters Annotations )
* If the @DefaultValue is not used in conjuctjion with “Parameter Annotations” and if any parameter is not present in the request then value will be
  + - an “empty collection” for List, Set or SortedSet
    - “null” for other object types and
    - “default values” for primitive types

1. **@BeanParam**

* The @BeanParam annotation is a new Annotation got added in JAX-RS 2.0 specification
* It allows us to inject an application-specific class whose property methods or fields are annotated with “Parameter Annotations”
* The JAX-RS runtime will introspect the @BeanParam parameter’s type for injection annotations and then set them as appropriate.

1. **@HttpMethod**

* Can Create new annotations that bind to Http methods other that GET, POST, PUT, DELETE, HEAD and PUT
* While HTTP is a ubiquitous, stable protocol, it is still constantly evolving. For example, consider the WebDAV Protocol.
* WebDAV protocol
* The WebDAV protocol makes the Web an interactive readable and writable medium.
* It allows users to create, change, and move documents on web servers.
* It does this by adding a bunch of new methods to HTTP like MOVE, COPY, MKCOL, LOCK, and UNLOCK
* Although JAX-RS does not define any WebDAV-specific annotations, we could create them ourselves using the @HttpMethod annotation:

package org.rest.webdav;

import…;

@Target({ElementType.METHOD})

@Retention(RetentionPolicy.RUNTIME)

@HttpMethod(“LOCK”)

public @interface LOCK {

…

}

* Here, we have defined a new @org.rest.webdav.LOCK annotation using @HttpMethod to specify the HTTP operation it binds to. We can then use it on JAX-WS resource methods:

@Path(“/employee”)

public class EmployeeResource{

@Path(“{id}”)

@LOCK

public void lockEmployeeID(@PathParam(“id”) String id){

…

}

}

* Now WebDAV clients can invoke LOCK operations on our web server and they will be dispatched to the lockEmployeeID() method.

1. **@Encoded**

* “Parameter Annotations” are encoded as per HTTP specification
* By default, JAX-RS decodes these values before converting them into the desired Java types.
* Sometimes, we may want to work with the raw encoded values.
* In this case, using the @javax.ws.rs.Encoded annotation gives us the desired effect:
* Ex:

@GET

@PRODUCES(“application/xml”)

public String getData(@Encoded @QueryParam(“regno”) String regnum) { … }

1. **@Produces**

* This annotation is used to specify the MIME media types of representations a resource can produce and send back to client.
* For example,

“text/plain”,

“application/json”,

“application/xml”, etc.,

* @Produces can be applied at both the class and method levels.
* If @Produces is applied at the class level, all the methods in a resource can produce the specified MIME types by default
* If it is applied at the method level, it overrides any @Produces annotations applied at class level
* For example:

@Path(“/myResource”)

@Produces(“text/plain”)

{

public class SomeResource

{

@GET

public String doGetAsPlainText(){

…

}

@GET

@Produces(“text/html”)

public String doGetHtml(){

…

}

}

* The doGetAsPlainText method defaults to the MIME type of the @Produces setting, and specifies that the method can produce HTML rather than plain text.
* The value of @Produces is an array of String of MIME types. For example:

@Produces({“tjmage/jsped”,”image/png”})

* Hence more than one media may be declared in the same @Produces declaration.

Ex:

@GET

@Produces({“application/xml”, “application/json”})

public String doGetAsXmlOrJson(){

…

}

* The doGetAsXmlOrJson method will get invoked if either of the media types application/xml and application/json are acceptable.
* If both are equally acceptable (i.e, Request with Accept Header value as “\*/\*”), then the former will be chosen because it occurs first.
* If no methods in a resource are able to produce the MIME type in a client request, the Jersey runtime sends back an HTTP “406 Not Acceptable” error

1. @Consumes

* This annotation is used to specify the MIME media types of representation a resource that it can consume from the client.
* @Consumes can be applied at both the class and the method levels
* If it is applied at the class level, all the methods in a resource can consume the specified MIME types by default.
* If it is applied at the method level, it overrides any @Consumes annotations applied at the class level
* The value of @Consumes is an array of String of acceptable MIME types. For example:

@Consumes({“text/plain”,”text/html”})

* If a resource is unable to consume the MIME type of a client request, the Jersey runtime sends back an HTTP “415 Unsupported Media Type” error.
* **javax.ws.rs.core.MediaType**
  + - * **It’s a Concrete Class part of JAX-RS API which has lot of Constants with most popular MIME Types**
      * **Rather that typing MIME media types, It is possible to refer to constant values, which may reduce typographical errors**
      * **EX:**

Rather than typing

@Produces(“application/xml”)

We can use

@Produces(“application/xml”)

**JAXB and JSON JAX-RS Handlers**

* + - Once we apply JAXB annotation to Java classes,

with JAX-RS api it is very easy to exchange XML/JSON data between client and web services

* + - The built-in JAXB and JSON (Jettison, Jackson, etc...,) handles will automatically takes care of Marshalling & Unmarshalling of these Java Classes to XML/JSON
    - Also, by default, JAX-RS api will take care of the creation and initialization of JAXBContext instances
    - Because the creation of JAXBContext instances can be expensive, JAX-RS implementation usually cache them after they are first initialized.