Objectives

- Explore the HATEOAS idea with JAX-RS
  - Create a todo-like web service with paging

HATEOAS

- HATEOAS is a fundamental aspect of the web and REST architectures
- Architectural principle of the web describing *linking* and *form submission*
- HATEOAS is an acronym that stands for "Hypermedia As The Engine Of Application State"
- In plain terms, it says that RESTful should be self-describing
  - Example: If a resource is a large list of items, the service might return the result in pages with links or buttons for navigation

JAX-RS Service


```
@GET
public Response getMyList(@QueryParam("size") int size) {
    Gson gson = new Gson();
    String json;

    PagedHelper p = new PagedHelper();
    if (size == 0) {
        p.setList(list);
    } else {
        p.setList(list.subList(0, size));
    }

    p.setNext("next url (from Paging Service)");
    p.setPrev("prev url (from Paging Service)");

    json = gson.toJson(p, PagedHelper.class);

    return Response.ok(json).build();
}
```

- Use a JAX-RS service and query parameters to retrieve a *range* of items (obviously this example is not complete)

Front-End


```
function getListItems(p) {
  $.ajax({
    url : 'services/items?page=' + p,
    dataType : 'json',
    success : function (r) {
      app.items = r.list;
      app.nextpage = r.next;
      app.prevpage = r.prev;

      updateList();
    }
    });
  }
```

- Editing our previous AJAX request to access the additional information that is being returned as JSON
- Note that we can use the `.attr()` method from jQuery to set the `href` attribute dynamically

Cloning the Web Application

- We have set up a public repository containing a basic Java web application[1]
- You can clone this project and import to Eclipse as you have done previously
- Make sure you are in the Java EE perspective (upper right corner)

[1] `https://github.com/marks1024/exercise-paging-361`

Task

- Edit the starter web application as follows
  - Implement paged responses via query parameters (10 items per page)
  - Set the buttons so that the user can move, dynamically, forwards and backwards through the list
  - The buttons should become inactive if they would point past the beginning or end of the list
- Deploy your application to your local servlet container
- This exercise involves editing both the back-end and the front-end