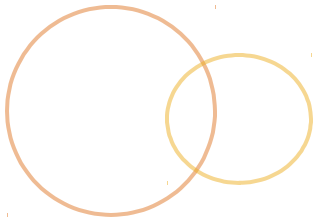
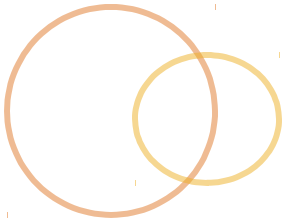
More Injection Features of JAX-RS

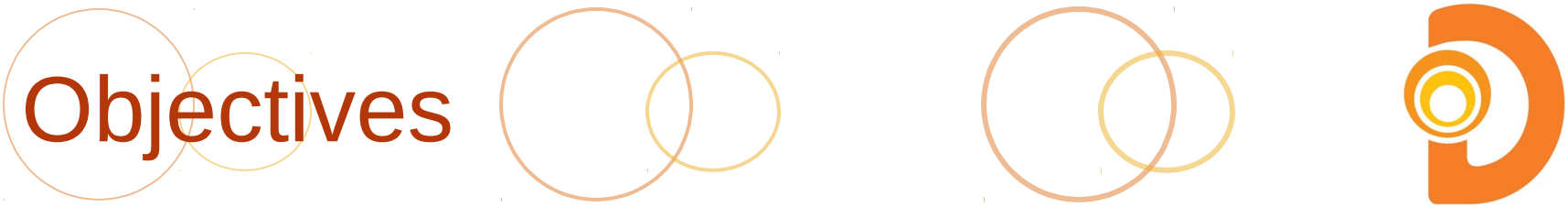




Objectives

- State the per-invocation lifecycle model of JAX-RS service objects
- Use @Singleton to create singleton service objects
- State the restriction on parameter injection that applies to singleton service objects
- Inject parameters directly into instance variables for non-singleton service objects
- Inject parameters into object constructors for non-singleton service objects
- Use @Context to inject UriInfo and/or HttpHeaders objects

Objectives



- Extract all the headers of a request from an HttpHeaders object into a MultivalueMap.
- Extract single values from a MultivalueMap
- Extract all query parameters from a UriInfo object

JAX-RS Default Object Lifecycle

- By default, every incoming request gets a newly created instance of the root resource class
- This allows the use of instance variables, rather than method parameters, as targets for injection either directly, or via a constructor

```
public class Resource {
  @HeaderParam(“intuit-tid”) private String tid;
  public Resource(@QueryParam(“name”) String name) {
```

Member Injection

- Injecting HTTP parameters into the service object, rather than the method parameters, can simplify handling large numbers of parameters, particularly if consistency rules or other code needs to be applied in all cases

Modifying The Lifecycle

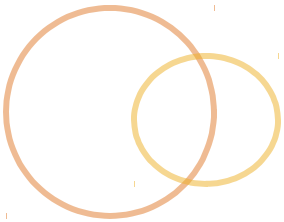
- If the per-invocation object lifecycle is not needed, it can be disabled
  - Annotate the service class @Singleton (JAX-RS 2.x)
  - If using the Application class approach to configuration, return an instance of the class from the method

```
public Set<Object> getSingletons()
```

  - And do not return the class in the method

```
public Set<Class<?>> getClasses()
```

Lifecycle and TCRS



- Recall that the Spring framework used in TCRS creates instances of business logic implementations, and injects these into the instances of the root resource classes
- Root resource classes will have the default (instance per request) lifecycle
- Business logic implementations will be singletons

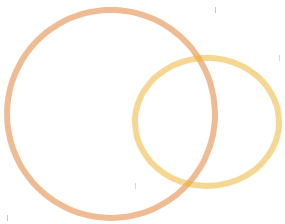
Injection In Singletons

- If a root resource is configured as a singleton, it cannot have injection into either the constructor, or member variables
  - This would be nonsensical, since the object is shared between multiple, potentially concurrent, requests, but the injected parameter data must be per-request

Using @Context

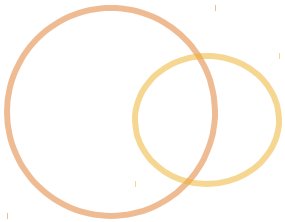
- The @Context annotation can inject (in any of the ways seen so far) some additional data, beyond the various XxxParam types seen so far
- Two types that can be injected with this annotation are generally interesting:
  - UriInfo—provides information about the request URI, such as absolute URI, path segments, and all query params (even ones you can't name in advance)
  - HttpHeaders—gives access to ***all*** the headers (even ones you can't name in advance)

Using HttpHeaders



```
@Path("/headers") @GET
public String getInfo(
  @Context HttpHeaders headers) {
  StringBuilder sb = new StringBuilder();
  sb.append("x-my-header: ")
    .append(headers.getHeaderString("x-my-header"))
    .append('\n');
  return sb.toString();
}
```

Using HttpHeaders



```
@Path("/headers") @GET
public String getInfo(
  @Context HttpHeaders headers) {
  StringBuilder sb = new StringBuilder();
  MultivaluedMap<String, String> hd =
    headers.getRequestHeaders();
  hd.forEach((k, lv)-> {
    sb.append("key: ").append(k).append('\n');
    lv.stream().forEach(v->{
      sb.append("  ").append(v).append('\n');
    });
  });
  return sb.toString();
}
```

Using UriInfo

```
@Path("/uri") @GET
public String getUriInfo(
  @Context UriInfo uriInfo) {
  StringBuilder sb = new StringBuilder();
  sb.append("Full URI: ")
    .append(uriInfo.getAbsolutePath().toString())
    .append('\n');
  List<PathSegment> lps = uriInfo.getPathSegments();
  lps.forEach(ps->
    sb.append(ps.getPath()).append('\n')
  );
  return sb.toString();
}
```

Using UriInfo

```
@Path("/uri") @GET
public String getUriInfo(
  @Context UriInfo uriInfo) {
  StringBuilder sb = new StringBuilder();
  MultivaluedMap<String, String> mmqp =
    uriInfo.getQueryParameters();
  mmqp.forEach((k, lv)-> {
    sb.append("key: ").append(k).append('\n');
    lv.stream().forEach(v->{
      sb.append("  ").append(v).append('\n');
    });
  });
  return sb.toString();
}
```

Lab Exercise

- Create a service endpoint