Consider a movie library service which returns the list of movies released on a given year.

**Resource**

Method: GET

URL: /movie/{year}

**Parameters**: Year

|  |  |  |
| --- | --- | --- |
| **Response** | **HTTP Status Code** | **Description** |
| Success | 200 – Ok | Movie list is returned |
| Input Invalid | 400 – Bad Request | The input year maybe invalid |
| Failure | 500 – Internal Error | Uncaught exception occurred such as RuntimeException |

**Setup a Web project**

Create a dynamic web project in eclipse and name it jaxrs-movie-service. I have used a JEE 6 server i.e. JBoss 7.

*Note: You can use maven and create a****maven-archetype-webapp****project as well. We won’t need any jars in our lib as JAXRS is part of JEE 6.*

**Deployment descriptor updates**

Update web.xml to match the below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"      xmlns="<http://java.sun.com/xml/ns/javaee>"      xsi:schemaLocation="<http://java.sun.com/xml/ns/javaee> <http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd>"      id="WebApp\_ID" version="3.0">      <display-name>jaxrs-movie-service</display-name>        <servlet-mapping>          <servlet-name>javax.ws.rs.core.Application</servlet-name>          <url-pattern>/\*</url-pattern>      </servlet-mapping>  </web-app> |

Create an empty beans.xml under WEB-INF folder. (This will enable CDI capabilities which can be made use of if desired).

**The source code**

Let’s define a POJO which will hold the movie information.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package org.prashantpro.jaxrs.movie;    /\*\*   \* @author Prashant Padmanabhan <[https://javaspecialist.wordpress.com](https://javaspecialist.wordpress.com/)>   \*   \*/  public class Movie {        private int year;      private String title;      private String genre;        public Movie(int year, String title, String genre) {          this.year = year;          this.title = title;          this.genre = genre;      }      //Getters and setters left out for brevity.      ...  } |

Next, create the REST service which will expose the movie libraries movie list. Here we aren’t using any checked business exception handlers. This is a simple service which returns the HTTP Status code based on conditional constructs.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | package org.prashantpro.jaxrs.movie;    import java.util.ArrayList;  import java.util.List;    import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.PathParam;  import javax.ws.rs.Produces;  import javax.ws.rs.QueryParam;  import javax.ws.rs.core.MediaType;  import javax.ws.rs.core.Response;    /\*\*   \* @author Prashant Padmanabhan <[https://javaspecialist.wordpress.com](https://javaspecialist.wordpress.com/)>   \*   \*/  @Path("/movie")  @Produces(MediaType.APPLICATION\_JSON)  public class MovieLibrary {      //Store the in memory movie list as our repository.      static final List MOVIE\_LIST = new ArrayList();        //Build a dummy list of movies to work with.      static {          MOVIE\_LIST.add(new Movie(1971,"Dirty Harry","Action"));          MOVIE\_LIST.add(new Movie(2008,"Gran Torino","Drama"));          MOVIE\_LIST.add(new Movie(2012,"Argo","Drama"));      }        @GET      @Path("/{year}")      public Response getMovies(@PathParam("year") int year) {          if(year < 1880 || year > 9999) {              //Invalid input for year so return HTTP Status 400              return Response.status(Response.Status.BAD\_REQUEST).build();          }          List list = getMoviesByYear(year);          return Response.status(Response.Status.OK).entity(list).build();      }        private List getMoviesByYear(int targetYear) {          List found = new ArrayList();          for(Movie movie : MOVIE\_LIST) {              if(movie.getYear() == targetYear)                  found.add(movie);          }          return found;      }    } |

Once the above code is deployed as a WAR file in a JEE 6 server such as JBoss 7, we can then hit the URL:

[*http://localhost:8080/jaxrs-movie-service/movie/1971*](http://localhost:8080/jaxrs-movie-service/movie/1971)

Where;

**jaxrs-movie-service** – Is the context name of our application

**/movie** – Is the REST resource identified by *@Path(“/movie”)*

**1971 – I**s the input which maps to the *getMovies* method via *@Path(“/{year}”)*

The above should result in the below response:

Response headers:

* Status Code: 200 OK
* Content-Type: application/json

Response body:

|  |  |
| --- | --- |
| 1 | [{"year":1971,"title":"Dirty Harry","genre":"Action"}] |

This was simple to begin with, but in real world we would have some facade or service layer which would do the processing. The facade may throw business exceptions which would then result in passing a different response code with an appropriate message.

The complex case discussed below explains the same.

**Complex case**

Consider the movie library service would also require to list out movies based on a genre in the given order of ASC or DESC.

For simplicity let’s say we have only two genres “**Action**” and “**Drama**“.

In case of any problem the detailed message must be returned by the REST Service. This means we need to return some error messages along with the right HTTP Status code.

We can start throwing business exceptions but these won’t get translated to an appropriate HTTP Response code.

Here, we make use of Exception Mapping providers. These map a checked or runtime exception to an instance of Response. An exception mapping provider implements the ExceptionMapper<T> interface and is annotated with @Provider.

**Resource**

Method: GET

URL: /movie/list

**Parameters**: order, genre

|  |  |  |
| --- | --- | --- |
| **Response** | **HTTP Status Code** | **Description** |
| Success | 200 – Ok | Movie list is returned |
| Input Invalid | 400 – Bad Request | Service must return the details of what went wrong.Example genre specified was wrong.Sort order related issue. |

Let’s create a application exception which would contain the error details.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package org.prashantpro.jaxrs.movie;    import java.util.List;    /\*\*   \* @author Prashant Padmanabhan <[https://javaspecialist.wordpress.com](https://javaspecialist.wordpress.com/)>   \*   \*/  public class BusinessException extends Exception {        private static final long serialVersionUID = 1L;        private List messages;        public BusinessException(List messages) {          super();          this.messages = messages;      }        public List getMessages() {          return messages;      }  } |

Now, the Exception Mapper which will map our Business Exception to the correct Response code.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package org.prashantpro.jaxrs.movie;    import javax.ws.rs.core.Response;  import javax.ws.rs.ext.ExceptionMapper;  import javax.ws.rs.ext.Provider;    /\*\*   \* @author Prashant Padmanabhan <[https://javaspecialist.wordpress.com](https://javaspecialist.wordpress.com/)>   \*   \*/  @Provider  public class ExceptionHttpStatusResolver implements          ExceptionMapper {        @Override      public Response toResponse(BusinessException exception) {          Response.Status httpStatus = Response.Status.INTERNAL\_SERVER\_ERROR;            if (exception instanceof BusinessException)              httpStatus = Response.Status.BAD\_REQUEST;            return Response.status(httpStatus).entity(exception.getMessages())                  .build();      }  } |

The **MovieLibrary.java**gets a new method as shown below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | @GET  @Path("/list")  public Response getMoviesByGenre(@QueryParam("order") String order,@QueryParam("genre") String genre) throws BusinessException {      List errorMessages = new ArrayList();        if(order == null || order.length() == 0) {          errorMessages.add("order is required");      }        if(genre == null || genre.length() == 0) {          errorMessages.add("genre is required");      }        if(!"ASC".equals(order) && !"DESC".equals(order)) {          errorMessages.add("order of either ASC or DESC must be specified");      }        if(!"Action".equals(genre) &&  !"Drama".equals(genre)) {          errorMessages.add("genre of either Action or Drama must be specified");      }      if(!errorMessages.isEmpty()) {          throw new BusinessException(errorMessages);      }        List list = listMoviesByGenre(genre,order);      return Response.status(Response.Status.OK).entity(list).build();  }    private List listMoviesByGenre(String genre, String order) {      //Just return the list as is as this is a demo      //We would use some logic to do the filtering and ordering in real world apps.      return MOVIE\_LIST;  } |

If you invoke the service without any of the required parameters, it would return the HTTP status code of 400 along with the detailed messages. This works because our BusinessException thrown during validation checks is processed by the ExceptionHttpStatusResolver which translates the exception to the correct Response code.

When a resource method throws an exception for which there is an exception mapping provider, the matching provider is used to obtain a Response instance. The resulting Response is processed as if the method throwing the exception had instead returned the Response.

Example invocation URL:  
<http://localhost:8080/jaxrs-movie-service/movie/list?order=dummy>

The above should result in the below response:

Response headers:

* Status Code: 400 Bad Request
* Content-Type: application/json

Response body:

|  |  |
| --- | --- |
| 1  2  3  4  5 | [  "genre is required",  "order of either ASC or DESC must be specified",  "genre of either Action or Drama must be specified"  ] |