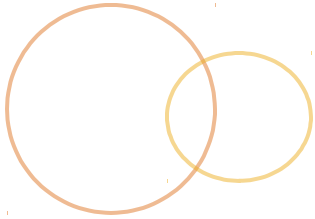
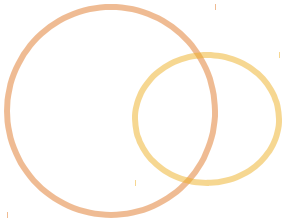
Getting Started With JAX-RS





Objectives

- State where to obtain the latest version of TCRS
- Build and run a provided version of the Intuit TCRS “starting point” project
- Import the TCRS project into Eclipse so that it can be edited, built, and run in that IDE
- Use the @Path annotation to define a “Root Resource” in a JAX-RS project
- Use @Path with a literal value, and @GET to cause JAX-RS to invoke a programmer-provided service method in response to an appropriate HTTP request

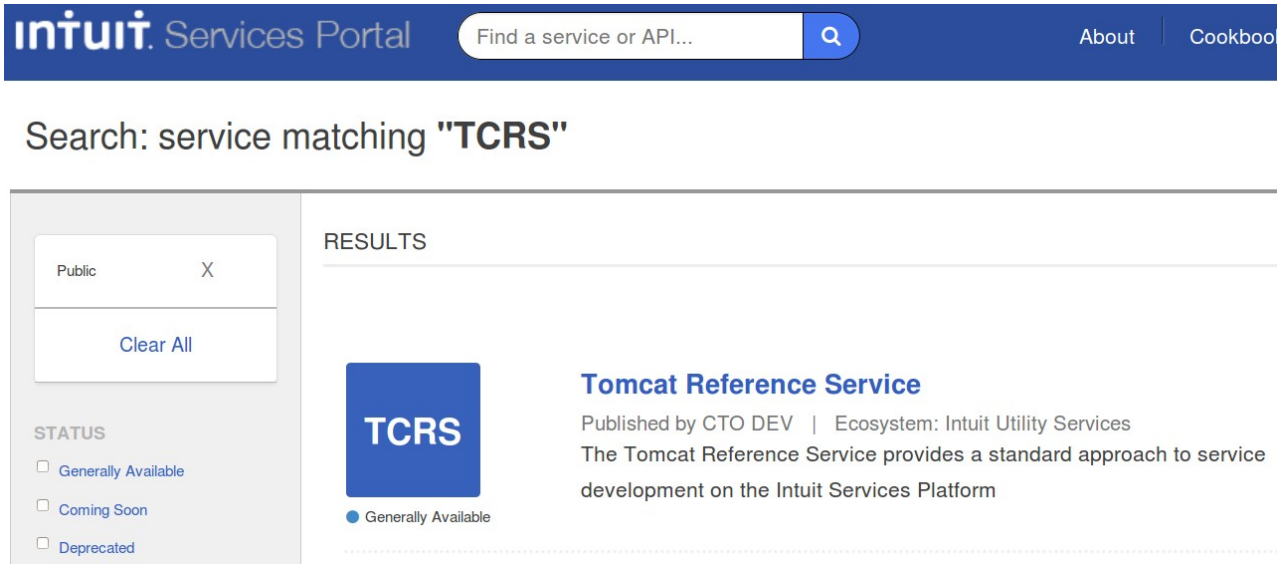
Objectives

- Modify the TCRS to add new root resources to the service
- Describe the meaning of the term “Provider” in JAX-RS
- State two annotations that can be used to make a class a JAX-RS Provider
- Use the @ApplicationPath annotation to modify the base URI to which a JAX-RS application responds
- Give an overview of two alternative methods by which JAX-RS can be configured, including how to find specific details for these methods in both Jersey and RESTeasy implementations of JAX-RS

Obtaining TCRS

- Go to: devinternal.intuit.com, search for TCRS

- Locate project page



Obtaining TCRS

- Read the Getting Started and Developer's Guide pages
  - Ensure right software / versions installed
  - Ensure environment properly configured
  - Download or clone the project
  - Build with Maven (follow instructions!)
  - Test launch with Maven
  - Import into Eclipse

JAX-RS Root Resource

- JAX-RS service starts by matching a “Root Resource”
  - Class annotated with @Path(“/some-path”)

```
@Path(“/customers”)
public class CustomersRootResource {
```

  - Can respond to requests with URIs starting with /customers
  - This is the first glimpse of JAX-RS “routing”, which selects business logic code to service a request

Java EE Context Root

- Java web containers (e.g. Tomcat) use the first part of the URL to identify a particular web application.

```
http://myhost.intuit.com:8080/my-service/customers
```

- First part leads request to the web container
- Second part leads to a particular web application
- Third part forms the “root” of the URI seen by our web application in JAX-RS

JAX-RS Service Method

- To handle a request (or “route” the request)
  - Path must be matched
  - HTTP method must be matched
  - (Other specified criteria might need to be matched)

```
@Path(“/customers”)
public class CustomersRootResource {
  @GET // matches HTTP GET method
  public String getCustomers() {
    // Matches GET /customers
    // Returns “all customers”
```

JAX-RS Service Method

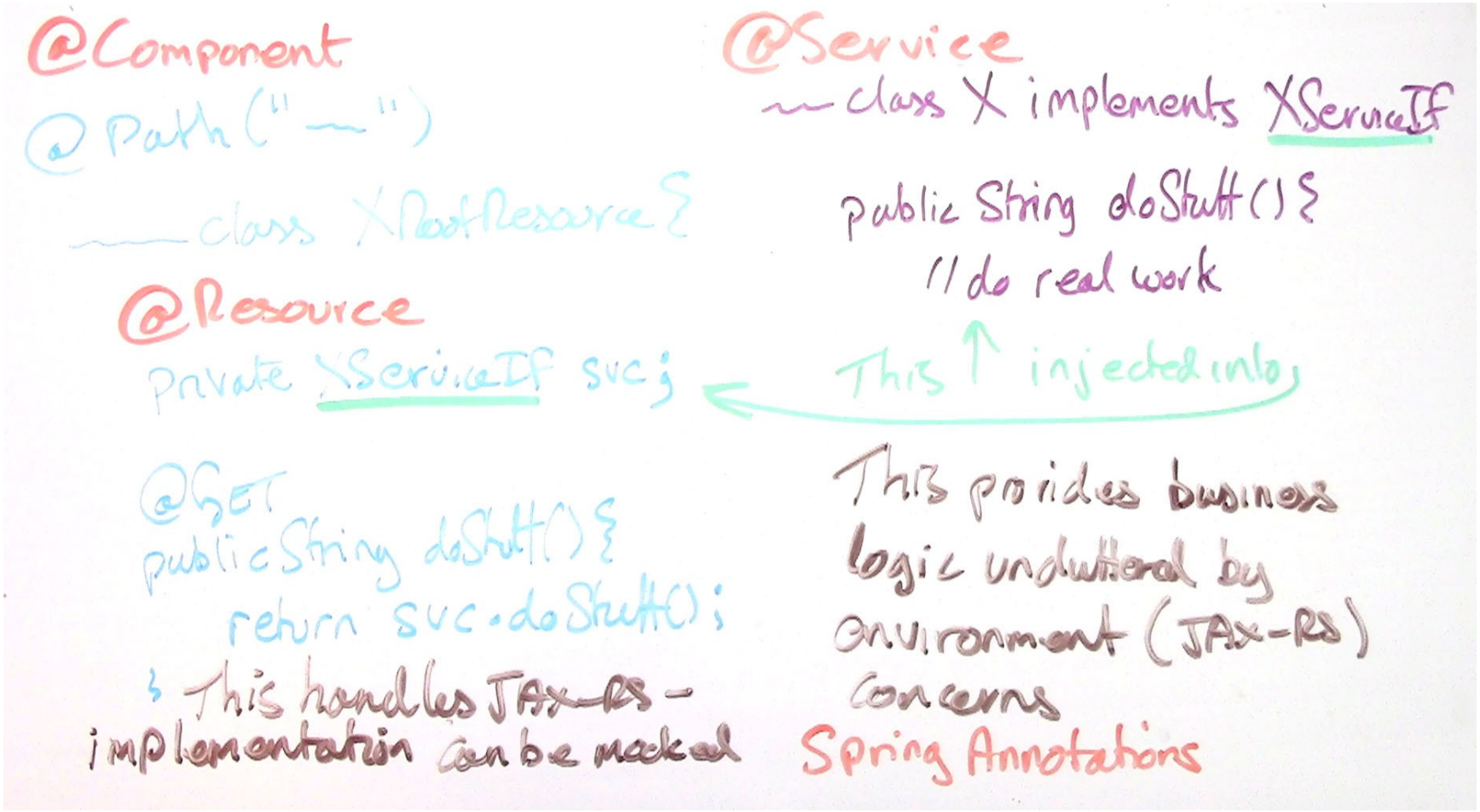
- Routing can happen on the method
  - This matches /customers/1234 in two steps

```
@Path(“/customers”)
public class CustomersRootResource {
  @Path(“/1234”)
  @GET // matches HTTP GET method
  public String getCustomer1234() {
    // Matches GET /customers/1234
    // Returns customer 1234
```

TCRS Structure

- TCRS is built using the Spring Framework
- Basic configuration of JAX-RS implementation (Jersey) is done
- Spring separates business logic from container interface
  - JAX-RS root resource is annotated @Component
  - Implementation logic is annotated @Service
  - Implementation is injected into root resource using @Resource

Spring Structure of TCRS



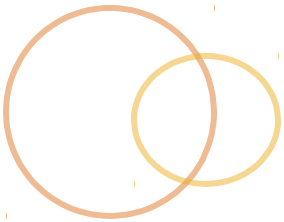
Adding to TCRS

- To add a new resource to TCRS:
  - Define a root resource
  - Define a business logic interface
  - Implement the business logic interface in a business logic class
  - Delegate from root resource to the business logic class for each service method

The Root Resource in TCRS

- The root resource class should:
  - Be located in the package `com.intuit.platform.sp.rest.resources`
  - Carry JAX-RS annotations (@Path, @GET etc.)
  - Be annotated @Component
  - Define a member of the business logic interface type
  - Annotate this member @Resource
  - Use this variable to delegate from root resource to business logic implementation

The Business Logic



- The business logic interface should
  - Be located in the package `com.intuit.platform.sp.rest.services`
  - Not be annotated for Spring or JAX-RS
  - Have ***no*** dependencies on JAX-RS
- The business logic implementation should
  - Be annotated @Service
  - Have ***no*** dependencies on JAX-RS
  - Implement the business logic interface

What is Spring Hiding?

- Spring is used primarily as an injection framework, facilitating unit testing with mocking
- Because it’s largely preconfigured Spring / TCRS hides the general configuration of JAX-RS implementations.
- If you work on projects that pre-date TCRS, you need to have an idea of where to look

General JAX-RS Configuration

- JAX-RS is a standard, implementations are available from multiple sources
- Plan to read the product documentation!
  - Jersey
    - The reference implementation
    - Used by TCRS
    - http://jersey.java.net/
  - REST Easy
    - Jboss project
    - Used in many existing Intuit services
    - http://resteasy.jboss.org/

General JAX-RS Configuration

- JAX-RS implementations offer many configuration options
- Container or not?
  - Run your application in a Java EE web container (such as Tomcat)
  - Run your application stand alone
- Intuit runs JAX-RS almost exclusively inside a container
  - “Grizzly” standalone mode is an option

From Request to Service Code

- Recall that for Java EE web-apps, the base URL must reach our web-container
- Next, the URL must match the “context root”

```
http://myhost.intuit.com:8080/my-service/customers
```

- First part leads request to the web container
- Second part leads to a particular web application
- Third part forms the “root” of the URI seen by our web application in JAX-RS (but not necessarily the root resources)

From Request to Service Code

- Java web-apps are built from servlets, and/or servlet filters
- Once inside our web-app, routing can follow instructions in web.xml, or use annotation-based configuration, to find the JAX-RS implementation, which is usually a servlet, but might be a filter
- Product documentation will tell you what you need to know; web.xml is easier to see what's going on

From Request to Service Code

- When servlets or filters are deployed, it's possible to define the sub-path that "triggers" them

…blah.com:8080/my-service/servlet-prefix/customers

- First part leads request to the web container
- Second part leads to a particular web application
- Third part is the mapping of the servlet / filter
  - This might be multiple-levels
- Fourth part leads to a root resource

From Request to Service Code

- Once in the JAX-RS implementation, JAX-RS must be able to find our root resources and other elements we wrote
- Classes, provided by us, that must be found directly by JAX-RS, are called providers. E.g.:
  - Root resources, annotated with @Path on the class
  - Generalized error handlers.
  - Utilities for converting between Java objects and wire formats such as JSON or XML
  - Unless there's a specific annotation (such as @Path) these are annotated @Provder

From Request to Service Code

- Providers can be located by JAX-RS in three common ways
  - Explicit listing in the return of a specific method in a specific class
  - Explicit listing in an <init-param> in web.xml
  - Package scanning in packages located in the web-app, based on packages listed in an <init-param> in web.xml
- The first approach is standard, the other two are generally more convenient, but are implementation dependent
  - Further implementation specific approaches may exist

The Application Class

- The one standard method of locating classes is to provide a subclass of `javax.ws.rs.core.Application`
- The class is annotated @ApplicationPath()
- The class might be declared in an <init-param> for the JAX-RS implementation servlet

```
<init-param>
  <param-name>javax.ws.rs.Application</param-name>
  <param-value>org.foo.MyApplication</param-value>
</init-param>
```

The Application Class

- When used, the Application class defines two methods

```
Set<Class<?>> getClasses()
Set<Object> getSingletons()
```

- The classes, and instances, returned by these will be used in the application

The ApplicationPath

- The Application classes, when used, is annotated @ApplicationPath
- This annotation takes an argument which might modify the “base” URI from which service is offered
  - Exact rules are quite complex, and depend on how the deployment is configured; refer to documentation, or just experiment

The ApplicationPath

`….com/my-svc/map/application-path/customers`

- First part leads request to the web container
- Second part leads to a particular web application
- Third part is the mapping of the servlet / filter
- Fourth part is the application path
- Fifth part leads to a root resource

Lab Exercise

- Using the provided TCRS snapshot:
- Build, run, and exercise the software according to the instructions on the TCRS Getting Started page
- Create a new root resource in your TCRS application
- The sub-path to your resource (below all the initial parts) should be /fruits
- Respond to a GET request on this URI with the String “Bananas Apples and Oranges”