REST for Analysts

DAY 3

Course Agenda

Day 1 – Service Fundamentals

- *HTTP Primer*
- *XML Primer*
- *XML/HTTP Services*
- *Service Orientation*

Day 2 – Defining REST APIs

- *Understanding Services*
- *RESTful Architecture Components*
- *RESTful Analysis & Design*

Day 3 – Building REST APIs

- **JSON Data Format**
- **Building a RESTful Service**
- **JAX-RS Services**
- **Crafting a Service from Scratch**

Day 4 – Analyzing & Testing REST Services

- Browser Tools & Extensions
- Testing Services
- Pulling it all Together

Building REST APIs

GOAL: LEARN THE BASICS OF REST DEVELOPMENT

Section 1

JSON DATA FORMAT

JSON – Native web format

Javascript Object Notation (JSON)

- Cut out an additional translation step
- Efficiently exchange data
- Suitable for message exchanges as well as persistence
- Simple, elegant

JSON Data Types

Strings

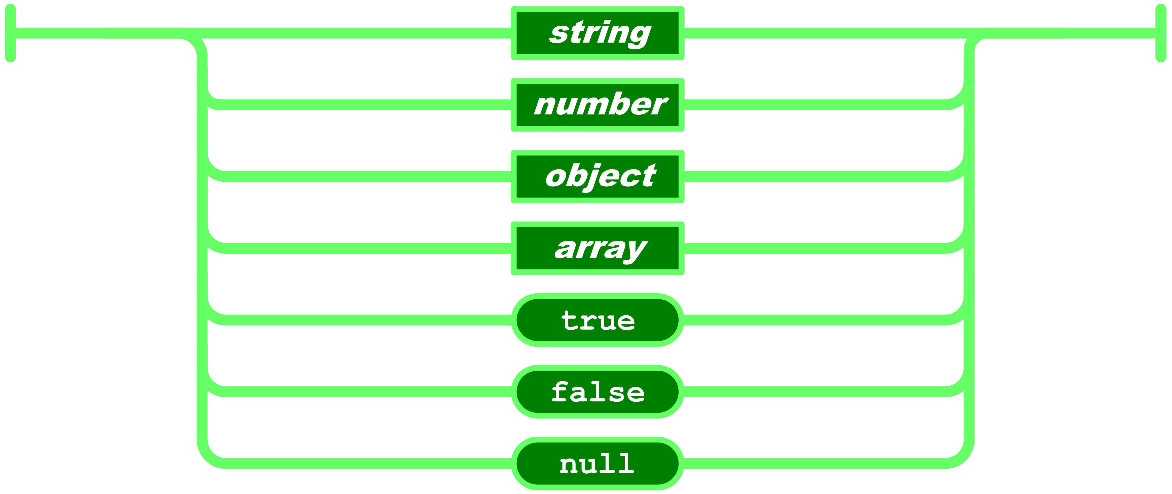
Numbers

Booleans

Objects

Arrays

`null`



Strings

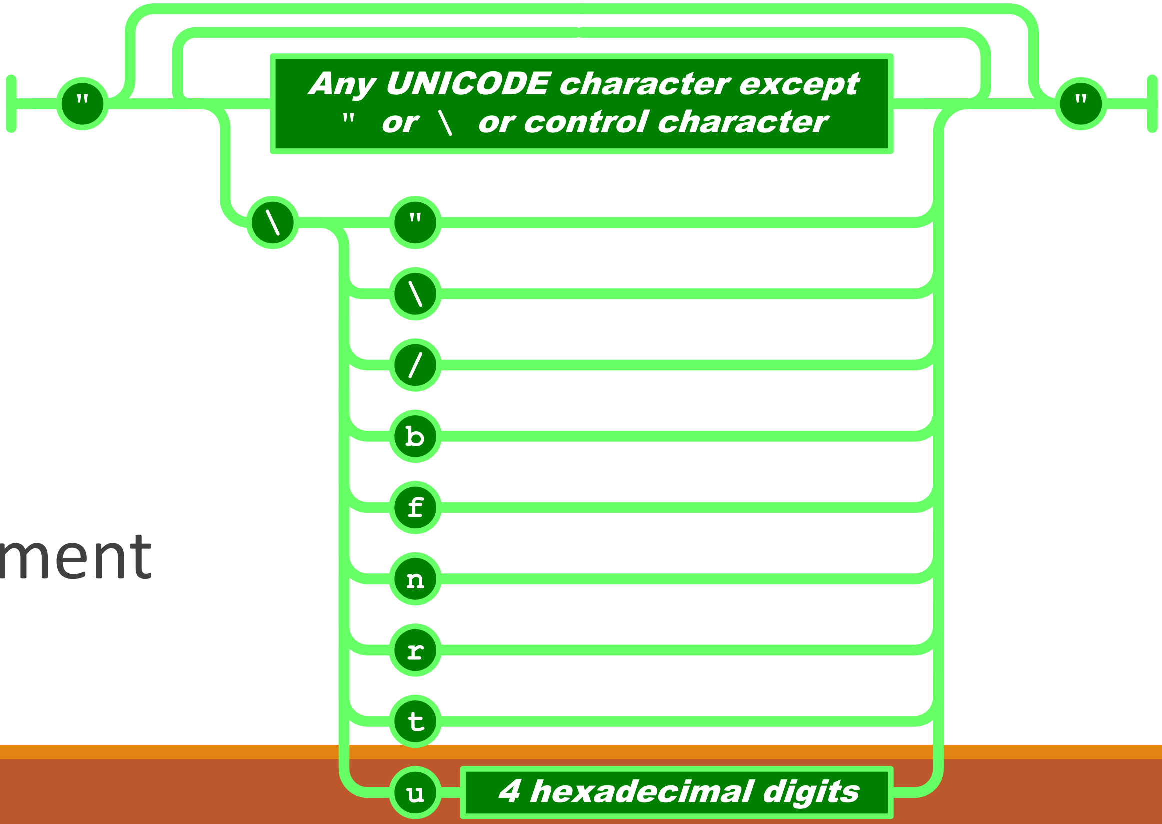
Sequence of 0 or more Unicode characters

No separate character type

- A character is represented as a string with a length of 1

Wrapped in "double quotes"

Backslash escapement



Numbers

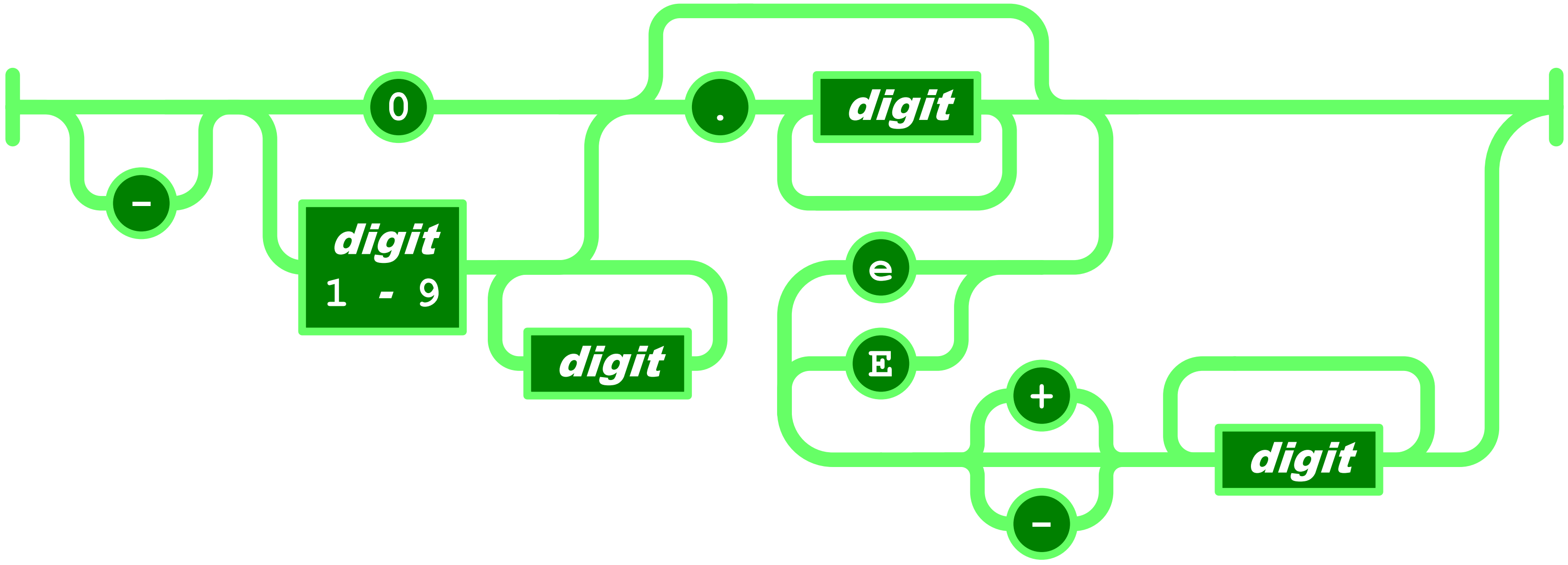
Integer

Real

Scientific

No octal or hex

No **`NaN`** or **`Infinity`** *(Use **`null`** instead)*



Other Literals

Boolean

- True
- False

Null

- A placeholder which represents nothing

Objects

Objects are unordered containers of key/value pairs

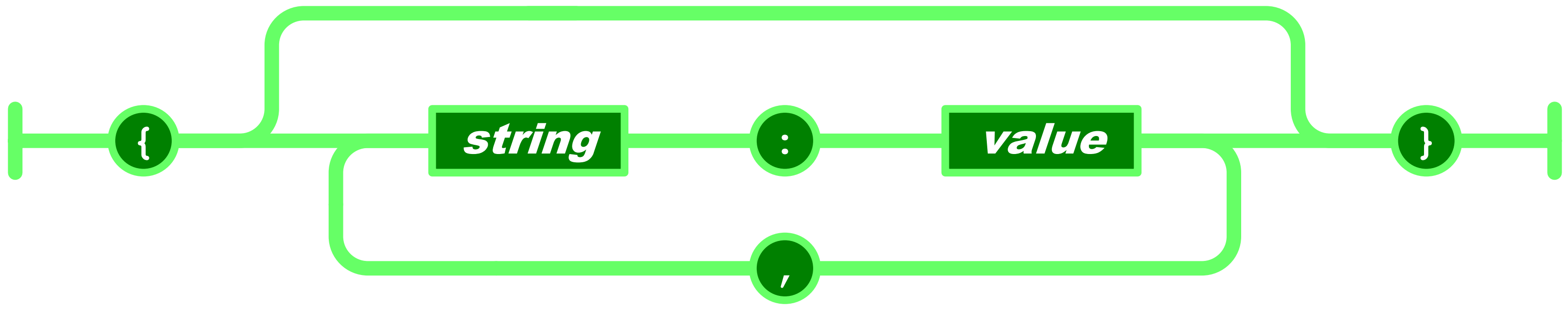
Objects are wrapped in **{ }**

, separates key/value pairs

: separates keys and values

Keys are strings

Values are JSON values: struct, record, hashtable, object



Object Sample (collapsed)

```
{"name":"Jack B. Nimble","at large":
true,"grade":"A","level":3,
"format":{"type":"rect","width":1920,
"height":1080,"interlace":false,
"framerate":24}}
```

Object Sample (formatted)

```
{
 "name": "Jack B. Nimble",
 "at large": true,
 "grade": "A",
 "level": 3,
 "format": {
      "type": "rect",
      "width": 1920,
      "height": 1080,
      "interlace": false,
      "framerate": 24
      }
}
```

Array

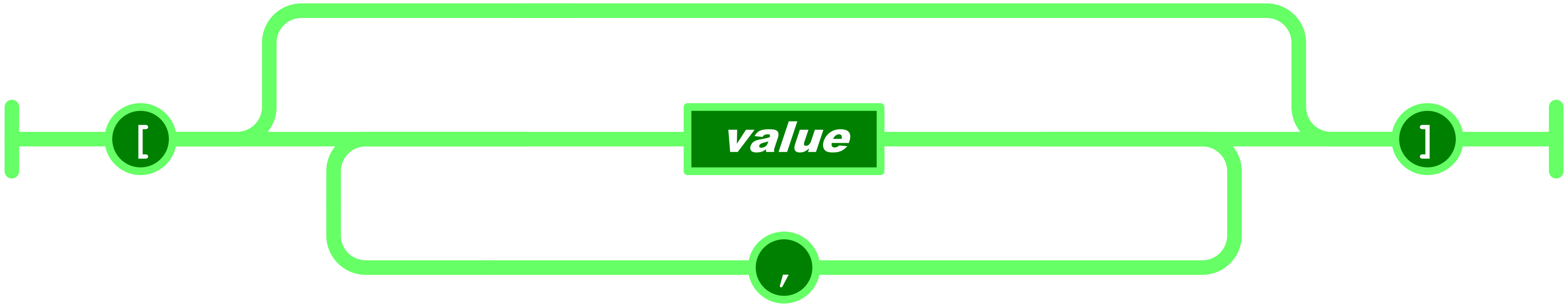
Arrays are ordered sequences of values

Arrays are wrapped in `[]`

Comma(,) separates values

JSON does not talk about indexing.

- An implementation can start array indexing at 0 or 1.



Array Sample

```
["Sunday", "Monday", "Tuesday",
  "Wednesday", "Thursday", "Friday",
  "Saturday"]


[
   [0, -1, 0],
   [1, 0, 0],
   [0, 0, 1]
]
```

Arrays vs Objects

Use objects when the key names are arbitrary strings.

Use arrays when the key names are sequential integers.

Don't get confused by the term Associative Array.

Using JSON

HTML Delivery.

JSON data is built into the page.

```
<html>...
 <script>
  var data = { ... JSONdata ... };
 </script>...
</html>
```

JSON Looks like Data

JSON's simple values are the same as used in programming languages.

No restructuring is required: JSON's structures look like conventional programming language structures.

JSON's object is record, struct, object, dictionary, hash, associate array...

JSON's array is array, vector, sequence, list...

Lab 11 – Service Data Formats

1. Launch your browser
2. Revisit the JSON data format example from yesterday **http://maps.googleapis.com/maps/api/geocode/json?address=Seattle+WA**
3. Get back in your groups and select one service from your project example.
4. Define the JSON data structure for the request and the response.

Section 2

BUILDING A RESTFUL SERVICE

Service Dev Checklist

1. Service Design
   - Business Model (process flow, capabilities diagram, etc.)
   - Data Model (entity catalog, resource diagram, etc.)
   - Application Model (service layering diagram, service composition diagram, service/system matrix, service operation catalog, etc.)
   - Technical Artifacts (service interface, schema, etc.)
   - Governance Guidance (taxonomy, URI pattern, data dictionary, etc.)

2. RESTful Service Development
   - Resource selection
   - URI definition
   - Service implementation (source code, config files, libraries / dependencies)

Defining the Resource & URI

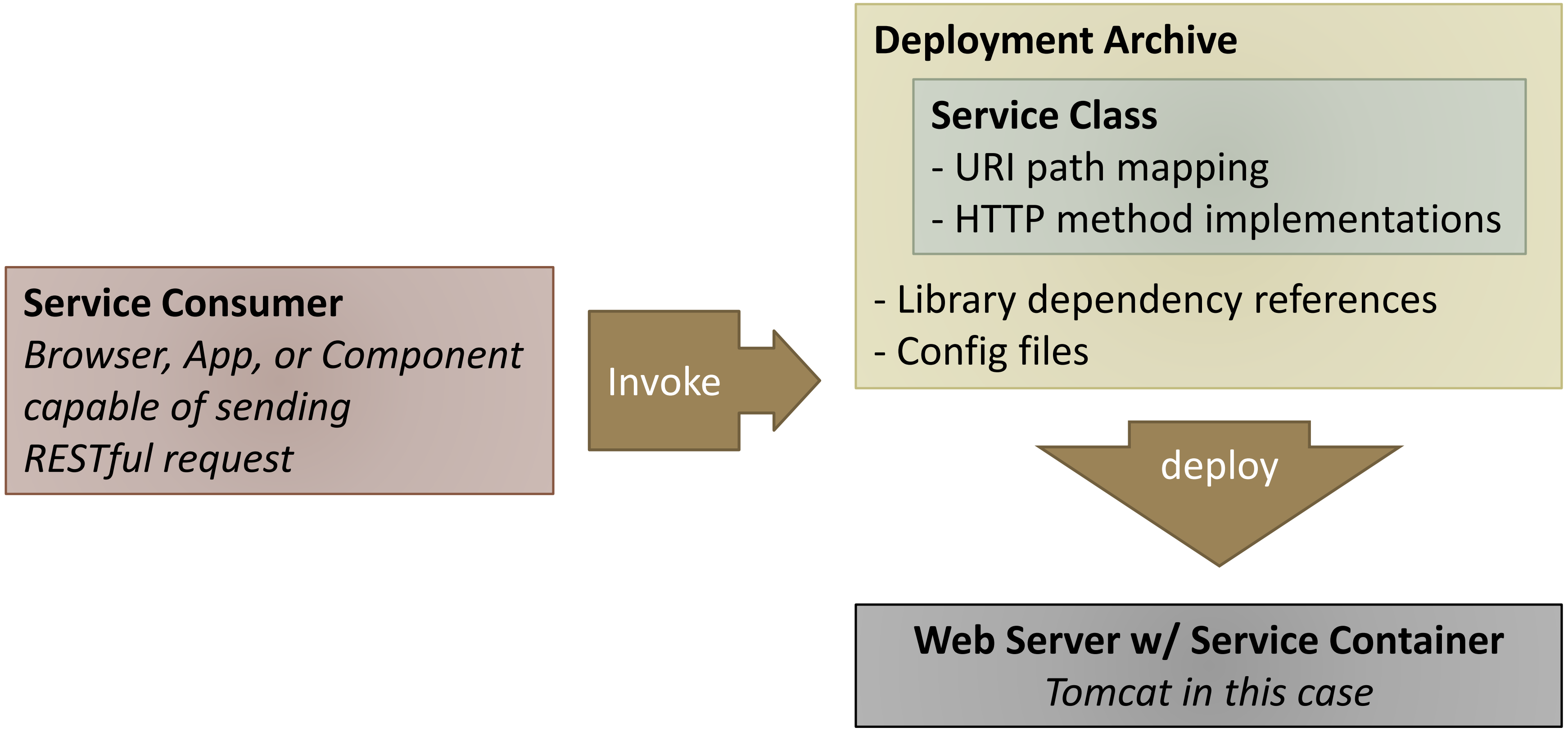
What resource(s) and collections will your service use?

- Should you define one or two services with deeply nested resource trees?
- Should you be better served by a flatter model with more resources and fewer sub-resource collections?

How will the URI be structured?

- Guidance from governance documentation may be helpful.
- Define "path" to your resource (could be multiple ways to "get" to the same data).
- What portions of the path will be static and what portions will be dynamic?

Service Implementation


Service Consumer
Browser, App, or Component capable of sending RESTful request
Invoke
Deployment Archive
Service Class
- URI path mapping
- HTTP method implementations
- Library dependency references
- Config files
deploy
Web Server w/ Service Container
Tomcat in this case

Lab 12 – Hello Cliché!

Open the ***Lab12*** folder.

Follow the instructions in the PDF you find there.

Prepare to discuss your findings with the class.

Section 3

JAX-RS SERVICES

JAX-RS API

Java API designed to ease the development of service based on REST architecture.

Native part of Java EE framework.

Supports building and deployment of RESTful web services.

Annotations can be used to simplify the development.

JAX-RS annotations are processed at runtime.

JAX-RS Implementations

Apache CXF – *open source*

Jersey – *RI from Sun (now Oracle)*

RESTeasy – *JBoss*

Restlet – *open source*

WebSphere Application Server – *IBM*

WebLogic Application Server – *Oracle*

A couple others…

JAX-RS Client & Service API

Client API

- Build a standalone client application to invoke a service
- Configure HTTP settings (method, content type, other request headers)
- Process the response (evaluate the status code, retrieve response headers, display results)

Service API

- Configure path settings for service and individual methods
- Configure parameters as portions of the URI path
- Map HTTP methods to Java methods
- Specify MIME type inputs / outputs by service method

***Note*: The Client and Service APIs can be used in combination or separately.**

Core Annotations

@Path – specifies the relative path for a resource class or method.

@GET, **@PUT**, **@POST**, and **@DELETE** – specify the HTTP request type of a resource.

@Produces – specifies the response Internet media types (used for content negotiation).

@Consumes – specifies the accepted request Internet media types (useful for deciding between two methods mapped to the same request type).

Sample Code

```
@Path(“catalog")
@GET
@Produces("application/xml")
public String searchCatalog( ) {
        // Pull info from catalog
        …
        return resultObject;
}
```

Parameter Annotations

@PathParam – binds the method parameter to a path segment.

@QueryParam – binds the method parameter to the value of an HTTP query parameter.

@HeaderParam – binds the method parameter to an HTTP header value.

@CookieParam – binds the method parameter to a cookie value.

@FormParam – binds the method parameter to a form value.

@DefaultValue – specifies a default value for the above bindings when the key is not found.

@Context – returns the entire context of the object (*for example @Context HttpServletRequest request*).

Parameter Sample Code

```
@Path(“user/{id}")
@POST
@Consumes(“text/json")
public String createUserFromJson(@PathParam(“id") String id) {
            …
            return newUserData;
 }


@Path(“user/{id}")
@POST
@Consumes(“application/x-www-form-urlencoded")
public String createUserFromForm(@PathParam(“id") String id) {
            …
            return newUserData;
 }
```

Lab 13 – Arithmetic Service

Open the ***Lab13*** folder.

Follow the instructions in the PDF you find there.

Prepare to discuss your findings with the class.

Building a JAX-RS Client App

There are several key classes in the JAX-RS Client API

`Client` – *main class for creating a request*

`WebResource` – *manages HTTP request details*

- `get()` for GET requests
- `post()` for POST requests

`ClientResponse` – *manages HTTP response details*

Client GET Example

```
Client client = Client.create();
String getUrl = "http://localhost:8080/Calculator/calc/tempconvert/c2f/32";
WebResource webResource = client.resource(getUrl);
ClientResponse response =
  webResource.accept("application/json").get(ClientResponse.class);

if( response.getStatus() != 200 ){
		throw new RuntimeException("HTTP Error: "+ response.getStatus());
}
String result = response.getEntity(String.class);
System.out.println(result);
```

Client POST Example

```
Client client = Client.create();
String postUrl = "http://localhost:8080/AddressBook/contact";
WebResource webResource = client.resource(postUrl);
String inputData = “{\”name\”:\”Bob\”,\”age\”:42}”;
ClientResponse response =
   webResource.accept("application/json").post(ClientResponse.class, inputData);

if( response.getStatus() != 201 ){
	throw new RuntimeException("HTTP Error: "+ response.getStatus());
}
String result = response.getEntity(String.class);
System.out.println(result);
```

Lab 14 – REST Client

Open the ***Lab14*** folder.

Follow the instructions in the PDF you find there.

Prepare to discuss your findings with the class.

Section 4

CRAFTING A SERVICE FROM SCRATCH

Service Development Checklist

1. **Service Design**
   - ❑ Models and artifacts
2. **RESTful Service Definition**
   - ❑ Resource selection
   - ❑ URI definition
   - ❑ Method selection (per URI)
3. **Service implementation**
   - ❑ Java class with functionality in one or more methods
   - ❑ `@Path` annotation on class and each method
   - ❑ `@GET` | `@POST` | `@PUT` | `@DELETE`
   - ❑ Define `@Param` (as needed) and adjust paths accordingly
   - ❑ Define `@Produces` and `@Consumes` (as needed)

New Project Steps

- ❑ New *Dynamic Web Project*
- ❑ Convert project to Maven
- ❑ Configure dependencies
- ❑ Create a Java class for the service
- ❑ Add in annotations
- ❑ Configure container via web.xml
- ❑ Deploy project to server
- ❑ Test service

Service Development

Same groups of 3-4 people

Have one or two people use their computer to develop a service within the context of your project scenario

Other team members should guide, observe, support, and provide peer review to those doing the computer work.

Steps (use prior two slides to guide this activity)

- Create a new project
- Build service
- Test it

daily retrospective

Today’s Goal

learn the basics of REST development

Do you feel like we’ve accomplished that goal today?

What aspect of today’s training most resonated with you?

Good and Better If

What was **good** that you would like to see more of tomorrow?

What could have been **better if** we had done it differently?

GOOD	**BETTER IF**