JAX-RS Basics 2

Sang Shin
JPassion.com
"Learn with Passion!"

Topics

- Sub resource locators
- Building URIs
  - *UriBuilder, UriInfo* classes
- Exceptions
  - *WebApplicationException, ExceptionMapper* classes
- Security
  - *@Context, SecurityContext class*
- Deployment options
  - Web app or Java SE app

Sub-Resource Locators

What is Sub-resource locator?

- Sub-resource locator is a method
  - Annotated with *@Path* but Not annotated with *@GET* or *@POST*
  - Returns a sub-resource, which itself contains methods with *@GET* or *@POST* annotations
- Sub-resource locators support polymorphism
  - A sub-resource locator may return different sub-type resource depending on the request (for example a sub-resource locator could return different sub-type resource dependent on the role of the principal that is authenticated).

Example: Sub-resource Locator

```
@Path("/item")
public class ItemResource {

   // Sub-resource locator returns a sub-resource
   @Path("content")
   public ItemContentResource getItemContentResource() {
      if (someBusinessLogic()){
        return new ItemContentResource1();
      }
      else{
        return new ItemContentResource2();
      }
   }

}

// Sub-resource ItemContentResource1
public class ItemContentResource1 {
   @GET public Response get() { ... }
   @PUT @Path("{version}")
   public void put(
         @PathParam("version") int version) { ... }
}
```

Buidling URIs with UriBuilder & UriInfo Classes (Linking Things Together)

UriBuilder Class

- A very important aspect of REST is hyperlinks, URIs, in representations that clients can use to transition the Web service to new application states
  - "hypermedia as the engine of application state"
- Building URIs and building them is not easy with *java.net.URI*, which is why JAX-RS has the *UriBuilder* class that makes it simple and easy to build URIs

UriInfo Class

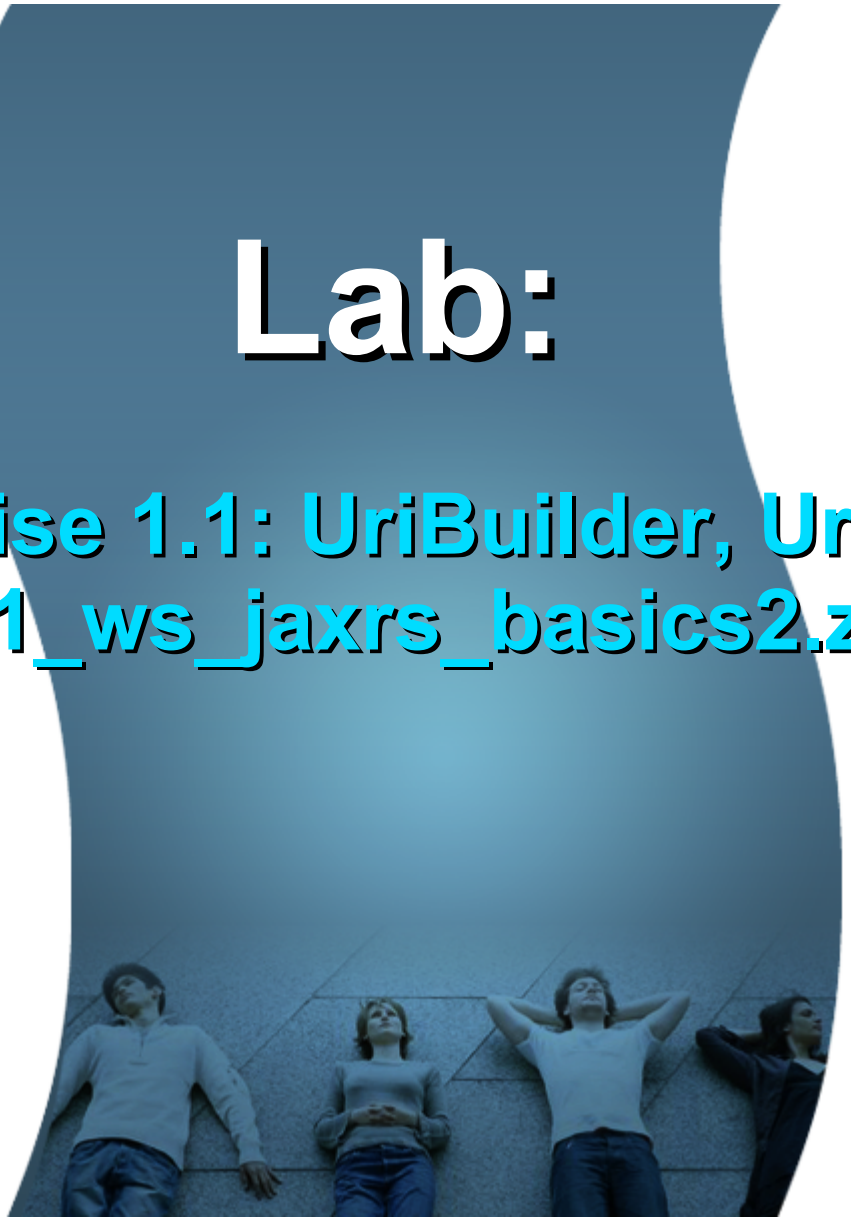
- Provides base URI information
  - The URIs that will be returned are typically built from the base URI the Web service is deployed at or from the request URI

UriBuilder & UriInfo

```
@Path("/users/")
public class UsersResource {

    @Context UriInfo uriInfo;
    ...

    @POST
    @Consumes(MediaType.APPLICATION_XML)
    public Response addUserFromXMLData(User user) throws IOException {

        UserDao.instance.getUserTestData().put(user.getId(), user);

        UriBuilder uriBuilder = UriBuilder.fromUri(uriInfo.getRequestUri());
        uriBuilder.path("{id}");
        return Response.created(uriBuilder.build(user.getId())).
                entity(user).
                type(MediaType.APPLICATION_XML).
                build();

    }
}
```

Lab:

Exercise 1.1: UriBuilder, UriInfo
4361_ws_jaxrs_basics2.zip



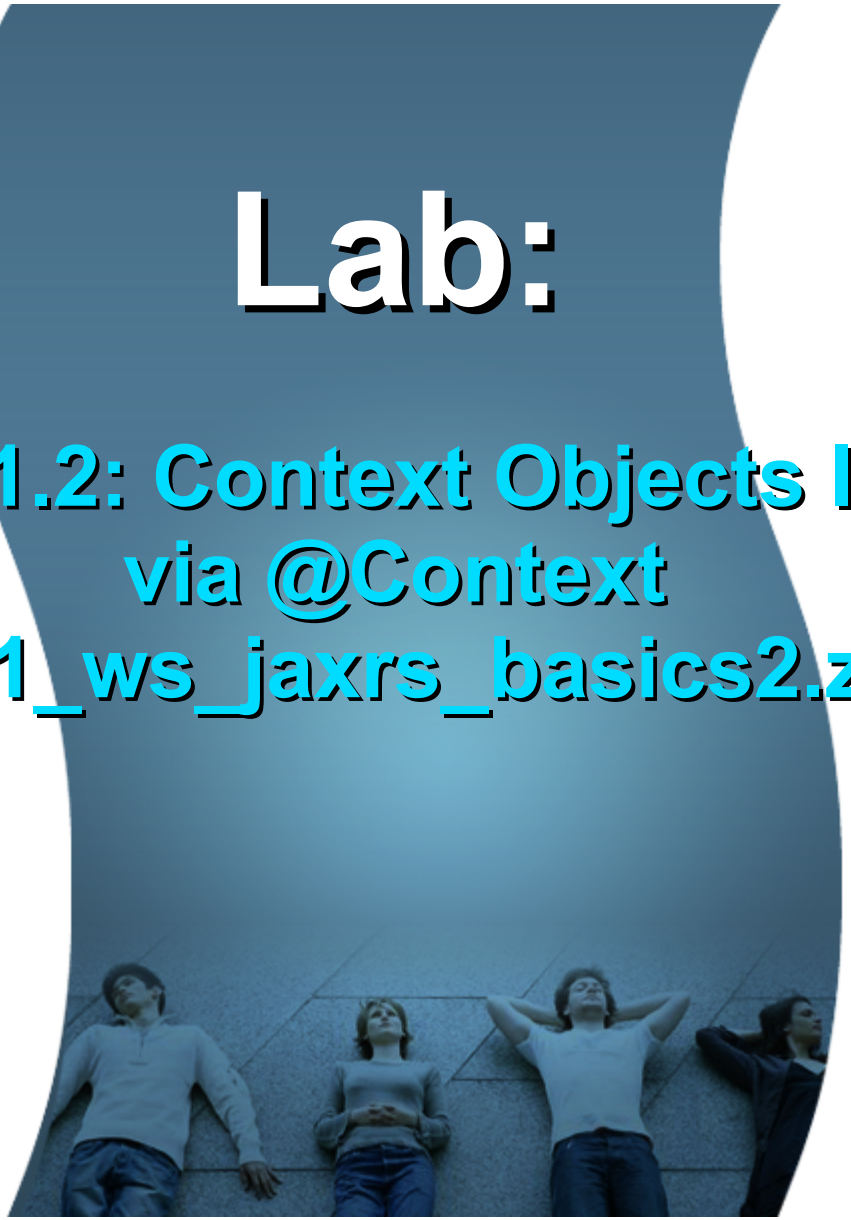
Context Object Injections via @Context

Context Object Injections via @Context

```
@Path("/users")
public class HelloWorldResource {

    @Context
    UriInfo uriInfo;

    @Context
    HttpHeaders httpHeaders;

    @Context
    SecurityContext securityContext;

    @Context
    Request request;

    @Context
    Providers providers;

...
```

Lab:

Exercise 1.2: Context Objects Injection via @Context
4361_ws_jaxrs_basics2.zip



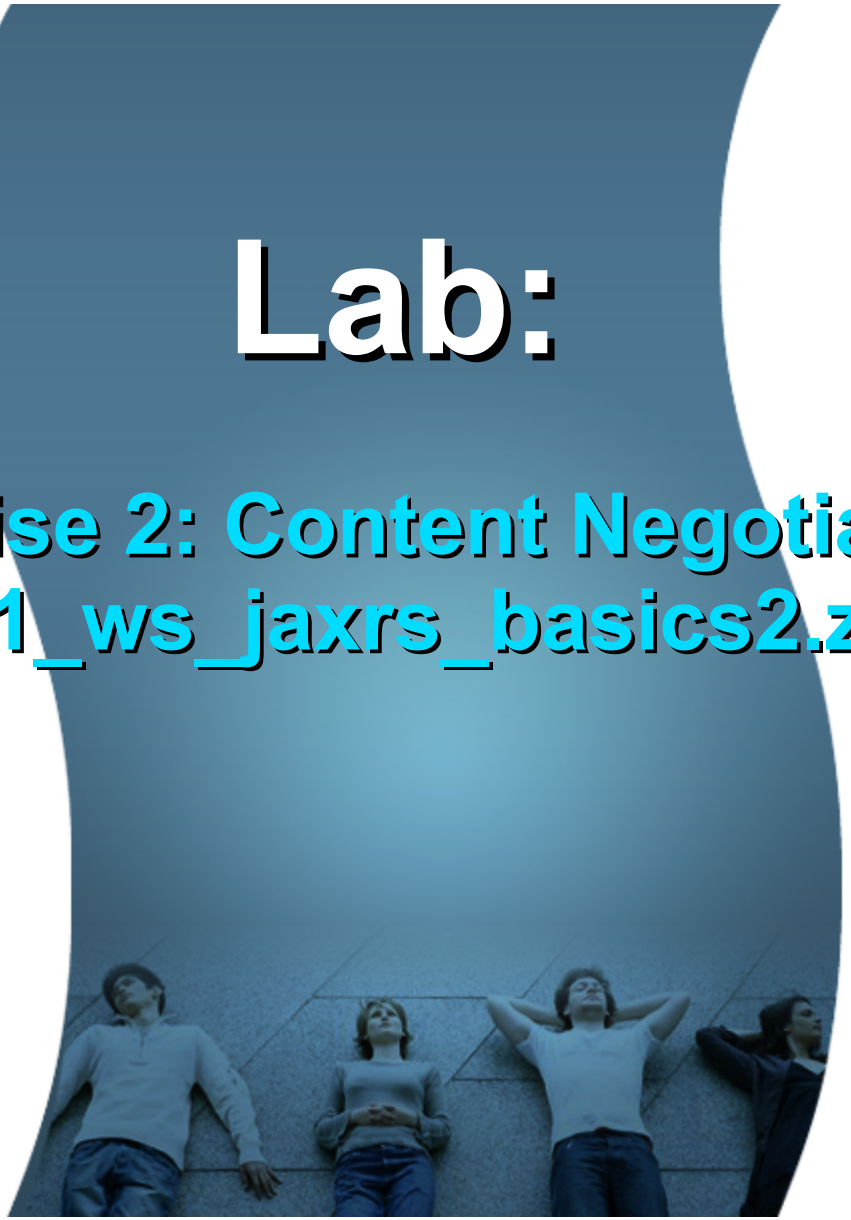
Content (Format) Negotiation

Content (Format) Negotiation

- One of the more popular features of REST applications is the ability to return different representations of resources
  - XML, JSON
- Client can ask content format in multiple ways
  - URL - resources/users/1.json
  - Request parameter – resources/users/1?format=json
  - Accept header

Lab:

Exercise 2: Content Negotiation
4361_ws_jaxrs_basics2.zip



Exception

NotFoundException

```
@Path("items/{itemid}/")
 public Item getItem(@PathParam("itemid") String itemid) {
        Item i = getItems().get(itemid);

        // Shows the throwing of a NotFoundException.
        // The NotFoundException exception is a Jersey specific
        // exception that extends WebApplicationException
        // and builds a HTTP response with
        // the 404 status code and an optional message
        // as the body of the response:
        if (i == null)
            throw new NotFoundException("Item,
                                        " + itemid + ", is not found");

        return i;
 }
```

WebApplicationException

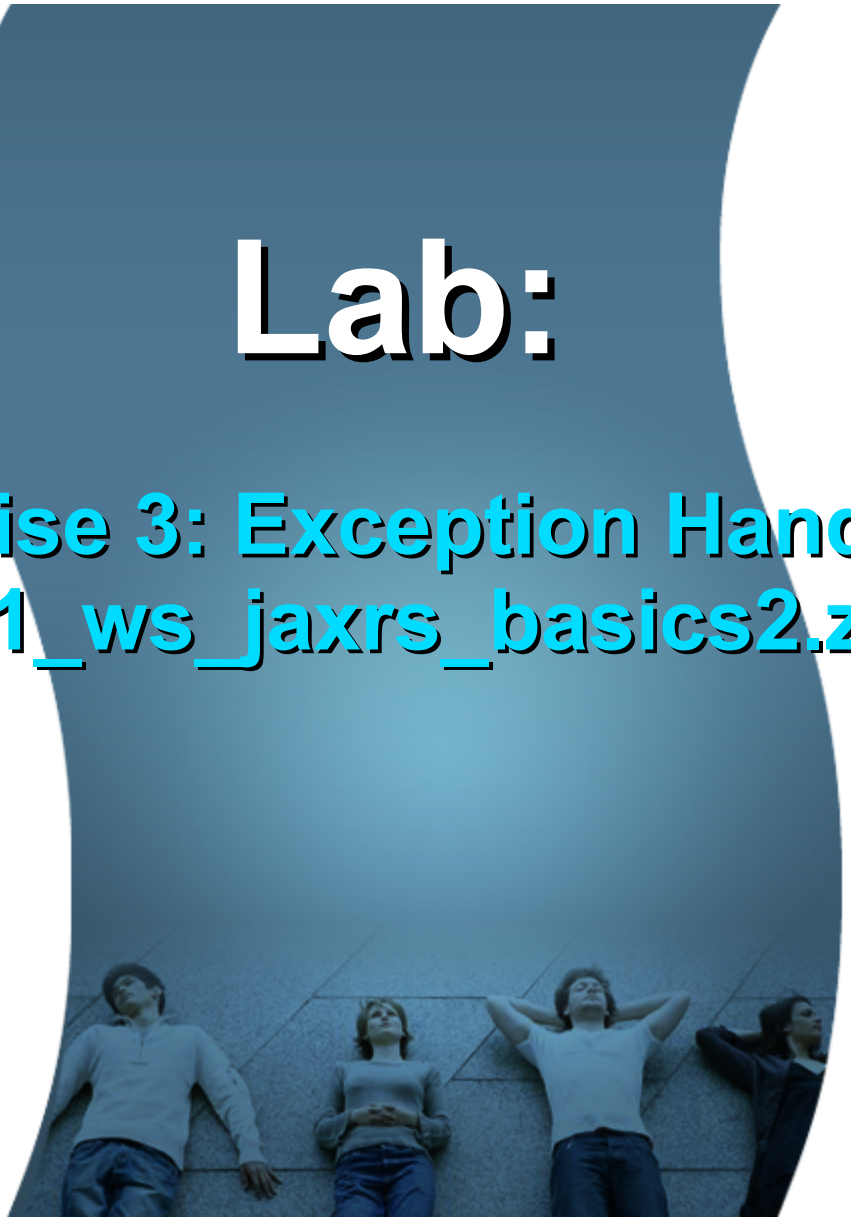
```
public class NotFoundException extends WebApplicationException {

    /**
     * Create a HTTP 404 (Not Found) exception.
     */
    public NotFoundException() {
        super(Responses.notFound().build());
    }

    /**
     * Create a HTTP 404 (Not Found) exception.
     * @param message the String that is the entity of the 404 response.
     */
    public NotFoundException(String message) {
        super(Response.status(Responses.NOT_FOUND).
                entity(message).type("text/plain").build());
    }

}
```

ExceptionMapper

```
// In other cases, it may not be appropriate to throw instances of
// WebApplicationException, or classes that extend
// WebApplicationException, and instead it may be preferable
// to map an existing exception to a response.
// For such cases it is possible to use the ExceptionMapper interface.
// For example, the following maps the EntityNotFoundException to a
// 404 (Not Found) response.
@Provider
public class EntityNotFoundMapper implements
        ExceptionMapper<javax.persistence.EntityNotFoundException> {

    public Response
    toResponse(javax.persistence.EntityNotFoundException ex) {
        return Response.status(404).
            entity(ex.getMessage()).
            type("text/plain").
            build();
    }
}
```

Lab:

Exercise 3: Exception Handling
4361_ws_jaxrs_basics2.zip



Security

Getting SecurityContext

- Security information is available by obtaining the *SecurityContext* using *@Context*, which is essentially the equivalent functionality available on the *HttpServletRequest*
- SecurityContext can be used in conjunction with sub-resource locators to return different resource depending on a user's role
  - For example, a sub-resource locator could return a different resource if a user is a preferred customer:

SecurityContext

```
@Path("basket")
// Sub-resource could return a different resource depending on
// if a user is a preferred customer or not
public ShoppingBasketResource get(@Context SecurityContext sc) {
    if (sc.isUserInRole("PreferredCustomer") {
        return new PreferredCustomerShoppingBaskestResource();
    } else {
        return new ShoppingBasketResource();
    }
}
```

Lab:

Exercise 4: Security
4361_ws_jaxrs_basics2.zip

Deployment Options

Servlet

- JAX-RS applications are packaged in WAR like a servlet
- For JAX-RS aware containers (Java EE 6)
  - web.xml can point to Application subclass
- For non-JAX-RS aware containers (Java EE 5)
  - web.xml points to the servlet implementation of JAX-RS runtime
- Application declares resource classes
  - Can create your own by subclassing
  - Reuse PackagesResourceConfig

JavaSE 6

- Use *RuntimeDelegate* class to create instance of end point
- Provides configuration information
  - Subclass of Application
- Jersey supports Grizzly, LW HTTP server and JAX-WS provider

```
Application app = new MyRESTApplication();
RuntimeDelegate rd = RuntimeDelegate.getInstance();
Adapter a = rd.createEndpoint(app, Adapter.class);
SelectorThread st = GrizzlyServerFactory.create(
    “http://127.0.0.1:8084/”, a);
```

RESTful Services

RESTful Services

- There are many RESTful services
  - Google
  - Flickr
  - Yahoo
- Data and its CRUD-based operations can be easily exposed as RESTful services

Lab:

Exercise 5: Build REST from Database 4361_ws_jaxrs_basics2.zip

Learn with Passion!
JPassion.com