Java RESTful Clients

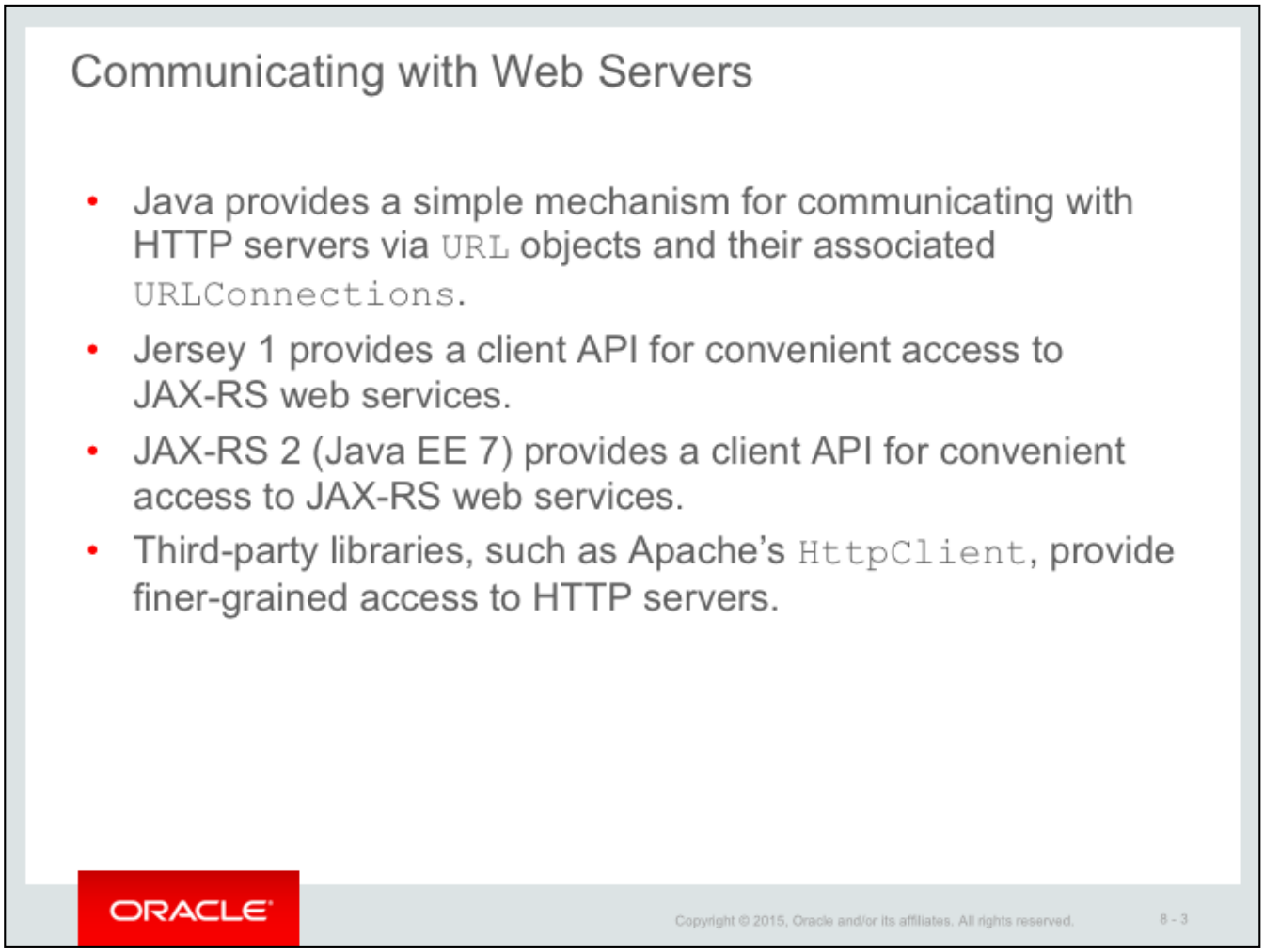
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Use the JAX-RS 2 Client API
- Describe the alternatives to the JAX-RS 2 Client API:
  - `HttpURLConnection` class
  - Jersey 1.X Client API

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.



JAX-RS 1.1 (Java EE 6) does not provide a client API but the JAX-RS 1.1 reference implementation, Jersey 1.x, includes a Jersey Client API. The Jersey Client API was the basis for the JAX-RS 2.0 Client API and only minor changes are needed to refactor code between the two APIs.

Creating a JAX-RS Client

```
public class JaxRsClient {
    public static void main(String[] args) {
        String baseURL =
                "http://localhost:8080/JaxRsExample/resources";
        Client client = ClientBuilder.newClient();
        WebTarget target
                = client.target(baseURL);
        target = target.path("message");
        Builder builder = target.request(MediaType.TEXT_PLAIN);
        String result = builder.get(String.class);
        System.out.println("Result: " + result);
```

Continue building the URL.

`WebTarget` represents a URL.

A `Builder` is used to specify headers.

Call an HTTP method, and provide the body (entity) and the return type.

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Method Chaining JAX-RS Client

```
public class JaxRsClient {
    public static void main(String[] args) {
        String baseURL =
                "http://localhost:8080/JaxRsExample/resources";
        Client client = ClientBuilder.newClient();
        WebTarget target
                = client.target(baseURL);
        String result = target
                .path("message")
                .request(MediaType.TEXT_PLAIN)
                .get(String.class);
        System.out.println("Result: " + result);
    }
}
```

Method calls are chained together.

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

javax.ws.rs.client.WebTarget

`WebTarget` allows the application to:

- Represent a base or complete URI
  - `client.target(String)`
- Append additional path elements to the URI
  - `path(String)`
- Append Query Parameters
  - `queryParam(String, Object…)`
- Append Matrix Parameters
  - `matrixParam(String, Object…)`
- Resolve a URI template
  - `resolveTemplate(String, Object)`
- Create an `Invocation.Builder` for a content type
  - `request(MediaType)`

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

javax.ws.rs.client.Invocation.Builder

`Invocation.Builder` allows the application to:

- Set the Accept header
  - `accept(MediaType...)`
- Set the Accept-Language header
  - `acceptLanguage(Locale...)`
- Update the Cookie header
  - `cookie(Cookie)`
- Set any HTTP header
  - `header(String, Object)`
- Call an HTTP method synchronously
  - `get`, `put`, `post`, `delete`, `head`, `trace`, `options`
- Call an HTTP method asynchronously
  - `async()`

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Obtaining the Response Entity

All the `Invocation.Builder` synchronous HTTP methods have three overloaded versions:

- Obtain the response entity by class type:

```
Person p = builder.get(Person.class);
```

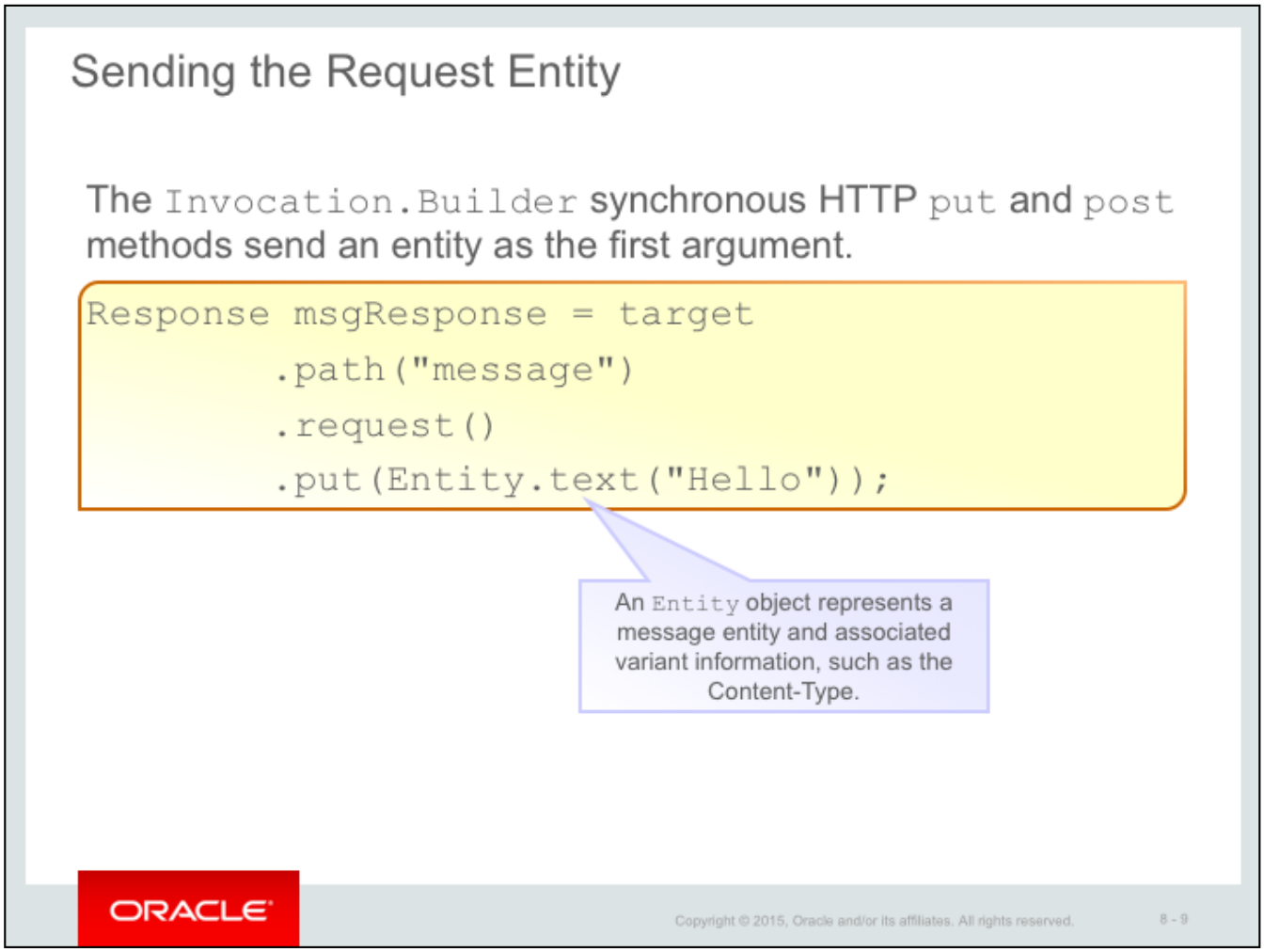
- Obtain a generic response entity with `GenericType`:

```
List<User> list =
 builder.get(new GenericType<List<User>>(){});
```

- Obtain a response, analyze it, and then read the entity:

```
Response response = builder.get();
//analyze Response status or headers
Person p = response.readEntity(Person.class);
```

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.



When calling the `put` or `post` methods, the Content-Type, Content-Language, and Content-Encoding headers will be set automatically.

There are five convenience methods to create `Entity` objects with common media types: `html`, `json`, `text`, `xhtml`, `xml`. When creating `Entity` objects of other types or if you need to override the language or encoding, use one of the static `entity` methods:

- `Entity.entity(T entity, MediaType mediaType)`
- `Entity.entity(T entity, Variant variant)`

Obtaining Reply Metadata

`Response` represents the complete reply message received by the client. Its API allows the application to access:

- Status code
- Message payload
- Response Headers
- Response Cookies

```
int status = msgResponse.getStatus();
Map<String,NewCookie> cookies =
  msgResponse.getCookies();
String power =
  msgResponse.getHeaderString("X-Powered-By");
Date date = msgResponse.getDate();
```

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Web Service Errors

Web Services can experience errors in two places:

- On the server
  - In your web service an exception is thrown. How you convey that to a client depends on the type of web service (SOAP or REST).
- On the client
  - Clients receive the errors produced by a web service.
  - Clients experience error without there being any error produced by a server (networking problems, for example).

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

javax.ws.rs.WebApplicationException

When not obtaining a `Response`, a `WebApplicationException` subclass is thrown. There are three direct sublcasses:

- `RedirectionException` – HTTP 3XX status codes
- `ClientErrorException` – HTTP 4XX status codes
- `ServerErrorException` – HTTP 5XX status codes

A more specific subclass will be thrown.

```
String stringResult = target
        .path("invalidpath")
        .request(MediaType.TEXT
        .get(String.class);
```

A `javax.ws.rs.NotFoundException` is thrown. `WebApplicationException` is a subclass of `RuntimeException`.

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Reading `Response` Error Status

When getting a `Response`, no exception is thrown.

```
int status = msgResponse.getStatus();
if(status >= 400 && status < 500) {
    System.out.println("Client error");
} else if(status >= 500) {
    System.out.println("Server error");
} else {
    System.out.println("STATUS: " + status);
}
```

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Alternative Java REST Clients

Although the JAX-RS 2 Client API (Java EE 7) is not available to you, you can still access RESTful web services in a number of ways:

- Jersey 1.X Client API
- `java.net.URL` and `java.net.HttpURLConnection`
- Third-party clients such as Apache's HttpClient

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

For more about Apache's HttpClient library, see http://hc.apache.org/httpcomponents-client-ga/.

Jersey 1.X Client API

JAX-RS 1.1 does not contain a client API. The JAX-RS 1.1 reference implementation (Jersey 1.X) provides a client API.

JAX-RS 2 Client	Jersey 1 Client
`ClientBuilder`	`com.sun.jersey.api.client.Client`
`Client`	`com.sun.jersey.api.client.Client`
`WebTarget`	`com.sun.jersey.api.client.WebResource`
`Response`	`com.sun.jersey.api.client.ClientResponse`

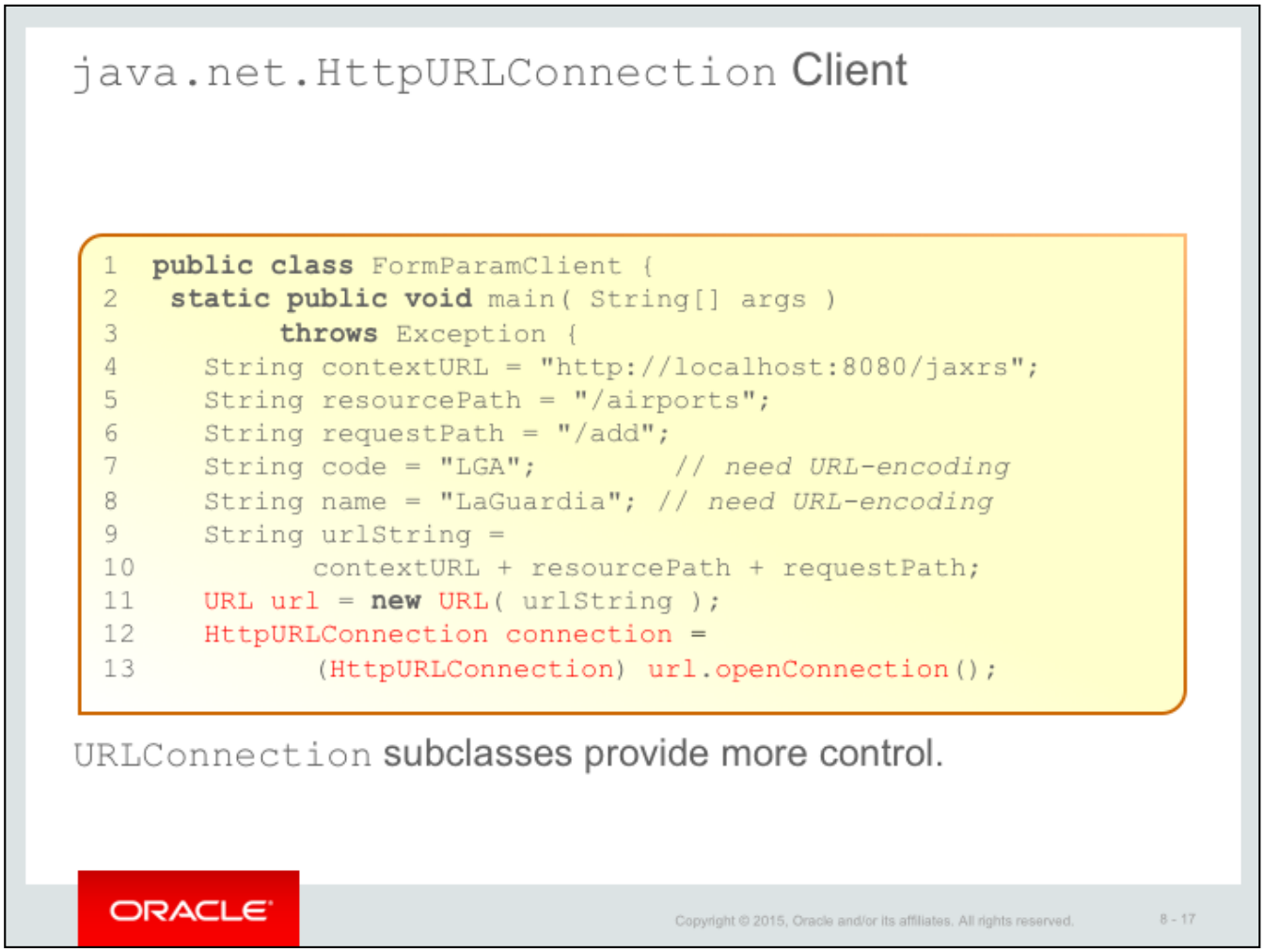
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The Jersey 1.X Client API provided the basis for the JAX-RS 2 Client API, but there are a couple of differences. For more information about transitioning from the Jersey 1.X Client API to the JAX-RS 2.0 Client API, see https://jersey.java.net/documentation/latest/migration.html#mig-client-api.

java.net.URL Client

```
1  public class SimplestClient {
2   static public void main( String[] args )
3         throws Exception {
4    String contextURL = "http://localhost:8080/jaxrs";
5    String resourcePath = "/airports";
6    String requestPath = "/numAirports";
7    String urlString =
8       contextURL + resourcePath + requestPath;
9    URL url = new URL( urlString );
10   InputStream result = (InputStream) url.getContent();
11   Scanner scanner = new Scanner( result );
12   System.out.println( "Result:_" + scanner.next() );
13  }
14 }
```

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.



Drawbacks of the Simple Approach

- Requires explicit matching of URL rewrite rules. To avoid invalid URLs, parameters may need to be provided using URL-encoding.
- Requires some awareness of the structure of HTTP messages
- Requires low-level I/O programming

java.net.HttpURLConnection Client

```
14    connection.setRequestMethod( "POST" );
15    connection.setAllowUserInteraction( true );
16    connection.setDoOutput( true );
17    connection.setDoInput( true );
18    connection.connect();
19    OutputStream os = connection.getOutputStream();
20    PrintWriter writer = new PrintWriter( os );
21    writer.print( "code=" + code + "&name=" + name );
22    writer.close();
23    InputStream result = connection.getInputStream();
24    BufferedReader reader =
25     new BufferedReader( new InputStreamReader(result) );
26    System.out.println( "Result: " + reader.readLine() );
27   }
28 }
```

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

HttpUrlConnection Response Status

Use the HttpUrlConnection.getResponseCode() method to get the HTTP response status:

```
URL url = new
URL("http://localhost:7001/app/resources/root");
HttpURLConnection conn =
    (HttpURLConnection) url.openConnection();
conn.connect();

switch(conn.getResponseCode()) {
  case 404:
  case HttpURLConnection.HTTP_BAD_REQUEST:
  default:
}
```

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Resources

Topic	Website
`HttpURLConnection`	http://docs.oracle.com/javase/7/docs/api/java/net/HttpURLConnection.html
Jersey Client API	https://jersey.java.net/documentation/1.18/client-api.html
JAX-RS 2 Client API	https://jersey.java.net/documentation/latest/client.html

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Use the JAX-RS 2 Client API
- Describe the alternatives to the JAX-RS 2 Client API:
  - `HttpURLConnection` class
  - Jersey 1.X Client API

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Practice 8: Overview

This practice covers the following topics:

- Creating a Java SE RESTful (JAX-RS) client to list the collection of media elements
- Adding to the client to read and delete individual media elements
- Adding to the client to add a new media element to the collection
- (Optional) Invoking the UploadServlet by using the RESTful client

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.