Service Based Approach Intro, Beans, JAXB, Web Services and JAX-RS

WS Slides #1

Serviced Based Approach

The Web is a marvelous “application”

- Has been up 24/7 for 30-40 years
- Has been able to expand many magnitudes
- More users, more data, more advanced services , …
- … the perfect application?

Hmmm.. wouldn’t it be good to build our application like that??

- So what are the key principles behind the Web?

Representational State Transfer (REST)

Key principles that makes the web work and scale

1. **Identification of resources** (anything that can be named as a target of hypertext)
2. **Manipulating of resources through representations** (in responses we get an representation of the resource, for example as XML)
3. **Self-descriptive messages** (each message contains all the information necessary to complete the task. Other ways to describe this type of message is "stateless")
4. **Hypermedia as the engine of application state** (HATEOAS), the client/server interaction state is in the hypermedia they exchange (client guided through application)

// Roy Fielding, author of HTTP specification

Implementing REST

Practical interpretation of REST

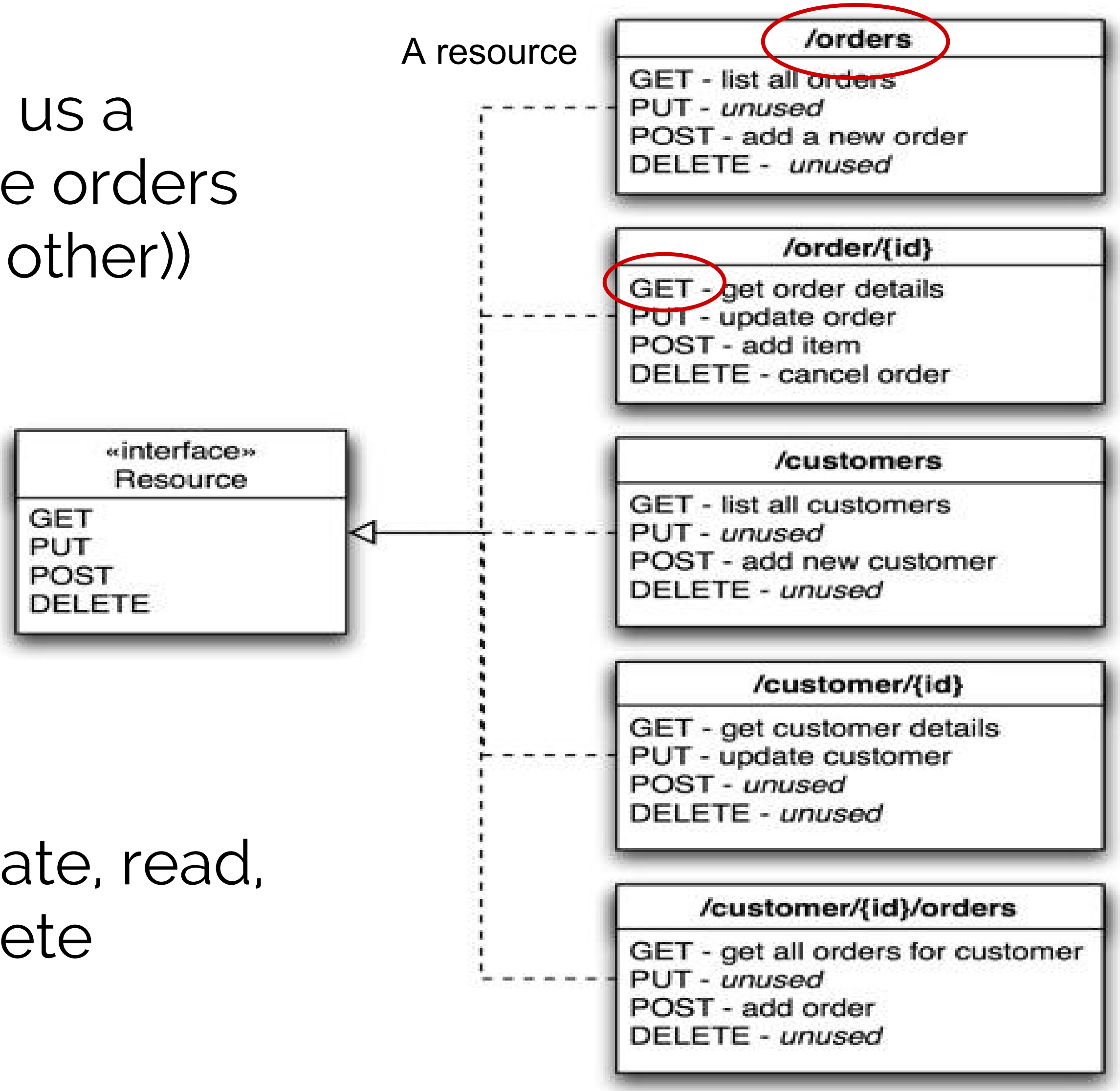
1. All resources accessible with URL's
2. Use XML (or JSON, or, .. more later) in HTTP-request/responses as <u>representation of objects</u>
3. HTTP is stateless and self descriptive (simple unified interface: GET, POST, PUT, DELETE, ...)
4. Embed links in response (i.e. present the options to the client, more to come ...)

RESTful CRUD Service

Resource URL:http://www.server.com/application/orders

URL above will give us a representation of the orders (possibly in XML (or other))

CRUD = create, read, update, delete



To Build an RESTful Application

I.e. an application adhering to the principles we use **Web Services** (and more...)

To use Web Services we need a few Java EE APIs, ... (and some design)
- **JAXB** (mostly in background)
- **JAX-RS**
- ... but first define Java Bean (aka Javabean, Java Beans, ...next slide)

Java Bean

Recurring term: **Java bean**, session bean, managed bean, enterprise bean...

Class must fulfill the below to be a Java Bean

- Private attributes and read/write methods for (relevant) attributes (attribute +set + get is called **a property**)
- Naming conventions for set/get-methods

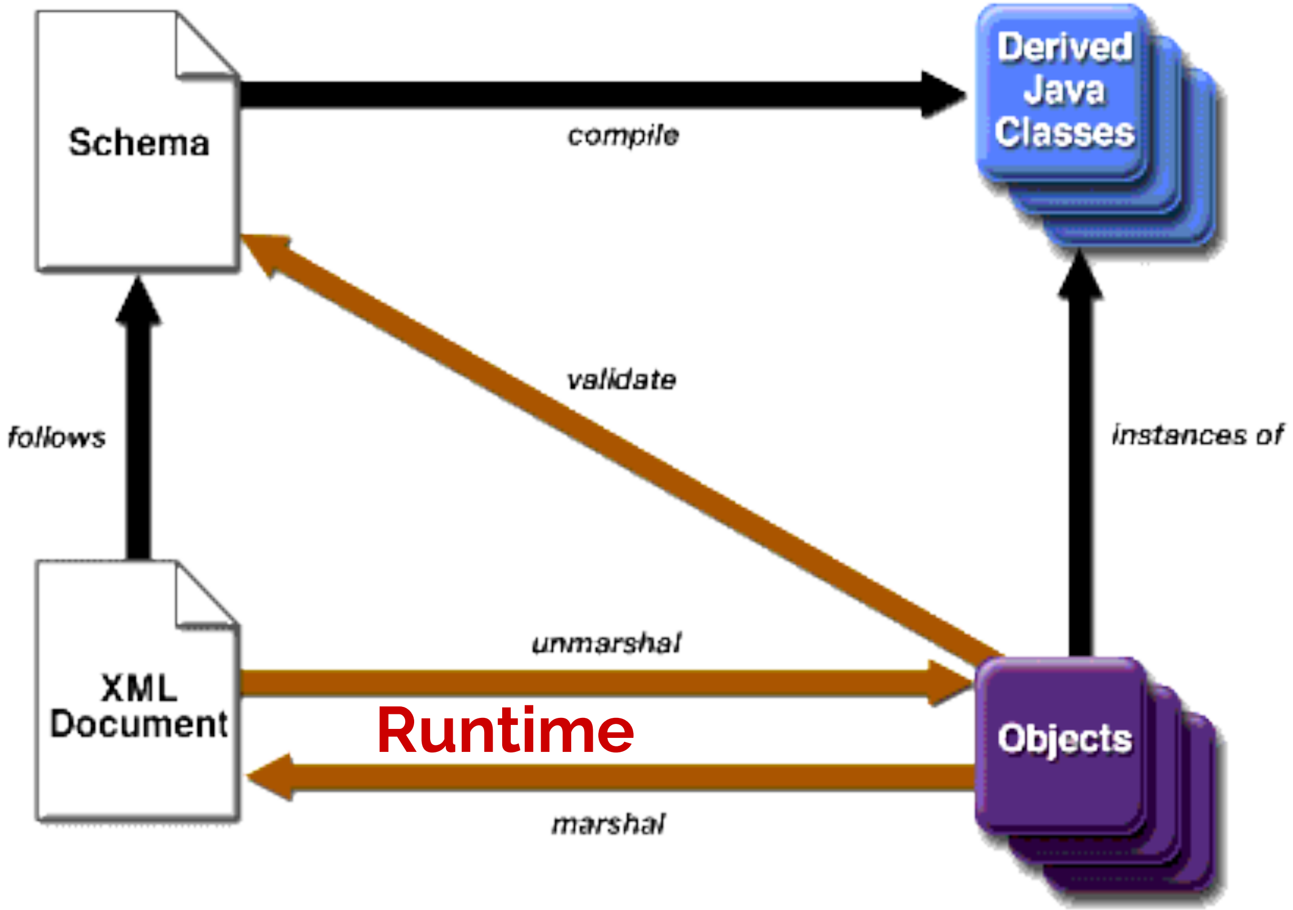
```
private String data;
public String getData();
public void setData( String str );
```

- Default constructor (possible protected), Serializable
- May generate events

.. sadly it's a bit of an antipattern... (can't use immutable...)

Java Architecture for XML Binding, JAXB

Purpose: Convert between XML Schema/XML documents and/or classes/objects



JAXB Basics

We will not use XML Schema from/to class

We use JAXB to **(un)marshal** objects

- Must have no-arg ctor (not shown below)

```
@XmlRootElement(name="person")
@XmlAccesorType(XmlAccessType.PROPERTY) //Annotation on methods (or XmlAccessType.
FIELD)
public class Person {

    private int id;
    private String fName;

    @XmlAttribute
    public void setId( int id ){
         this.id = id;
    }

    @XmlElement(name="fname")
    public void getFName(){
        return fName;
    }
}
```

JAXB Documentation

Reference implementation **Metro,** links course page

We'll try to avoid explicit use of JAXB, most work will be done in background, more to come...

Details at https://jaxb.java.net/2.2.6/docs/ch03.html#annotating-your-classes

Testing JAXB

JAXB part of Java SE, possible to run JUnit test without any dependencies!

- Use to check out the XML result of the annotations, see code samples

Web Services

Probably no commonly accepted definition ?!

[Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.] ***)**
// http://www.w3.org/TR/ws-arch/

***)** This is mostly a definition of WS-*, upcoming…

Web Services Programmers View

The application is composed of loosely coupled, distributed, reusable, platform/language independent services (resources)

Service has an agreed on/public interface/API

Presentation or functionality from two or more sources to create new services
- This is sometimes called a **mashup** application

Types of Web Services

WS-*, A stateless messaging service (Simple Object Access Protocol, SOAP), describing service interfaces in XML (Web Services Description Language, WSDL). Heavyweight. Code generation from WSDL and conversion to objects. WSDL example: http://www.vasttrafik.se/External_Services/TravelPlanner.asmx?WSDL

WS-REST, RESTful Web Service, an architectural style ....

Services vs. Resources

WS-* is a service oriented approach. The key abstraction is a **service** (a verb)

WS-REST, is not service oriented, it's resource-oriented, the key abstraction is a **resource** (a noun)

- So Web Service for REST is a bit misleading

WS-* vs. WS-REST

REST very hyped right now, but watch this ...

http://www.slideshare.net/pizak/rest-vs-ws-myths-facts-and-lies-352457

True believers in REST: RESTafarians

Anyway, ...we only use WS-REST

Web Services Roles

Consuming a Web Service, i.e a client
Producing, implement a Web Service

Many public Web Services available (normally need a key to send with requests, API-key or an account or …).

Example: Consuming some RESTful Services

Example: Flickr (photo service, no key)
http://api.flickr.com/services/feeds/photos_public.gne?tags=flower&lang=en-us&format=atom (try change format)

Example: YouTube (no key)
http://gdata.youtube.com/feeds/api/standardfeeds/most_viewed

Later examples using key (OAuth)

Java API for RESTful Web Services (JAX-RS)

Java Specification for REST, JSR 311

Reference implementation: **Jersey**

Client side and server side API's

Need configuration in web.xml (special Servlet...)

- NetBeans will notify if missing (light bulb)

REST Java Client Side

Of course possible to use the java.net.* directly

- Use **HttpURLConnection**... inconvenient, looooong strings
- ... have to convert representation

Generic client libraries: com.sun.jersey.api.client.Client

A bit smoother, working with objects... using a **WebResource** class

- ...but still cumbersome, many and strange parameters
- Possible to generate in NetBeans

Specialized API's, example: Twitter4J

- Converting REST API to Java OO API
- Comfortable, clean objects, parameters, ... prefer!

REST Non Java Client Side

Very common with JavaScript clients running directly in browser, upcoming

Also JavaScript libraries for Twitter and alike (more on JavaScript later)

REST Java Server Side

JAX-RS **resource classes**

- Class representing the resource
- Possible to map to URLs
- Can handle HTTP requests (similar Servlet doGet(), doPost())
- Often automatic conversion of representation in parameters and results
- Automatic extraction of parameters from URL
- Possible a hierarchy of resource classes matching different parts of URI

If a resource class in application, NetBean will add special icon in project (RESTful Web Services). No files only resources shown

JAX-RS Root Resource Class

The top level resource class (there are sub resources not used by us...)

Root resource class must have

- @Path class annotation
- Or at least one method with @Path or a request method designator (= annotation on method): @GET, @POST, @PUT, @DELETE,...a **resource method**
- Default ctor

Instantiated by the JAX-RS runtime

Content Negotiation (Conneg)

Different client needs different representation

- XML, JSON (text format), YAML,...
- Internationalization, Encoding,...

Which representation are we using?

Resource methods can handle different MIME types.
Specify with annotations

- @Consumes (mime type(s) in request...)
- @Produces (...in response)
- Must match "Accept" HTTP header else "Not Found"

Root Resource Example

A first root resource class

```
@Path("/persons")
public class PersonResource {  // Root resource,any Java class
    ...

    @GET
    //@Consumes often not needed more later
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Person> selectAll() {
    ...   // Will return List<Person> as XML  (JAXB in background)
    }

}
```

Person must be JAXB marsh:able

Running a Service

Possible on Tomcat but many dependencies

GlassFish works out of the box (possible dependency see samples)

Important log message from GlassFish when running (check!)

```
INFO: Root resource classes found:                         // Good!
  class edu.chl.hajo.bjj.resource.PersonResource
INFO: No provider classes found.                           // No problem!
```

Testing a Service

There are testing frameworks (Jersey Test Framework) but too complex for now, avoid

Simpler (but manual): **cURL**, command line tool for transferring data with URI syntax http://curl.haxx.se/

Command line post example

```
curl -v  http://localhost:8080/service_based/rs/persons --request
POST  --data "pnumb=99&fname=XX&age=99"
```

Conneg Details

Any JAXB annotated class can be marshaled as response

- As XML or mapped to JSON, Atom, ...

Can't return primitives int, double, boolean, ... !

- ... has no natural XML representation (possible to implement custom content handlers to marshal, we don't)...
- ... simpler, create a JAXB wrapper class (<boolean>true</boolean>)...
- ..or return a String (too simple?)

JAX-RS: URI Path Templates

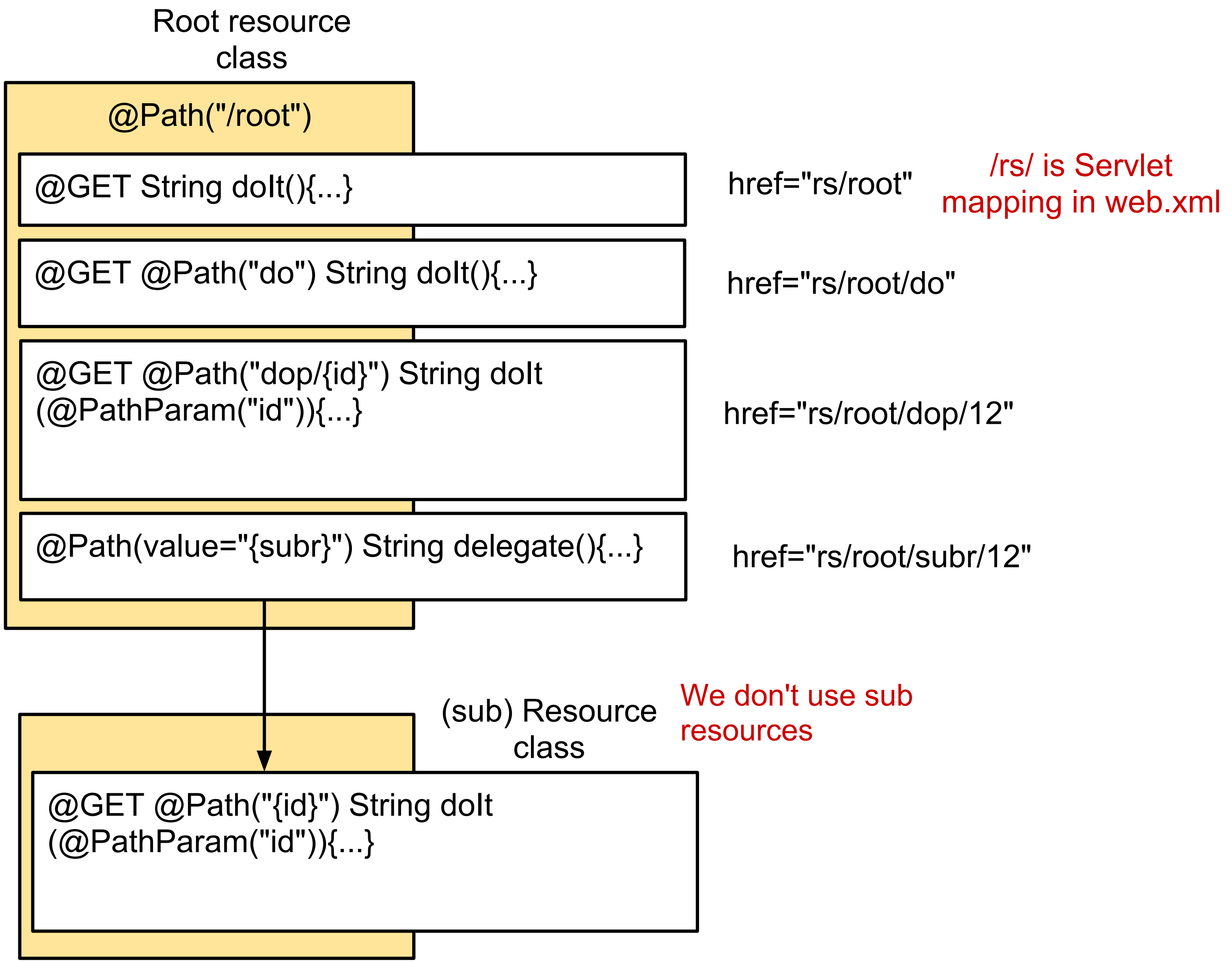
URI's with embedded parameters (substituted runtime). Annotations on methods

```
// username a parameter in request
@Path("/users/{username}")
// Possible using regex to match
@Path("users/{username: [a-zA-Z][a-zA-Z_0-9]}")
```

Possible to retrieve request parameter as path (method) parameter

```
@GET
@Produces("text/xml")
@Path("/users/{username}")
public String getUser(@PathParam("username")
                 String userName) {
     ...
}
```

URL Path Matching



Root resource class

@Path("/root")

@GET String doIt(){...}

href="rs/root"

/rs/ is Servlet mapping in web.xml

@GET @Path("do") String doIt(){...}

href="rs/root/do"

@GET @Path("dop/{id}") String doIt (@PathParam("id")){...}

href="rs/root/dop/12"

@Path(value="{subr}") String delegate(){...}

href="rs/root/subr/12"

(sub) Resource class

We don't use sub resources

@GET @Path("{id}") String doIt (@PathParam("id")){...}

Form Parameters

@FormParam, parameter annotation, extract posted form data (matching names)

```
@POST
@Consumes("application/x-www-form-urlencoded")
public Response post(@FormParam("pnumb") String pnumb,
        @FormParam("fname") String fname,
        @FormParam("age") int age) {
    ...
}
```

Type Conversions

Automatic type conversion from HTTP request (string) to Java

- Primitive types
- Some classes with special restrictions (ctor with exactly one String param, etc.)
- List<T>, Set<T> and SortedSet<T>

Conversion of Generic Types

Must use

```
// Get a list
List<Person> ps = ... (List is generic class)

// Wrap list (note anonymous subclass)
GenericEntity<List<Person>> ge =
    new GenericEntity<List<Person>>(ps) {};
```

More later...

JAX-RS Content Handlers

Some build in content handlers ((un)marshall from/to HTTP message body to specific Java type)

Mostly low level

- StreamingOutput, Reader, File, byte[], String, char[], ...
- Possible to implement custom content handlers (primitive types)
- Probably don't need

Context

Possible to inject "low level" objects in resource classes using @Context annotation (similar to ServletContext)

```
@Context
private HttpHeaders headers;

@Context
private UriInfo uriInfo

@Context
private Request request;
```

.. and more

Standard HTTP Response Codes

Successful response codes (GET, POST, PUT, DELETE)

- "200 OK", if return value not null, message has body
- "204 No Content" if return value null (but ok), no message body
- ...

Errors

- Client error 4xx
- Server Error 5xx
- Examples "404 Not Found", "406 Not Acceptable", wrong data format, "405 Method Not Allowed", bad method, 500 Internal Server Error (...NullPointerException ..:-) possible...)

http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

REST Response Codes

JAX-RS default response codes close to standard response codes (as described in HTTP 1.1)

Examples
GET: If found 200, else 204 (null)
POST: 201 Created

Possible to customize response codes, upcoming...

Return Types

As noted we can return objects or collections of any JAXB marsh:able type

- But we don't ...

Simpler and more uniform to <u>let all methods have return type </u>: javax.ws.rs.core.**Response**

- Possible to embed (marshalled) objects in responses
- Possible to customize for example response codes and more...
- Will use response codes from previous slide
- Inspect code samples

Return Type Examples

```
// There's a ResponseBuilder object in background
Person p = reg.selectByPk(pnumb);
if (p != null) {
   // ok = 200 OK
   return Response.ok(p).build();
} else {
   // noContent = 204 No Content
   return Response.noContent().build();
}

// Returning a collection
List<Person> ps = reg.selectAll();
GenericEntity<List<Person>> ge =
   new GenericEntity<List<Person>>(ps) {};
       return Response.ok(ge).build();
```

Caching

"The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions."// Roy Fielding

I.e.

- improve speed, because we want to deliver fast content to our consumer
- fault tolerance, because we want our service to deliver content also when it encounters internal failures
- scalability, because the WWW scales to billions of consumers through hypermedia documents and we just want to do the same thing
- reduce server load, because we don't want our servers to compute without the need of it

Types of Cache

Local cache, your browser's local copy

Proxy cache, a copy on some server on the way to the origin (the original Server), a middleman

- Content Delivery Network (CDN), large distributed system of servers deployed in multiple data centers in the Internet. The goal of a CDN is to serve content to end-users with high availability and high performance (example: Akamai)

Caching Strategy

Good candidates for caching are pages that:

- Are accessed frequently
- Are stable for a period of time
- Contain a majority of contents that can be reused by a variety of users

A good example would be catalog display pages

Pages with sensitive data shouldn't be cached

HTTP Header: Cache-Control

Cache-Control (HTTP 1.1) some parameters. Server says...

Value	**Description**
private	A cache mechanism may cache this page in a Private cache and resend it only to a single client. This is the default value. Most proxy servers will not cache pages with this setting.
public	Shared caches, such as proxy servers, will cache pages with this setting. The cached page can be sent to any user.
no-cache	Do not cache this page at all, even if for use by the same client.
no-store	The response and the request that created it must not be stored on any cache, whether shared or private. The storage inferred here is non-volatile storage, such as tape backups. This is not an infallible security measure.

Example Server Response

```
HTTP/1.1 200 OK
Content-type: application/xml
Cache-Control: private, no-store, max-age=300
```

- Only client may cache
- Must not be stored on disk
- Valid for 300 seconds

Cache Inconsistency

Cache introduces inconsistency!

- Possible resource served to a consumer is different from the one actually held by the server

To improve data's consistency:

- Cache validation
    - Use ETag header (a hash/MD5 encoding of the object). Server calculate/set ETag header, client store and add to next request or ...
    - ... use Last-Modified header, timestamp, proceed as above...

Cache validation

```
public interface Request {
  ResponseBuilder.evaluatePreconditions( ETag eTag)
  ResponseBuilder.evaluatePreconditions( Date lastModified)
  ResponseBuilder.evaluatePreconditions( Date lastModified, ETag
eTag)
}
```

Use on Server side

```
// Create an ETag object (etag) for actual object then...(compare with request)
ResponseBuilder builder = request.evaluatePreconditions(etag);
```

If precondition **true** builder == null

If precondition false builder == A ResponseBuilder with appropriate status (possible 412 Precondition failed)

HATEOAS

HATEOAS: The interaction state of the client/server in exchanged hypermedia

I.e embed all links useful for client in the response

Using Atom-links (XML based format for lists of related information)

- Links embedded in XML document (or other) or ...

Using HTTP Response “link” header

- Link in response (don't need to parse XML to get the link)

JEE HATEOAS

Not much support in JAX-RS (HATEOAS defined by application)

Running JEE Web Services

Possible to use Tomcat or other but many, many dependencies

Prefer GlassFish everything included