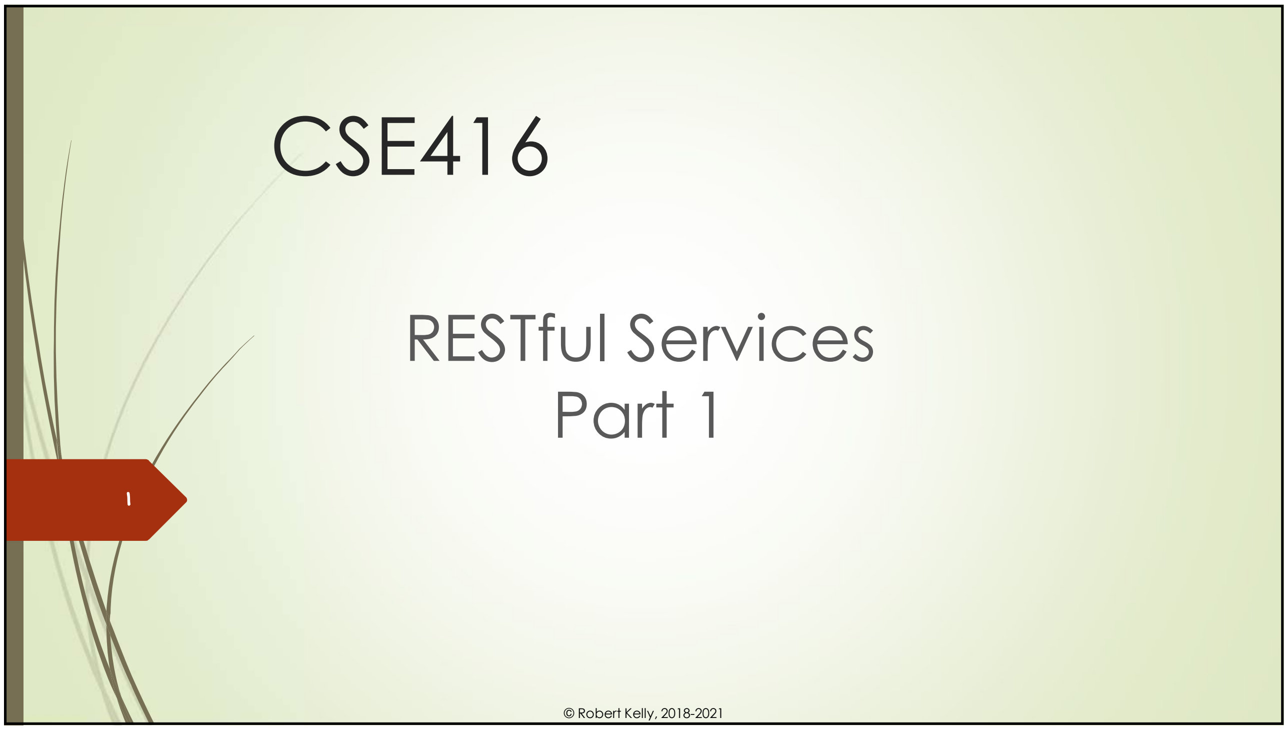


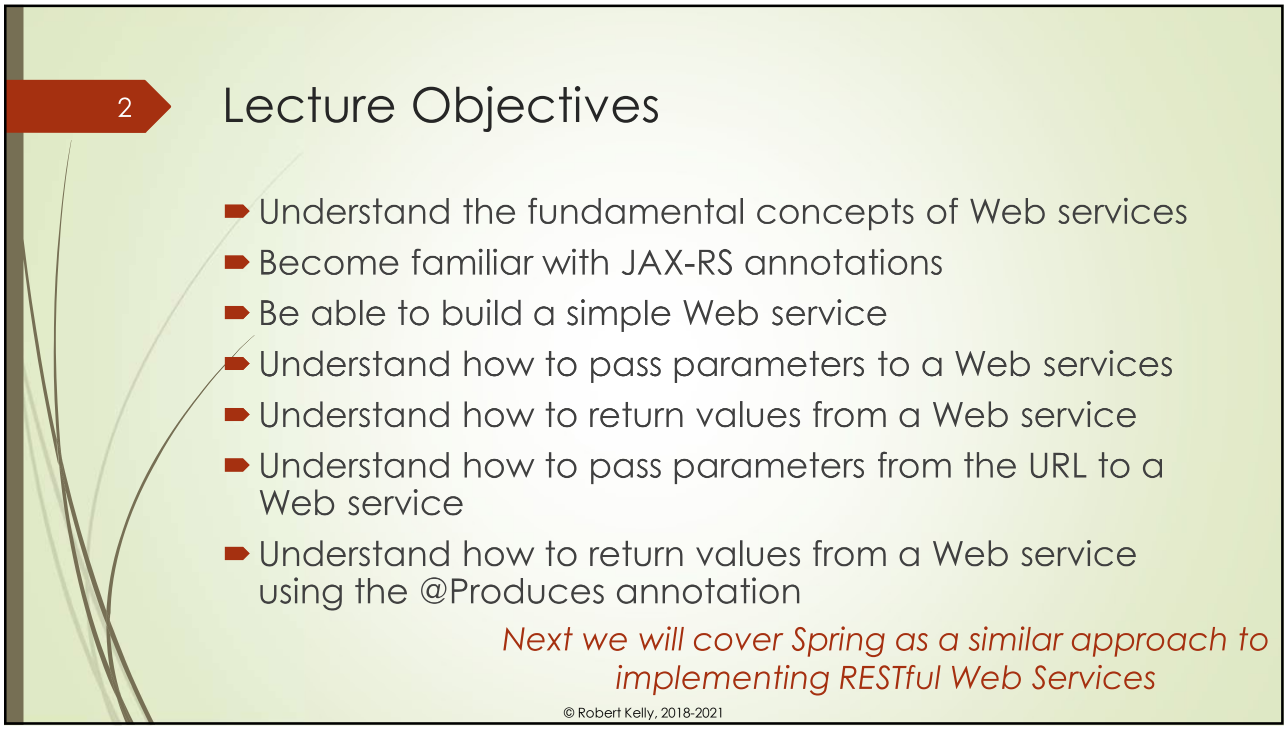
1

CSE416

RESTful Services
Part 1

© Robert Kelly, 2018-2021

1



2

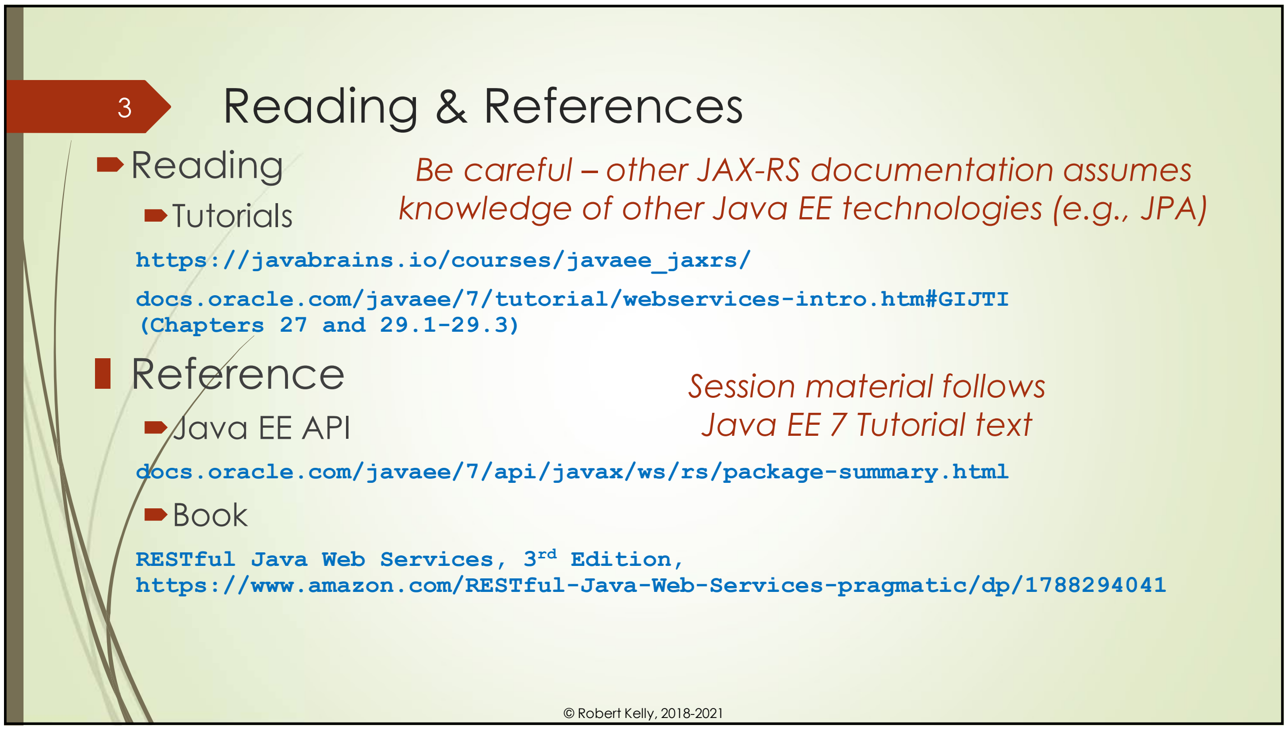
Lecture Objectives

- Understand the fundamental concepts of Web services
- Become familiar with JAX-RS annotations
- Be able to build a simple Web service
- Understand how to pass parameters to a Web services
- Understand how to return values from a Web service
- Understand how to pass parameters from the URL to a Web service
- Understand how to return values from a Web service using the @Produces annotation

Next we will cover Spring as a similar approach to implementing RESTful Web Services

© Robert Kelly, 2018-2021

2



3

Reading & References

- Reading
  - Tutorials

Be careful – other JAX-RS documentation assumes knowledge of other Java EE technologies (e.g., JPA)

`https://javabrains.io/courses/javaee_jaxrs/`

`docs.oracle.com/javaee/7/tutorial/webservices-intro.htm#GIJTI (Chapters 27 and 29.1-29.3)`

- Reference
  - Java EE API

Session material follows Java EE 7 Tutorial text

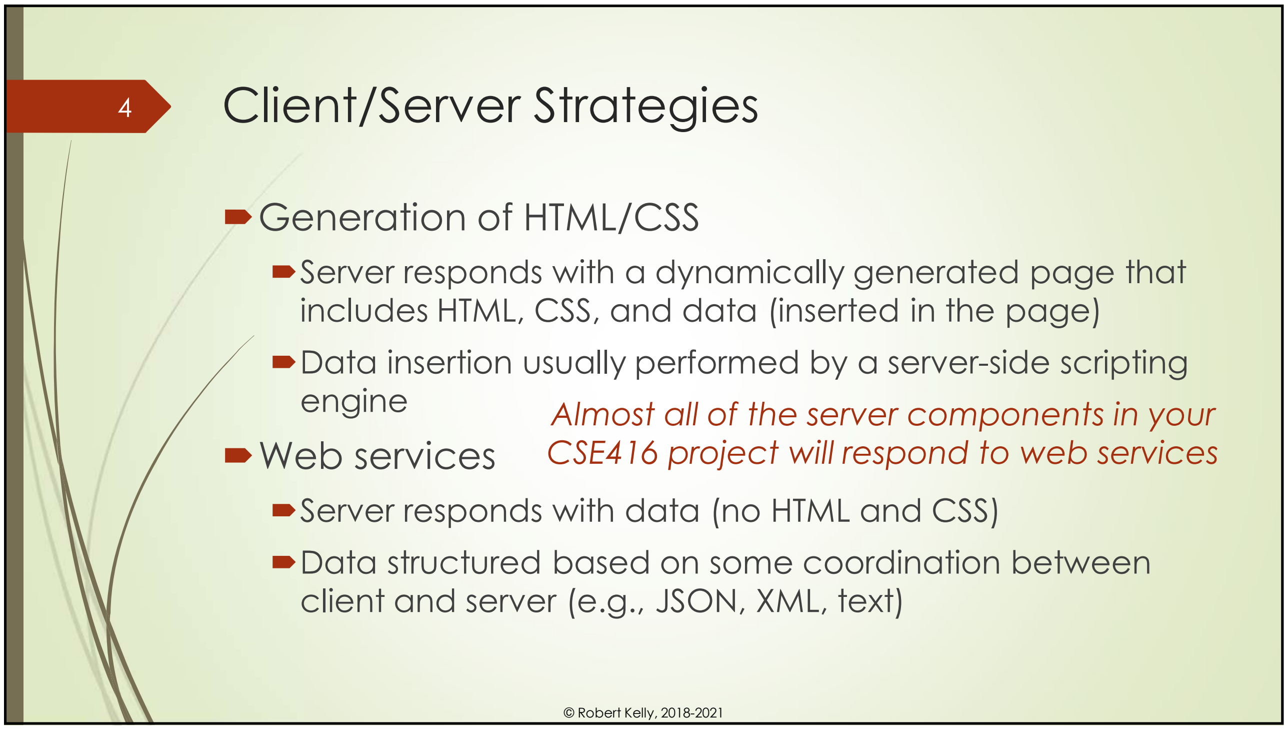
`docs.oracle.com/javaee/7/api/javax/ws/rs/package-summary.html`

  - Book

`RESTful Java Web Services, 3rd Edition, https://www.amazon.com/RESTful-Java-Web-Services-pragmatic/dp/1788294041`

© Robert Kelly, 2018-2021

3



4

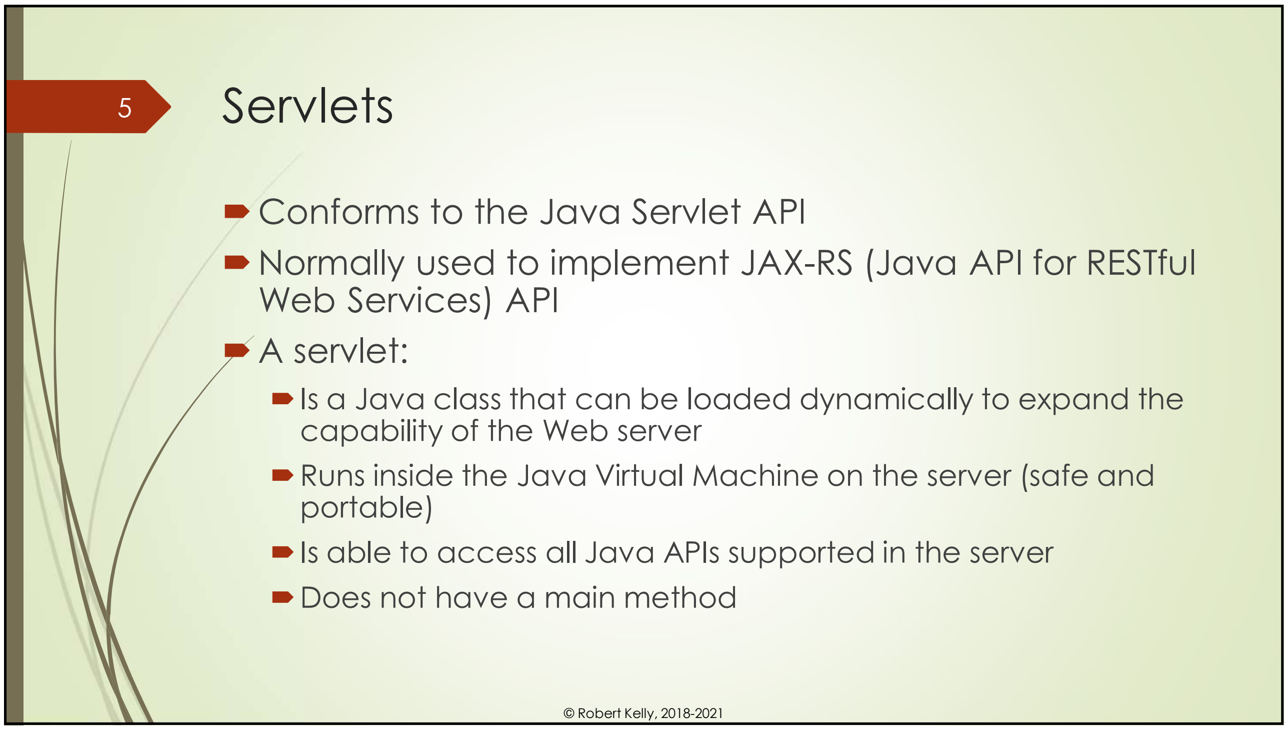
Client/Server Strategies

- Generation of HTML/CSS
  - Server responds with a dynamically generated page that includes HTML, CSS, and data (inserted in the page)
  - Data insertion usually performed by a server-side scripting engine
- Web services
  - Server responds with data (no HTML and CSS)
  - Data structured based on some coordination between client and server (e.g., JSON, XML, text)

Almost all of the server components in your CSE416 project will respond to web services

© Robert Kelly, 2018-2021

4

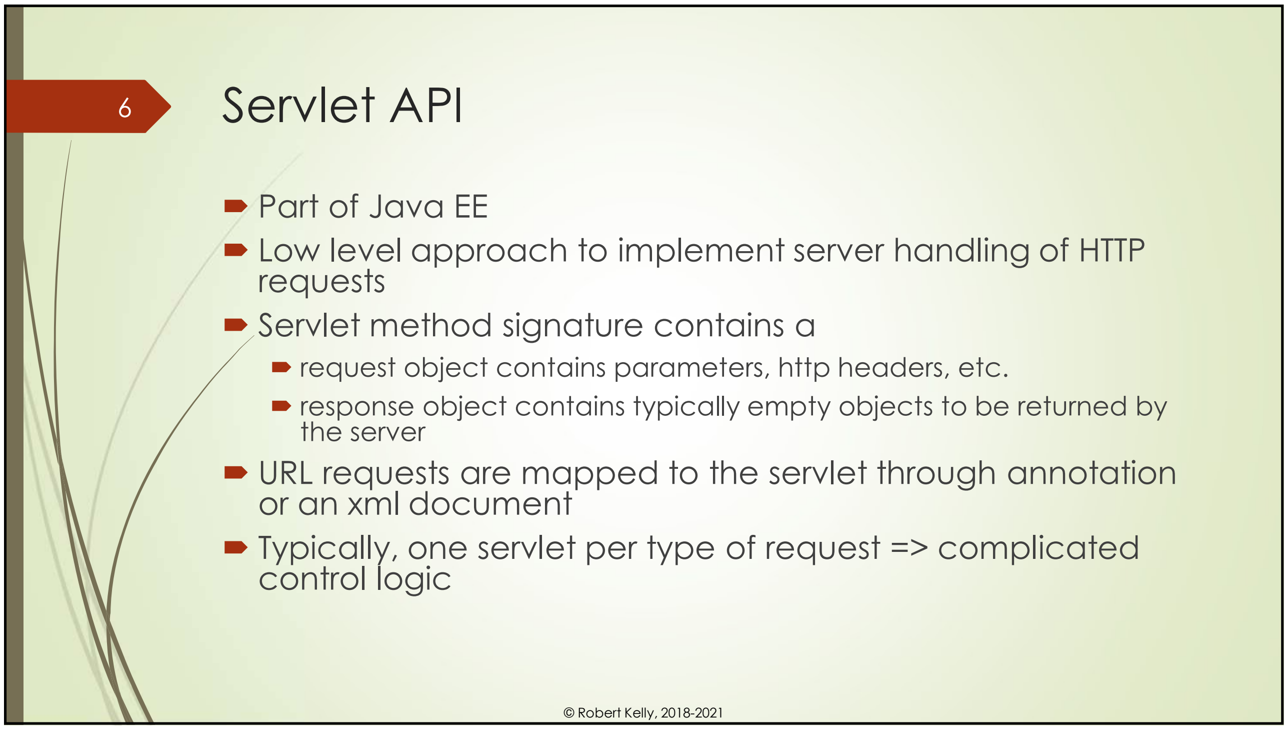


5

Servlets

- Conforms to the Java Servlet API
- Normally used to implement JAX-RS (Java API for RESTful Web Services) API
- A servlet:
  - Is a Java class that can be loaded dynamically to expand the capability of the Web server
  - Runs inside the Java Virtual Machine on the server (safe and portable)
  - Is able to access all Java APIs supported in the server
  - Does not have a main method

5



6

Servlet API

- Part of Java EE
- Low level approach to implement server handling of HTTP requests
- Servlet method signature contains a
  - request object contains parameters, http headers, etc.
  - response object contains typically empty objects to be returned by the server
- URL requests are mapped to the servlet through annotation or an xml document
- Typically, one servlet per type of request => complicated control logic

6

7

Client – Servlet Model

- Requires logic in servlet to route each request to a service method
- Does not directly use URL and other http data to route to a service
- Mapping of the URL to a servlet is handled with web.xml or Java Annotation in servlet class

```
<form method="get" action=
"http://localhost:8080/CSE336-2017/helloyou.html">
```

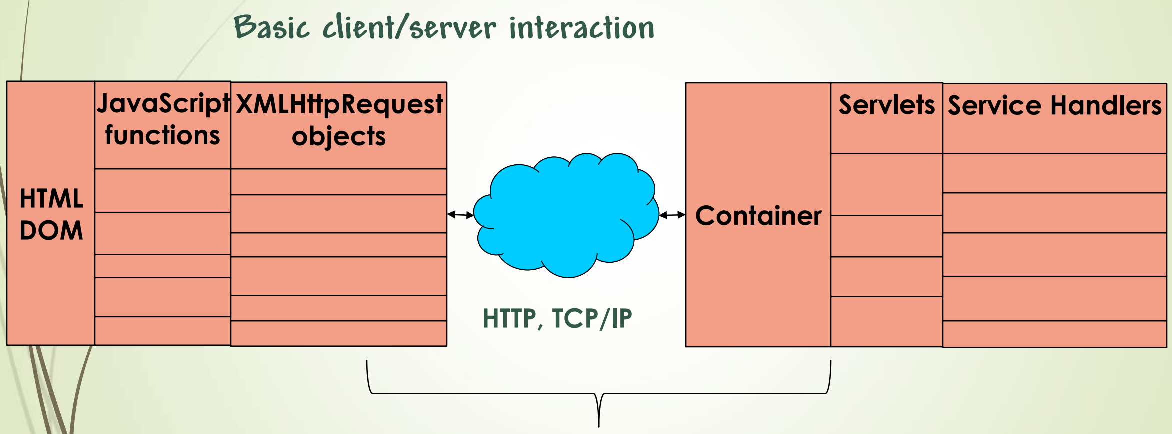
Servlet identified by the "helloyou.html" URL string usually acts as a controller, and routes to a service handler

React access will use a URL, but look different

7

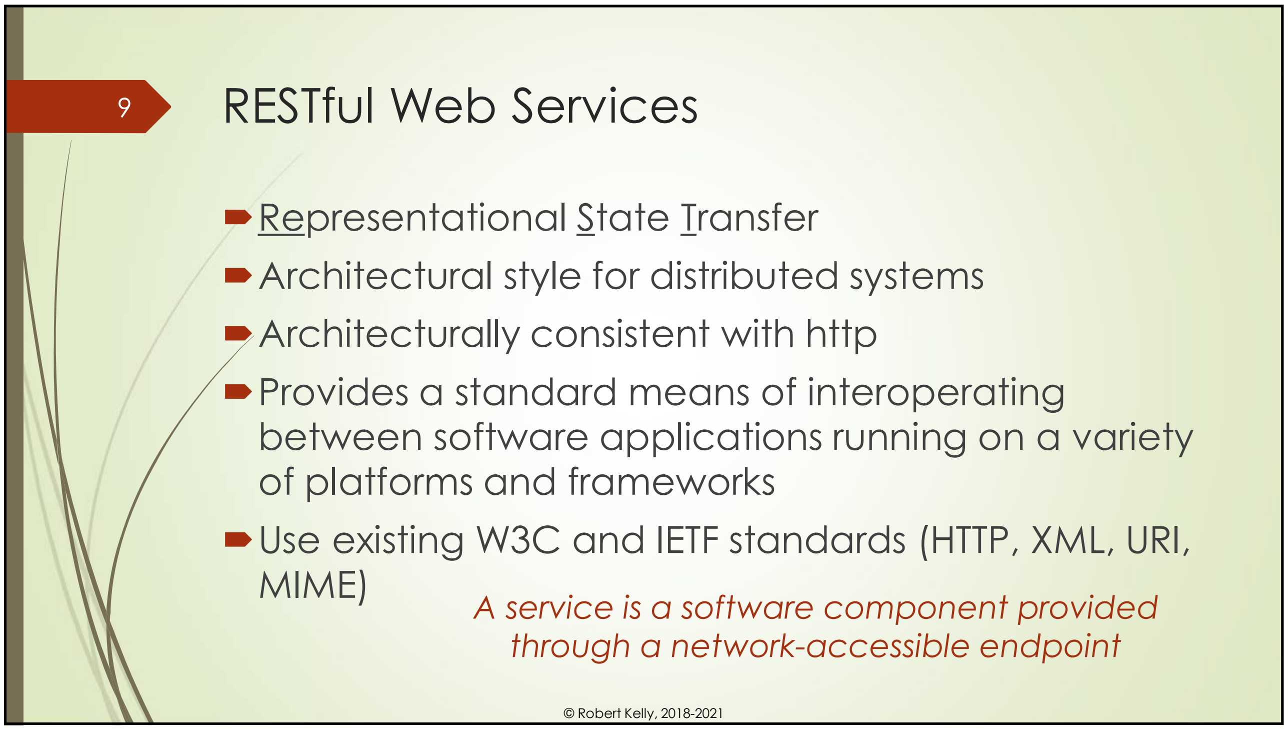
8

Servlet API Client/Server Interaction



Parts of the client/server interaction are abstracted by tools/libraries

8



9

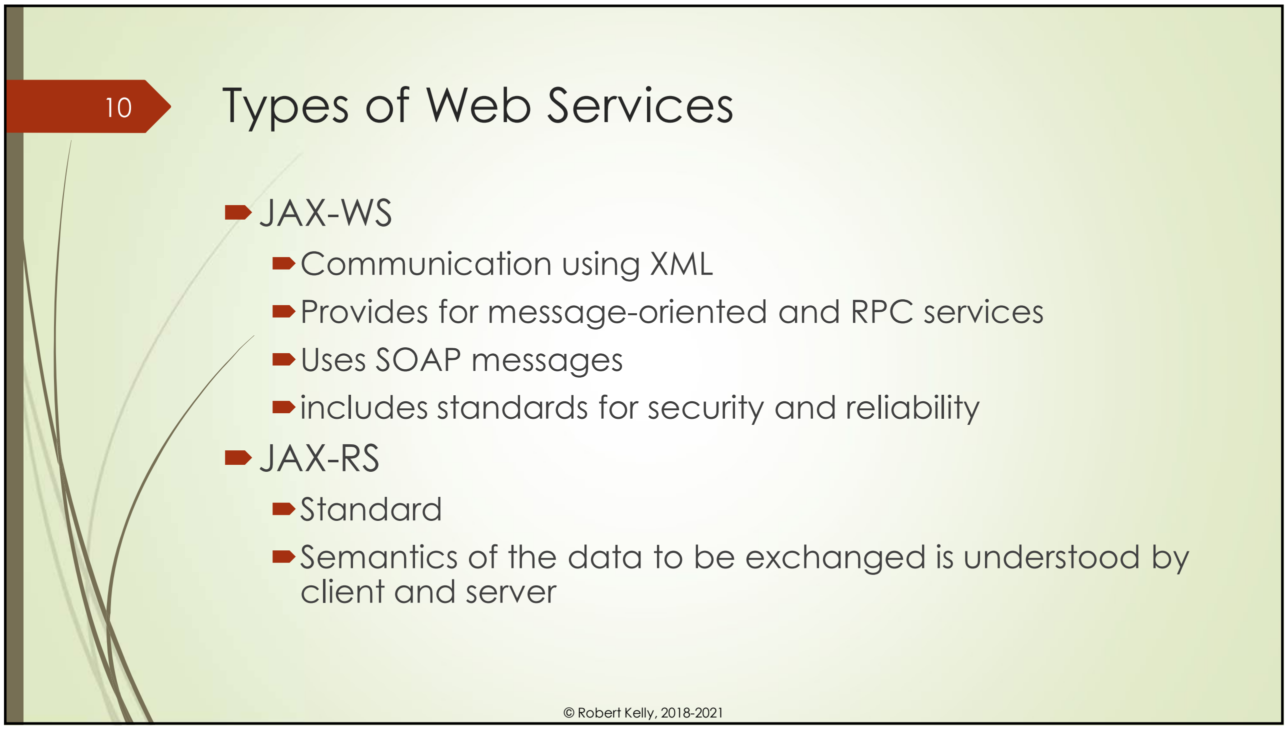
RESTful Web Services

- Representational State Transfer
- Architectural style for distributed systems
- Architecturally consistent with http
- Provides a standard means of interoperating between software applications running on a variety of platforms and frameworks
- Use existing W3C and IETF standards (HTTP, XML, URI, MIME)

A service is a software component provided through a network-accessible endpoint

© Robert Kelly, 2018-2021

9



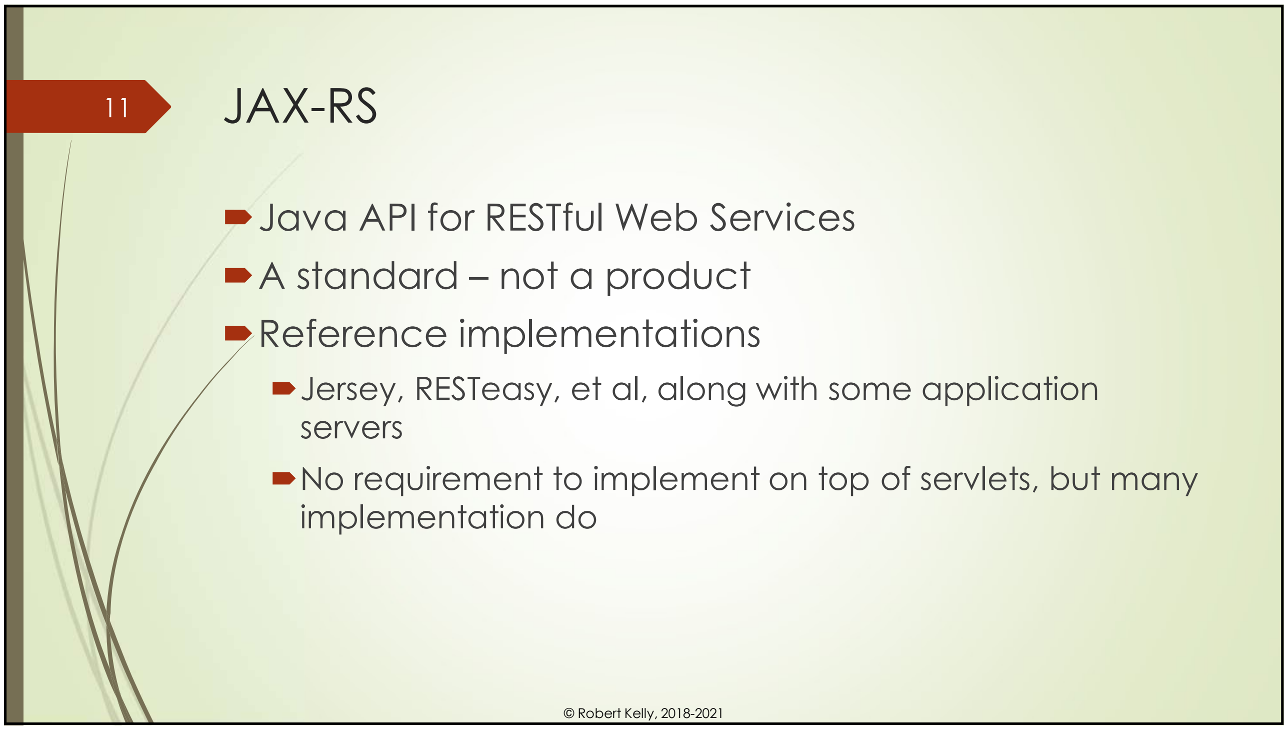
10

Types of Web Services

- JAX-WS
  - Communication using XML
  - Provides for message-oriented and RPC services
  - Uses SOAP messages
  - includes standards for security and reliability
- JAX-RS
  - Standard
  - Semantics of the data to be exchanged is understood by client and server

© Robert Kelly, 2018-2021

10



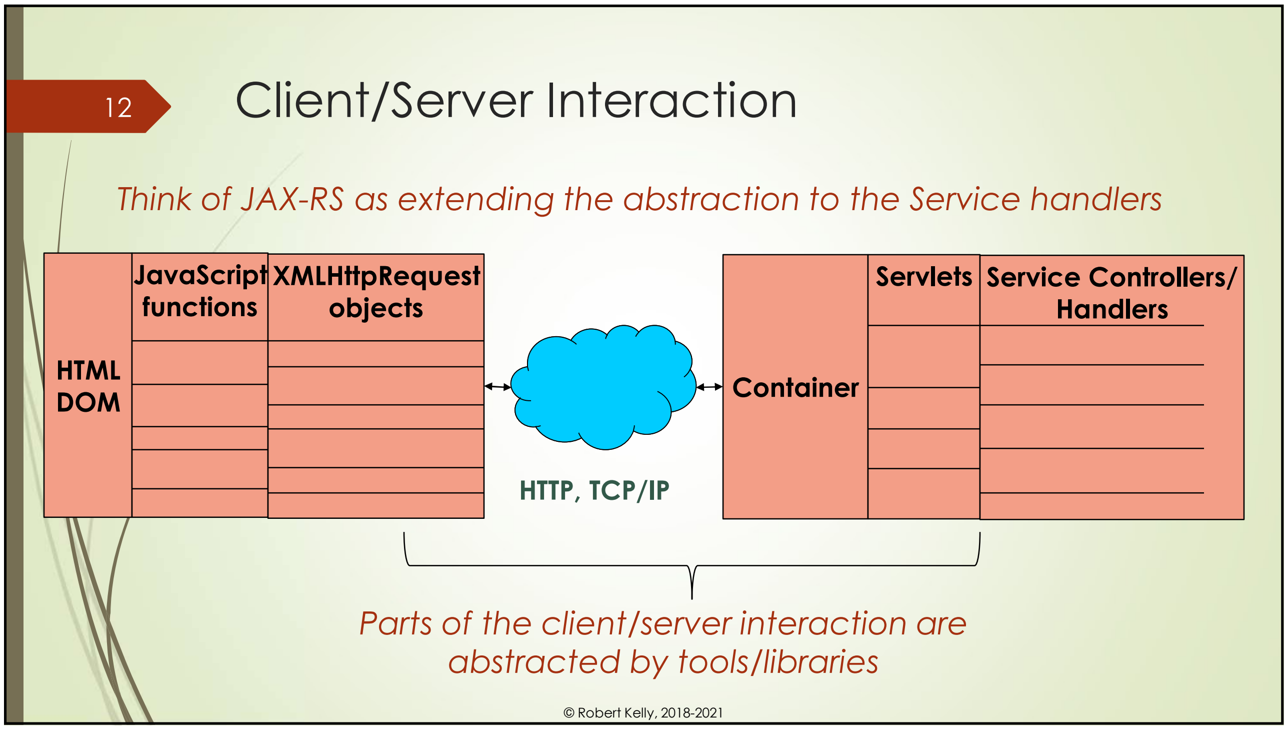
JAX-RS

- Java API for RESTful Web Services
- A standard – not a product
- Reference implementations
  - Jersey, RESTeasy, et al, along with some application servers
  - No requirement to implement on top of servlets, but many implementation do

11

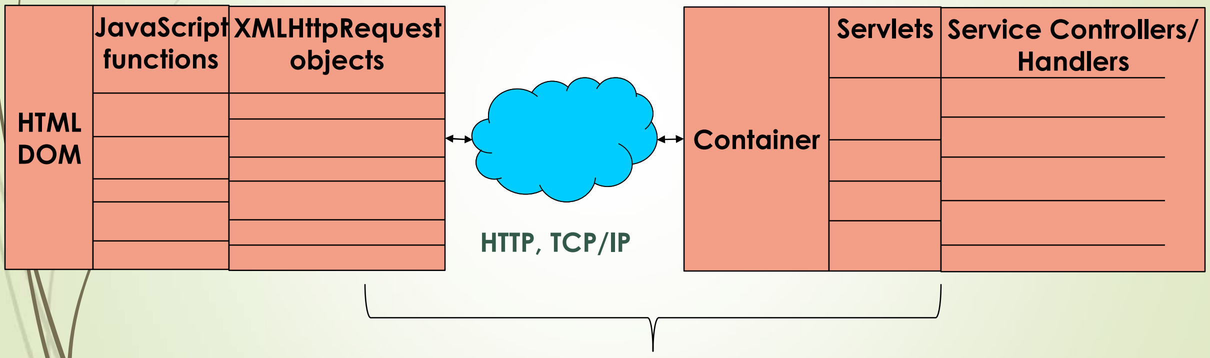
© Robert Kelly, 2018-2021

11



Client/Server Interaction

Think of JAX-RS as extending the abstraction to the Service handlers

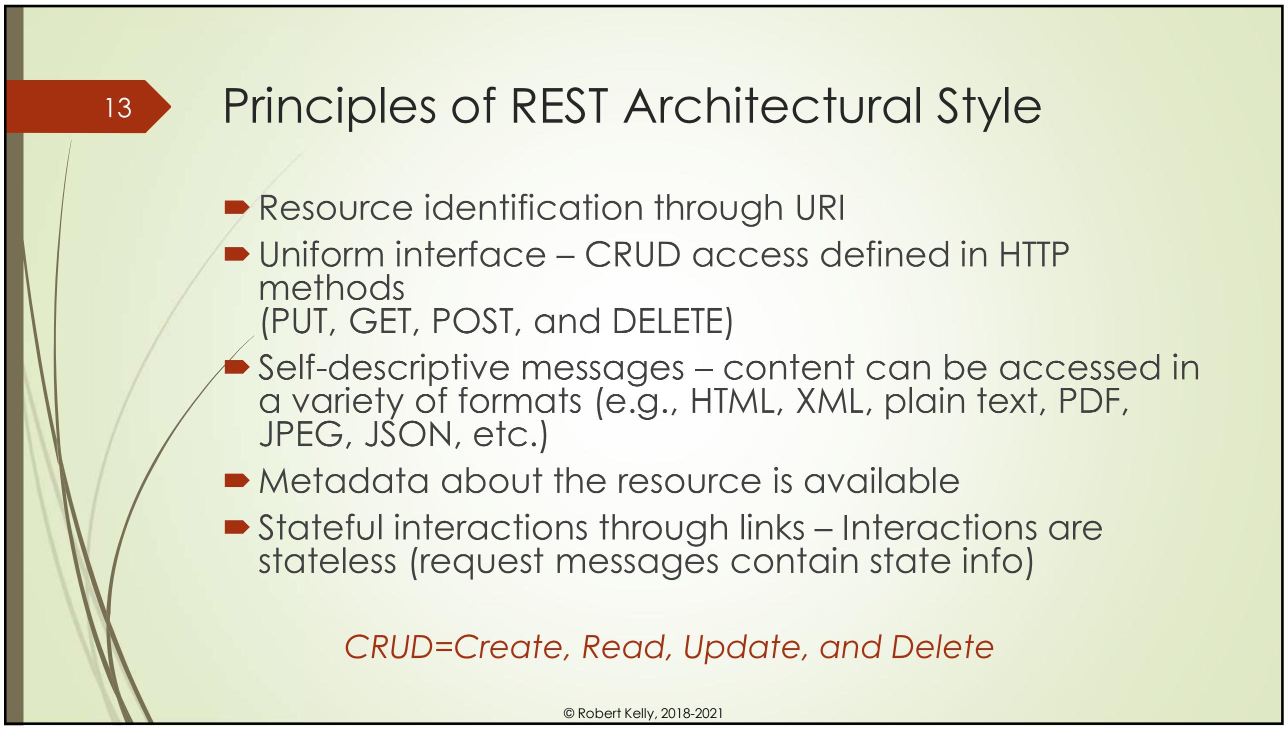


Parts of the client/server interaction are abstracted by tools/libraries

12

© Robert Kelly, 2018-2021

12



13

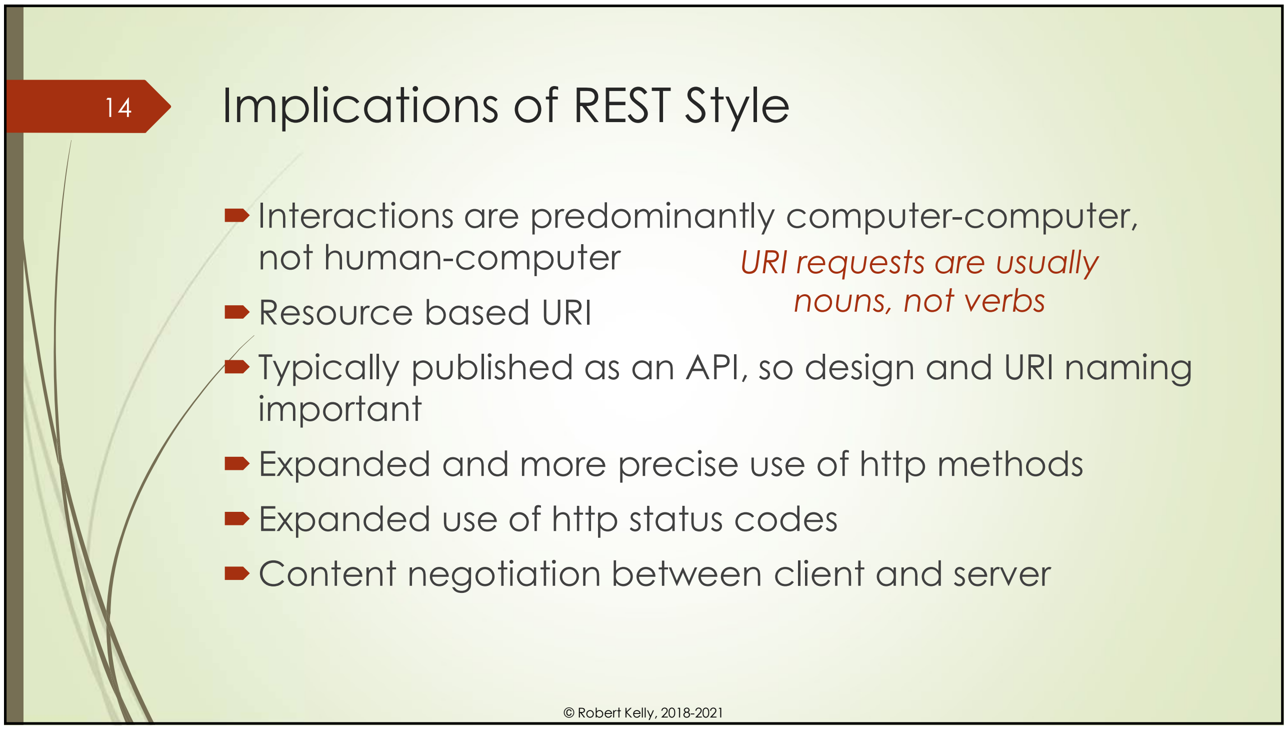
Principles of REST Architectural Style

- Resource identification through URI
- Uniform interface – CRUD access defined in HTTP methods
  (PUT, GET, POST, and DELETE)
- Self-descriptive messages – content can be accessed in a variety of formats (e.g., HTML, XML, plain text, PDF, JPEG, JSON, etc.)
- Metadata about the resource is available
- Stateful interactions through links – Interactions are stateless (request messages contain state info)

CRUD=Create, Read, Update, and Delete

© Robert Kelly, 2018-2021

13



14

Implications of REST Style

- Interactions are predominantly computer-computer, not human-computer
- Resource based URI
- Typically published as an API, so design and URI naming important
- Expanded and more precise use of http methods
- Expanded use of http status codes
- Content negotiation between client and server

URI requests are usually nouns, not verbs

© Robert Kelly, 2018-2021

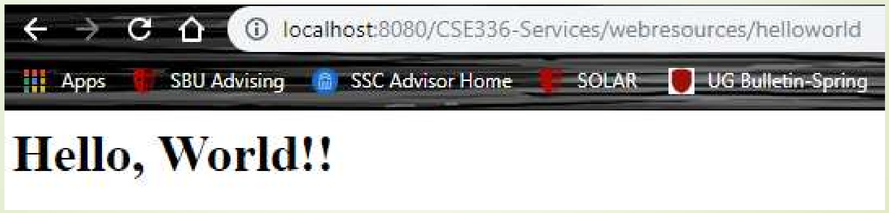
14

15

Example

- We start by building a very simple RESTful service
- Later we will extend this by
  - Passing parameters to the server
  - Negotiating content
  - Returning content

For all the examples, think of accessing the resources from your html/JavaScript running in your browser



Hello, World!!

© Robert Kelly, 2018-2021

15

16

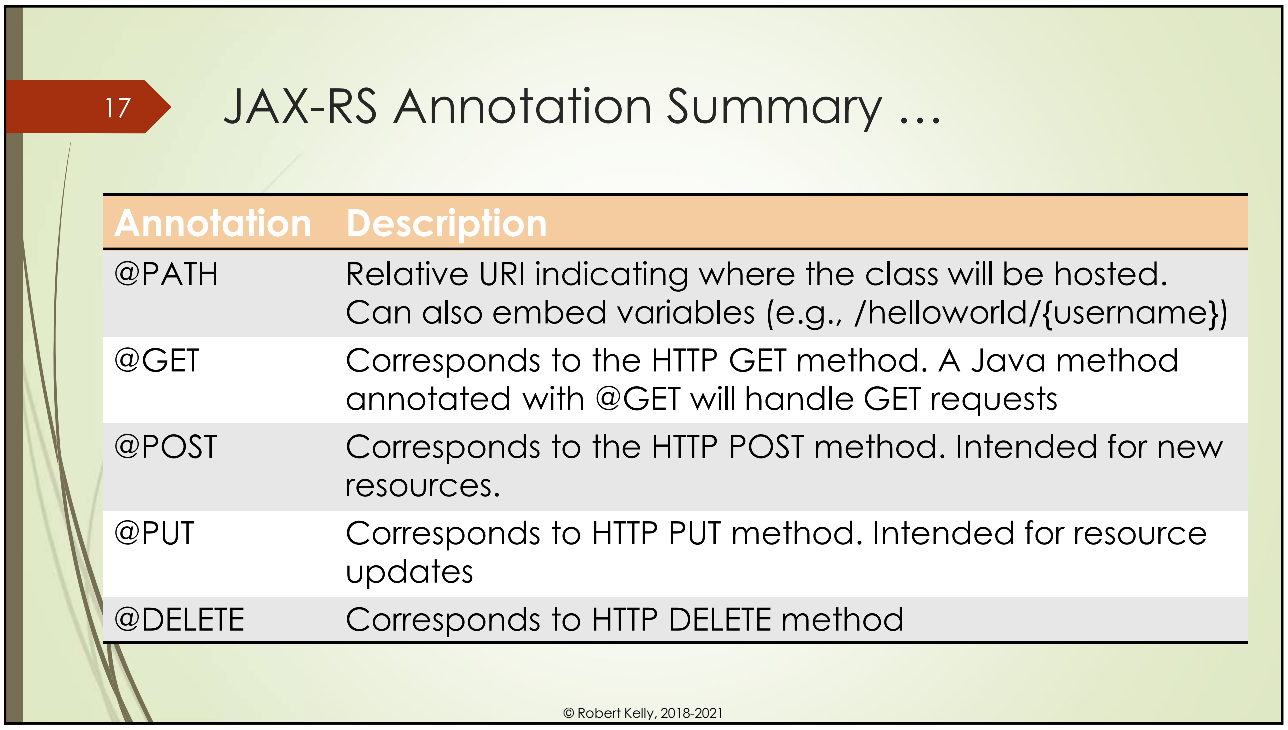
Creating a RESTful Root Resource Class

- Root resource classes are POJOs (plain old Java objects)
- Annotated with @Path or a request method designator (@GET, @PUT, @POST, or @DELETE)

JAX-RS uses Java Annotations

© Robert Kelly, 2018-2021

16



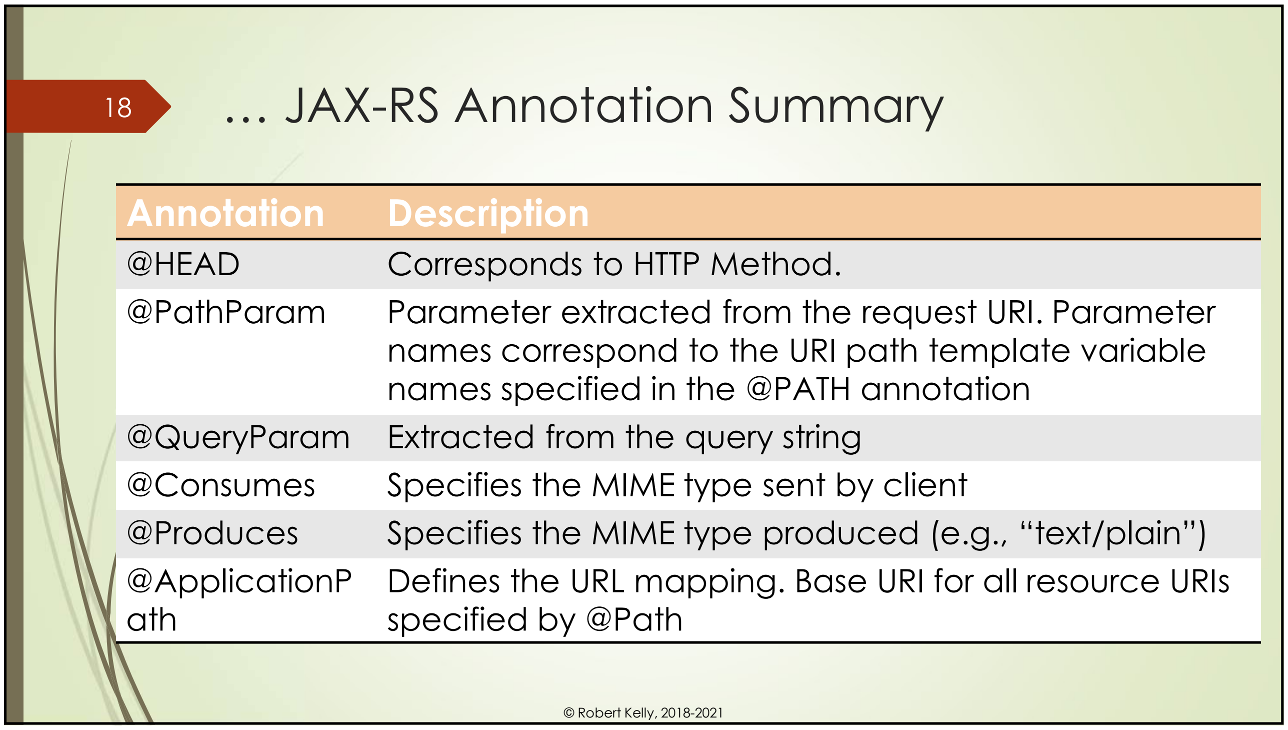
17

JAX-RS Annotation Summary …

Annotation	Description
@PATH	Relative URI indicating where the class will be hosted. Can also embed variables (e.g., /helloworld/{username})
@GET	Corresponds to the HTTP GET method. A Java method annotated with @GET will handle GET requests
@POST	Corresponds to the HTTP POST method. Intended for new resources.
@PUT	Corresponds to HTTP PUT method. Intended for resource updates
@DELETE	Corresponds to HTTP DELETE method

© Robert Kelly, 2018-2021

17



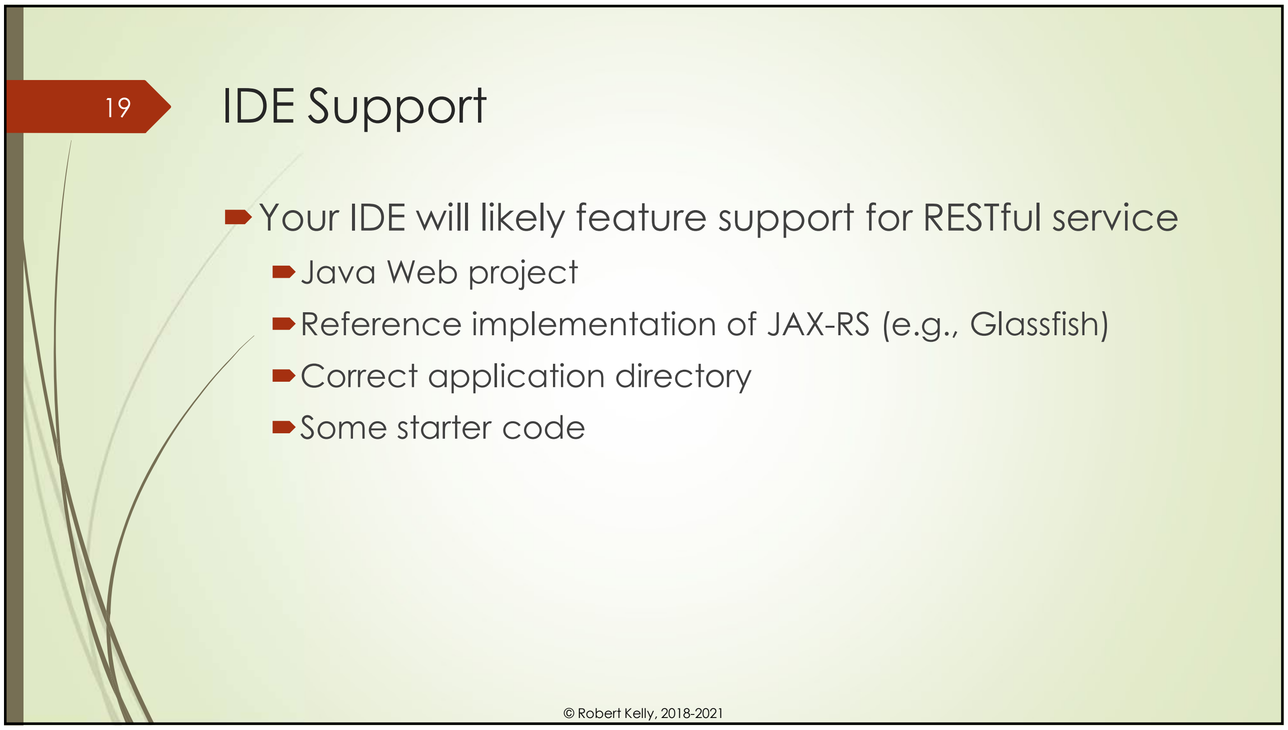
18

… JAX-RS Annotation Summary

Annotation	Description
@HEAD	Corresponds to HTTP Method.
@PathParam	Parameter extracted from the request URI. Parameter names correspond to the URI path template variable names specified in the @PATH annotation
@QueryParam	Extracted from the query string
@Consumes	Specifies the MIME type sent by client
@Produces	Specifies the MIME type produced (e.g., “text/plain”)
@ApplicationPath	Defines the URL mapping. Base URI for all resource URIs specified by @Path

© Robert Kelly, 2018-2021

18

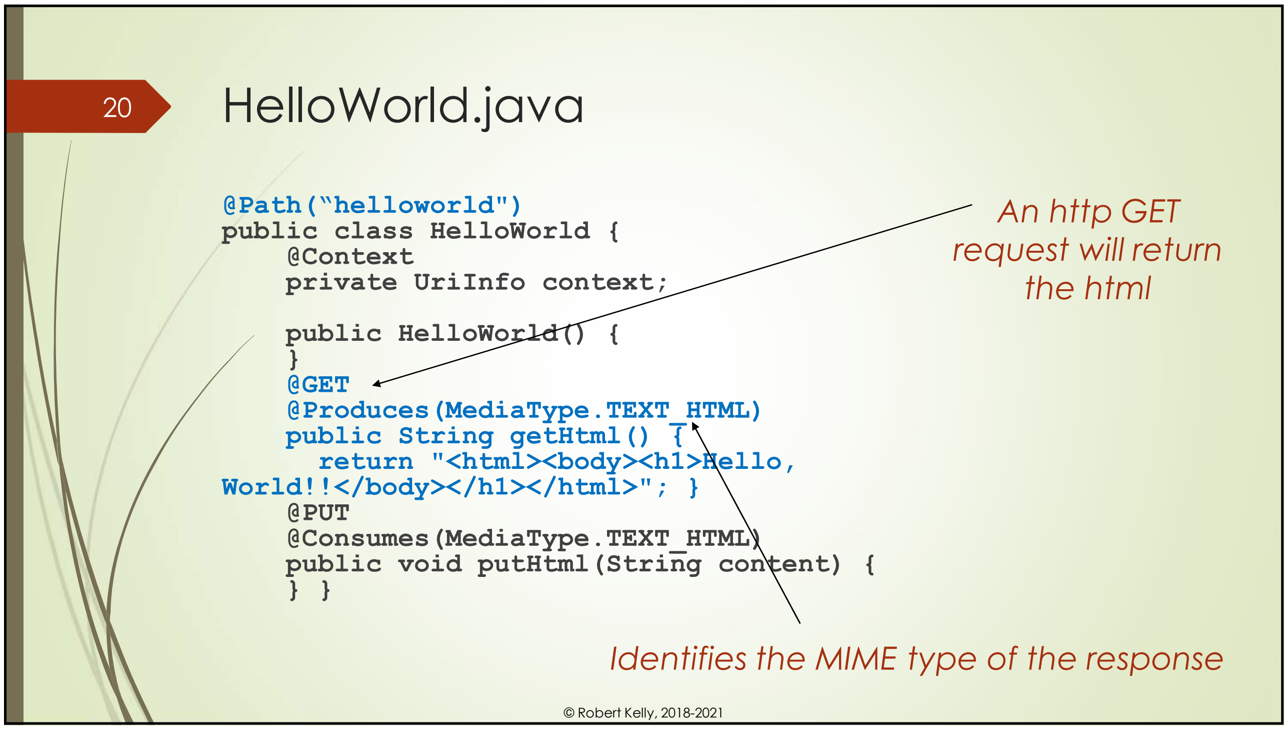


19

IDE Support

- Your IDE will likely feature support for RESTful service
  - Java Web project
  - Reference implementation of JAX-RS (e.g., Glassfish)
  - Correct application directory
  - Some starter code

19



20

HelloWorld.java

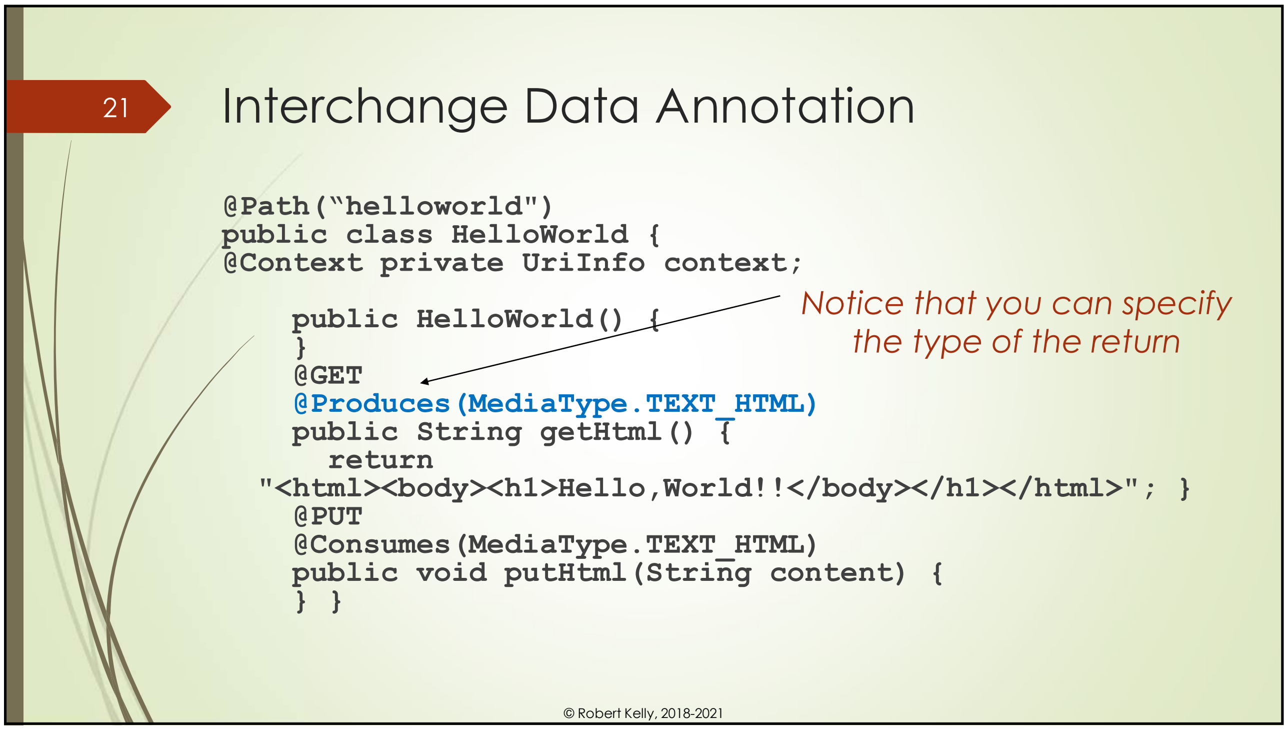
```
@Path(“helloworld")
public class HelloWorld {
     @Context
     private UriInfo context;

     public HelloWorld() {
     }
     @GET
     @Produces(MediaType.TEXT_HTML)
     public String getHtml() {
       return "<html><body><h1>Hello,
World!!</body></h1></html>"; }
     @PUT
     @Consumes(MediaType.TEXT_HTML)
     public void putHtml(String content) {
     } }
```

An http GET request will return the html

Identifies the MIME type of the response

20



21

Interchange Data Annotation

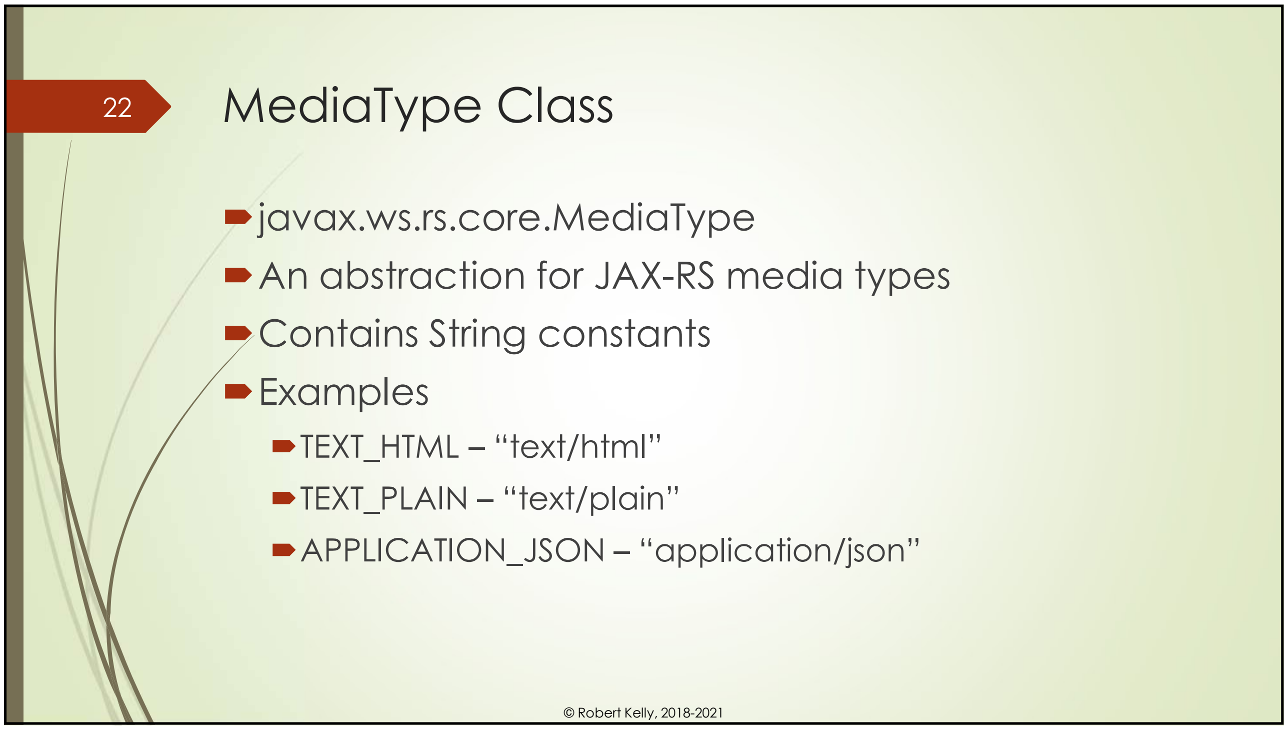
```
@Path("helloworld")
public class HelloWorld {
@Context private UriInfo context;

    public HelloWorld() {
    }
    @GET
    @Produces(MediaType.TEXT_HTML)
    public String getHtml() {
      return
  "<html><body><h1>Hello,World!!</body></h1></html>"; }
    @PUT
    @Consumes(MediaType.TEXT_HTML)
    public void putHtml(String content) {
    } }
```

Notice that you can specify the type of the return

© Robert Kelly, 2018-2021

21



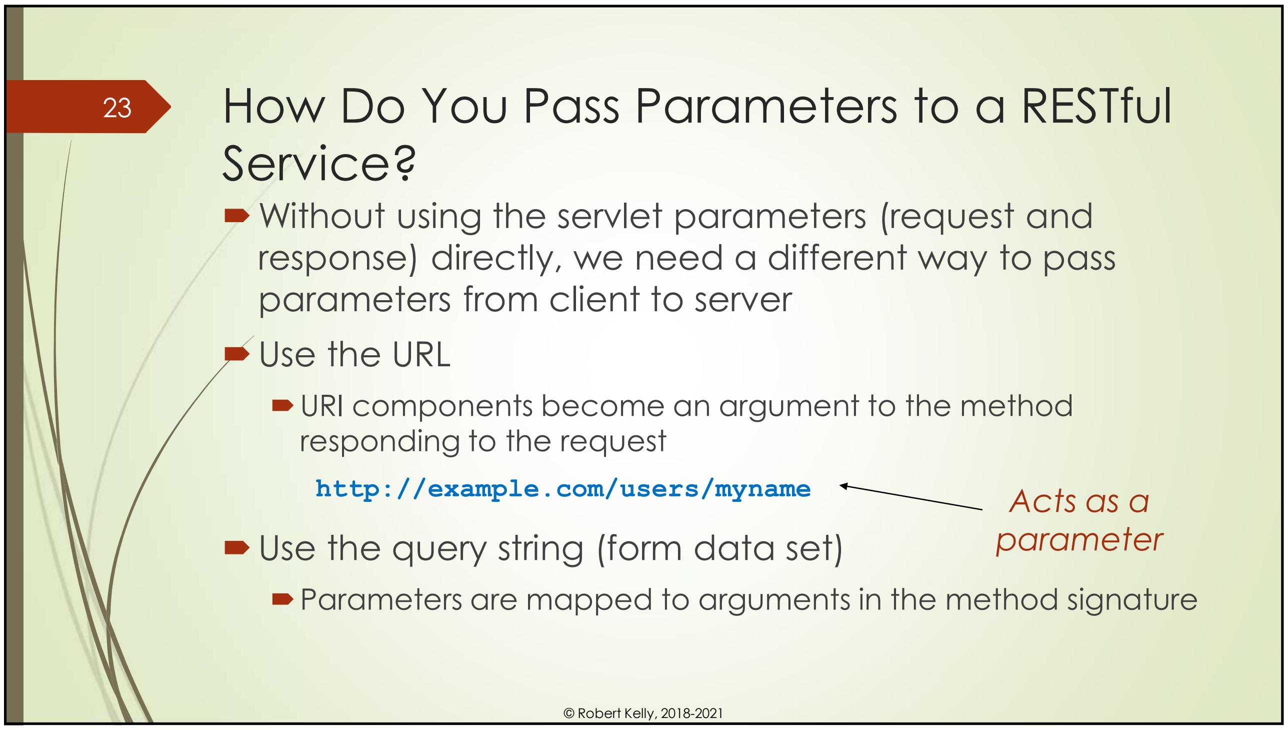
22

MediaType Class

- javax.ws.rs.core.MediaType
- An abstraction for JAX-RS media types
- Contains String constants
- Examples
  - TEXT_HTML – “text/html”
  - TEXT_PLAIN – “text/plain”
  - APPLICATION_JSON – “application/json”

© Robert Kelly, 2018-2021

22



23

How Do You Pass Parameters to a RESTful Service?

- Without using the servlet parameters (request and response) directly, we need a different way to pass parameters from client to server
- Use the URL
  - URI components become an argument to the method responding to the request

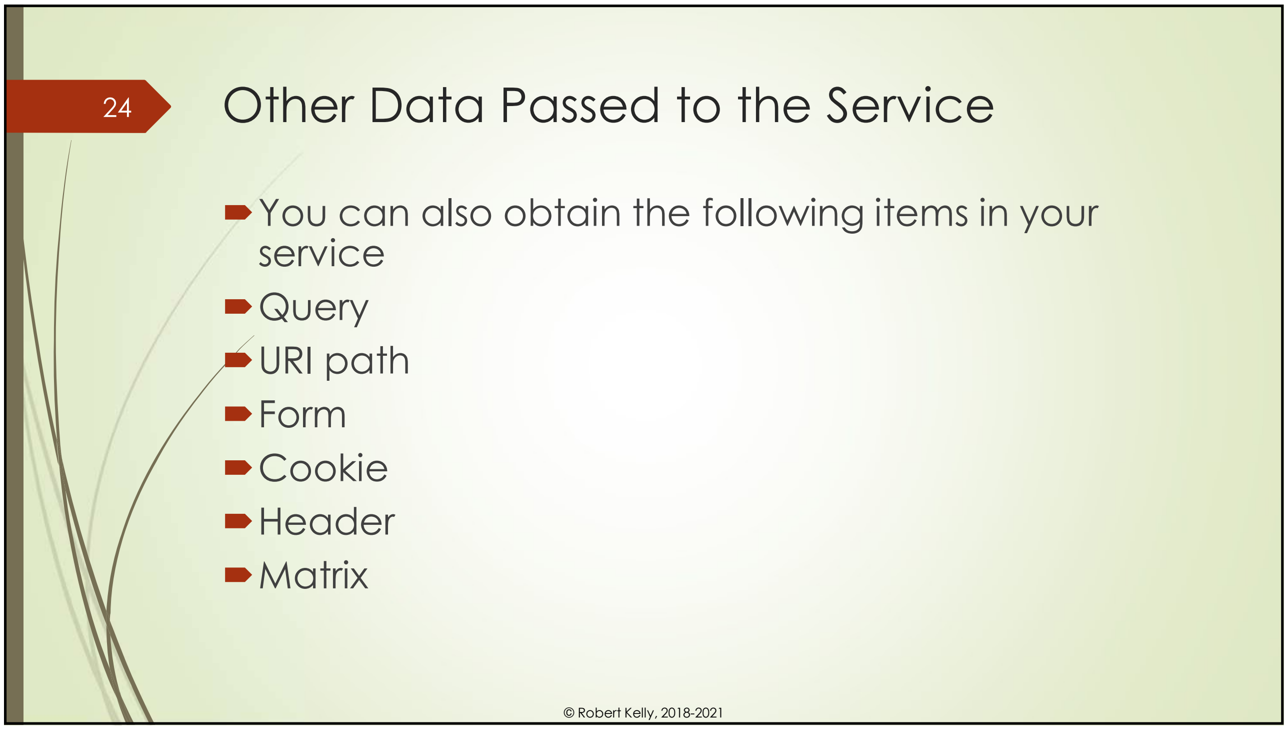
```
http://example.com/users/myname
```

Acts as a parameter

- Use the query string (form data set)
  - Parameters are mapped to arguments in the method signature

© Robert Kelly, 2018-2021

23



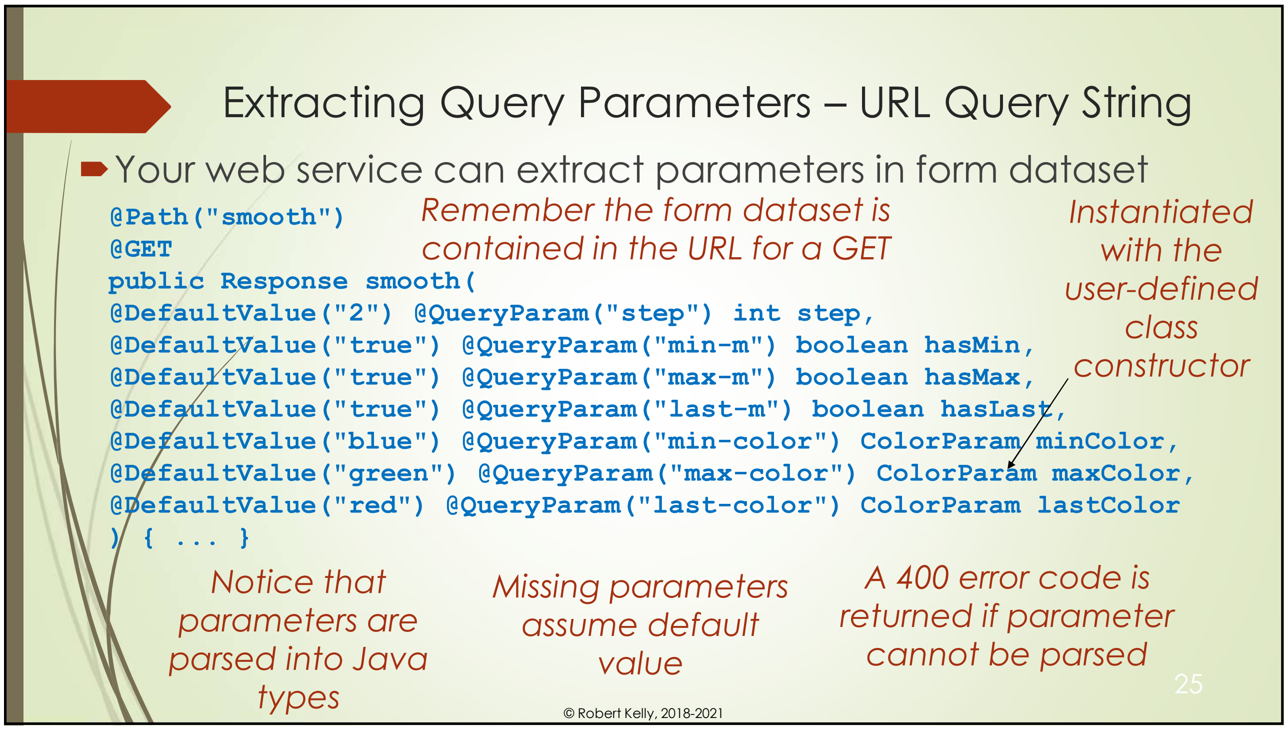
24

Other Data Passed to the Service

- You can also obtain the following items in your service
- Query
- URI path
- Form
- Cookie
- Header
- Matrix

© Robert Kelly, 2018-2021

24



Extracting Query Parameters – URL Query String

- Your web service can extract parameters in form dataset

```
@Path("smooth")
@GET
public Response smooth(
@DefaultValue("2") @QueryParam("step") int step,
@DefaultValue("true") @QueryParam("min-m") boolean hasMin,
@DefaultValue("true") @QueryParam("max-m") boolean hasMax,
@DefaultValue("true") @QueryParam("last-m") boolean hasLast,
@DefaultValue("blue") @QueryParam("min-color") ColorParam minColor,
@DefaultValue("green") @QueryParam("max-color") ColorParam maxColor,
@DefaultValue("red") @QueryParam("last-color") ColorParam lastColor
) { ... }
```

Remember the form dataset is contained in the URL for a GET

Instantiated with the user-defined class constructor

Notice that parameters are parsed into Java types

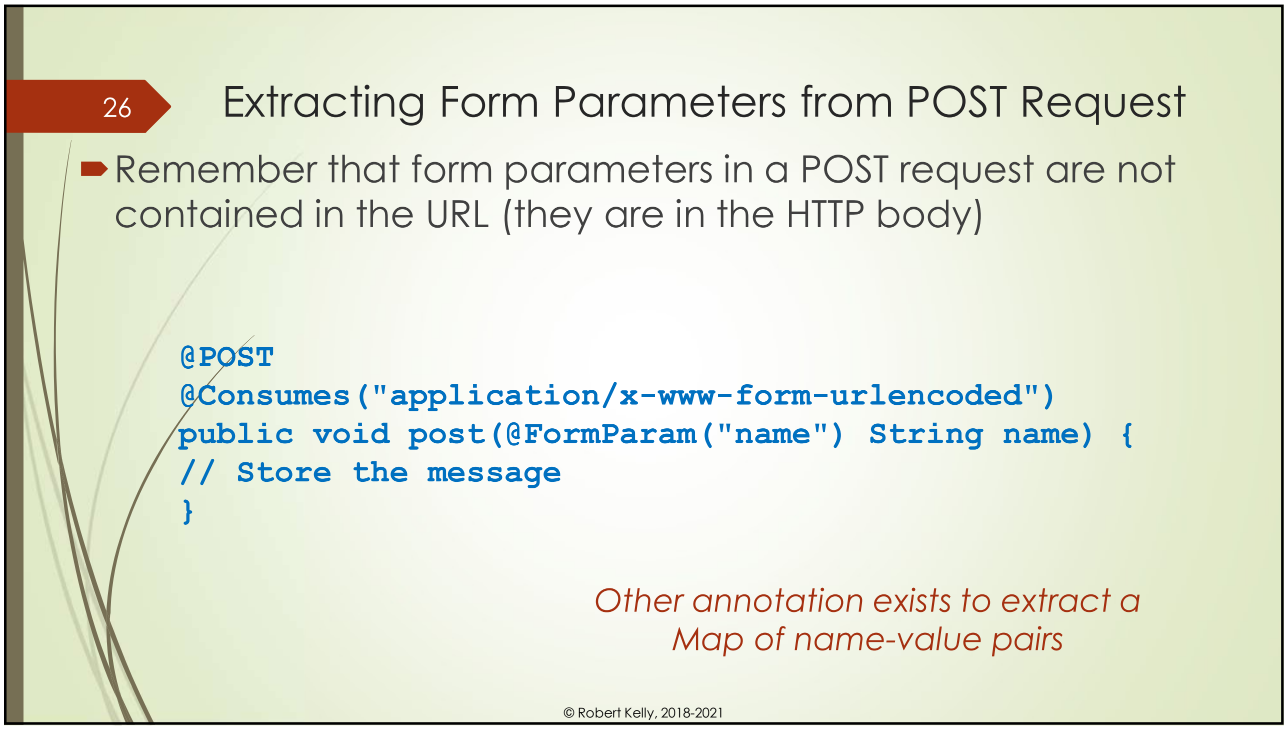
Missing parameters assume default value

A 400 error code is returned if parameter cannot be parsed

25

© Robert Kelly, 2018-2021

25



26

Extracting Form Parameters from POST Request

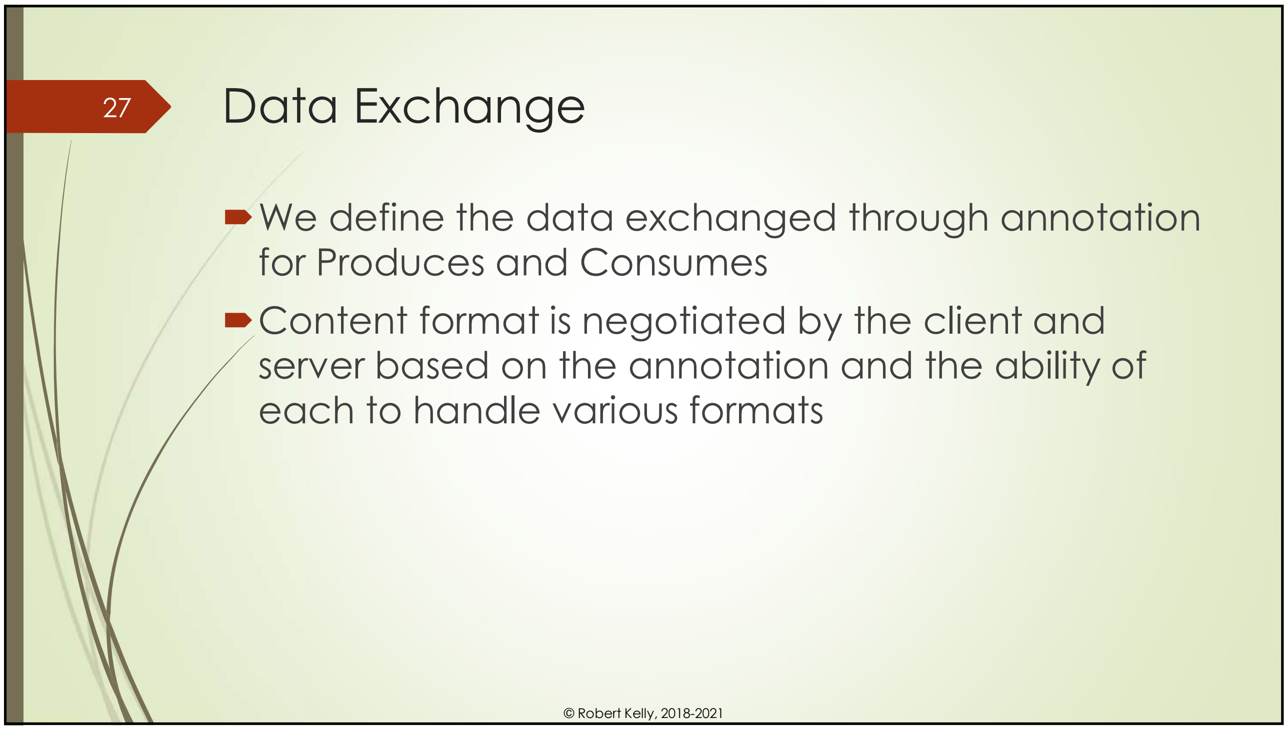
- Remember that form parameters in a POST request are not contained in the URL (they are in the HTTP body)

```
@POST
@Consumes("application/x-www-form-urlencoded")
public void post(@FormParam("name") String name) {
// Store the message
}
```

Other annotation exists to extract a Map of name-value pairs

© Robert Kelly, 2018-2021

26

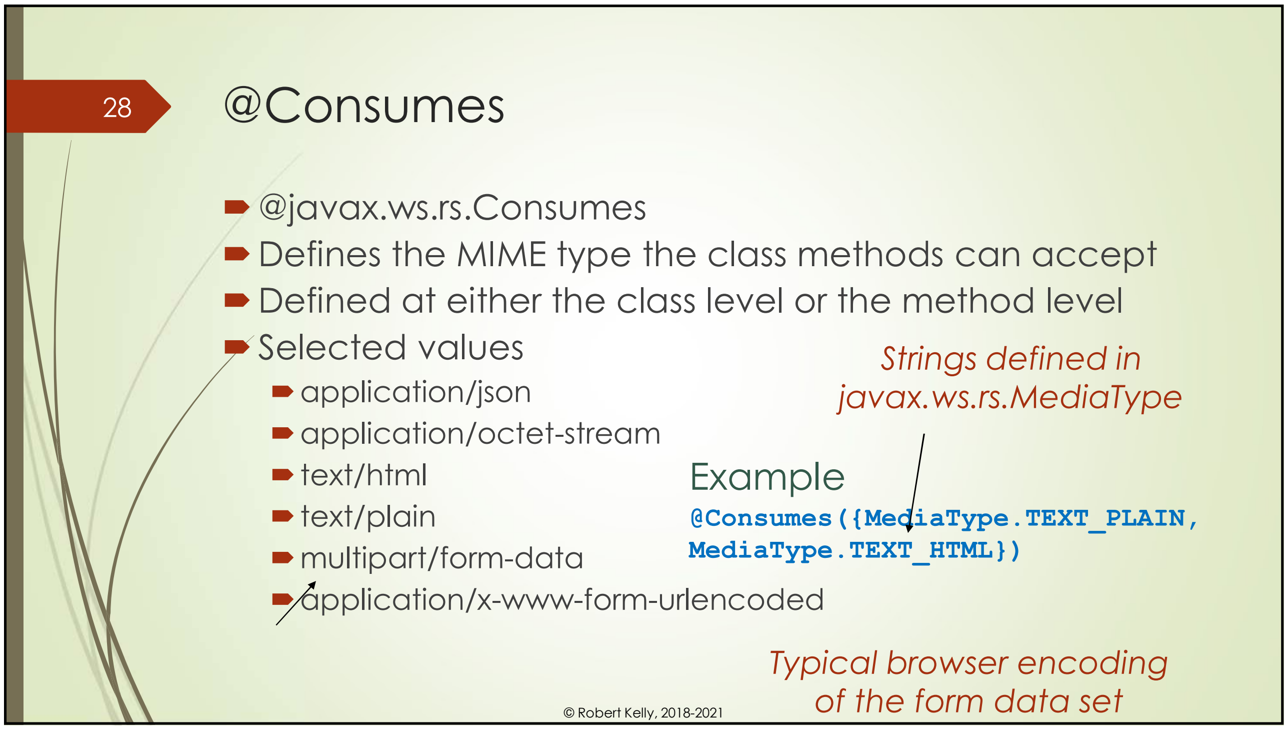


27

Data Exchange

- We define the data exchanged through annotation for Produces and Consumes
- Content format is negotiated by the client and server based on the annotation and the ability of each to handle various formats

27



28

@Consumes

- @javax.ws.rs.Consumes
- Defines the MIME type the class methods can accept
- Defined at either the class level or the method level
- Selected values
  - application/json
  - application/octet-stream
  - text/html
  - text/plain
  - multipart/form-data
  - application/x-www-form-urlencoded

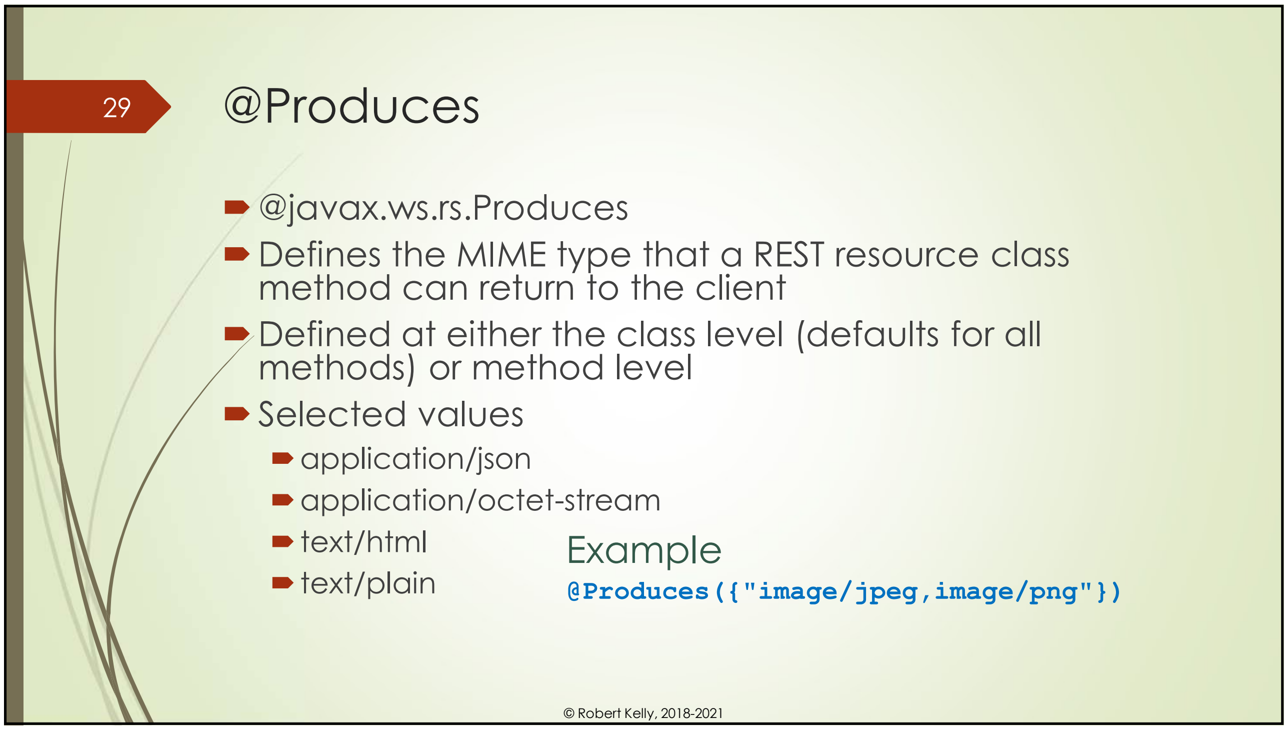
Strings defined in javax.ws.rs.MediaType

Example

```
@Consumes({MediaType.TEXT_PLAIN,
MediaType.TEXT_HTML})
```

Typical browser encoding of the form data set

28



29

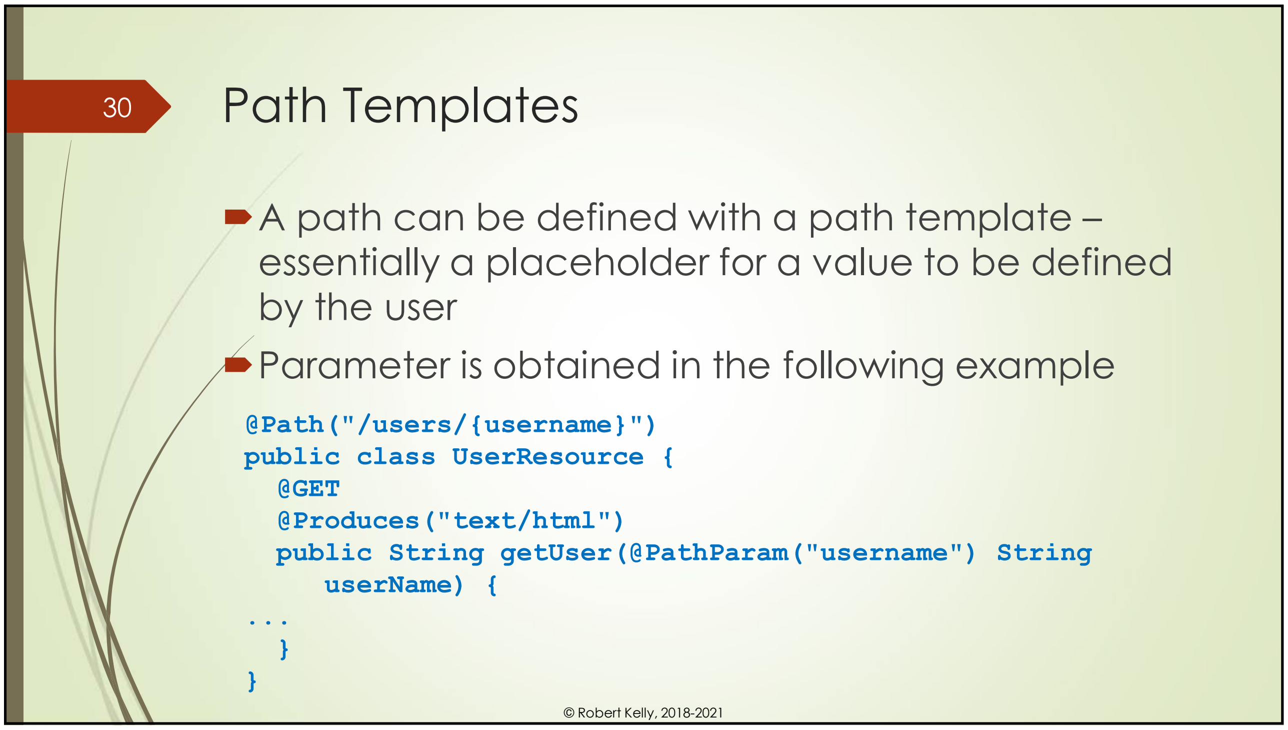
@Produces

- @javax.ws.rs.Produces
- Defines the MIME type that a REST resource class method can return to the client
- Defined at either the class level (defaults for all methods) or method level
- Selected values
  - application/json
  - application/octet-stream
  - text/html
  - text/plain

Example

```
@Produces({"image/jpeg,image/png"})
```

29



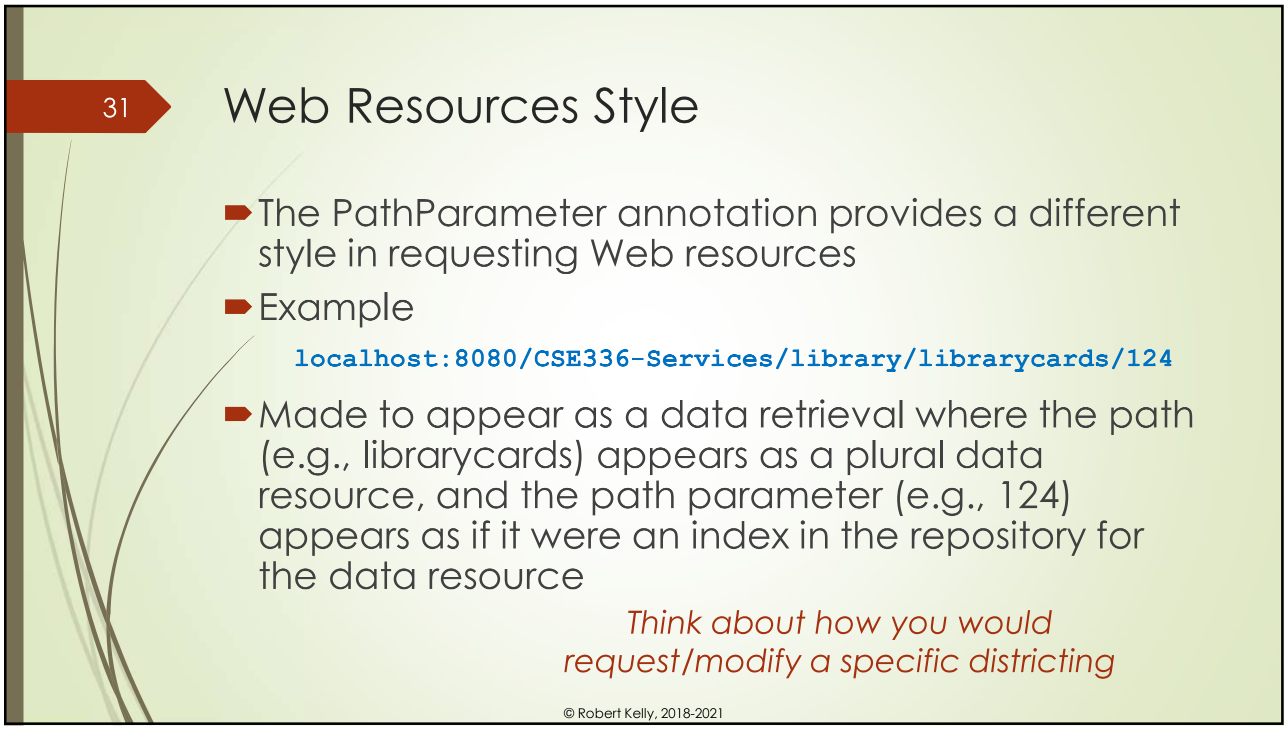
30

Path Templates

- A path can be defined with a path template – essentially a placeholder for a value to be defined by the user
- Parameter is obtained in the following example

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

30



31

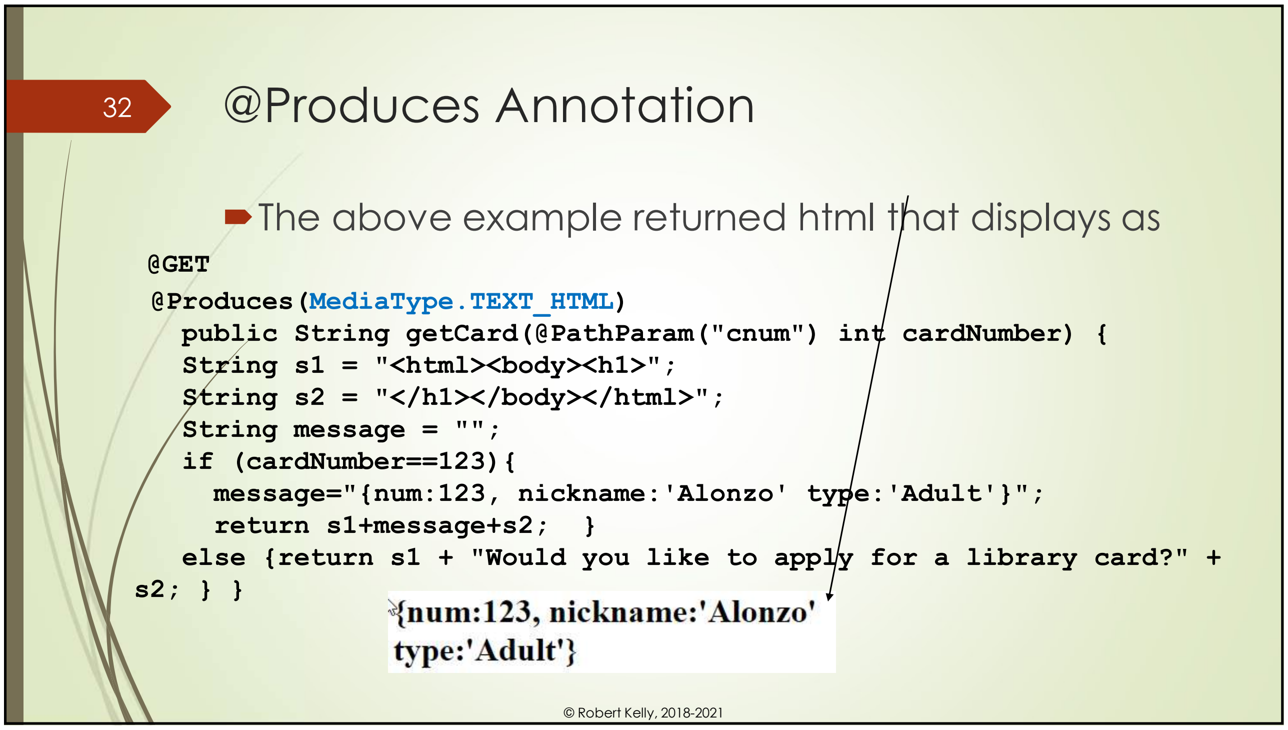
Web Resources Style

- The PathParameter annotation provides a different style in requesting Web resources
- Example

  `localhost:8080/CSE336-Services/library/librarycards/124`
- Made to appear as a data retrieval where the path (e.g., librarycards) appears as a plural data resource, and the path parameter (e.g., 124) appears as if it were an index in the repository for the data resource

Think about how you would request/modify a specific districting

31

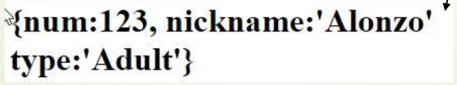


32

@Produces Annotation

- The above example returned html that displays as

```
@GET
@Produces(MediaType.TEXT_HTML)
  public String getCard(@PathParam("cnum") int cardNumber) {
  String s1 = "<html><body><h1>";
  String s2 = "</h1></body></html>";
  String message = "";
  if (cardNumber==123){
    message="{num:123, nickname:'Alonzo' type:'Adult'}";
    return s1+message+s2;  }
  else {return s1 + "Would you like to apply for a library card?" +
s2; } }
```



{num:123, nickname:'Alonzo' type:'Adult'}

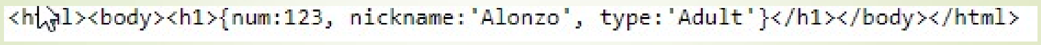
32

33

@Produces Example

- If we change the @Produces annotation, the response is not evaluated as html, and only appears as plain text

```
 @GET
  @Produces(MediaType.TEXT_Plain)
    public String getCard(@PathParam("cnum") int cardNumber)
{
```



```
<html><body><h1>{num:123, nickname:'Alonzo', type:'Adult'}</h1></body></html>
```

33

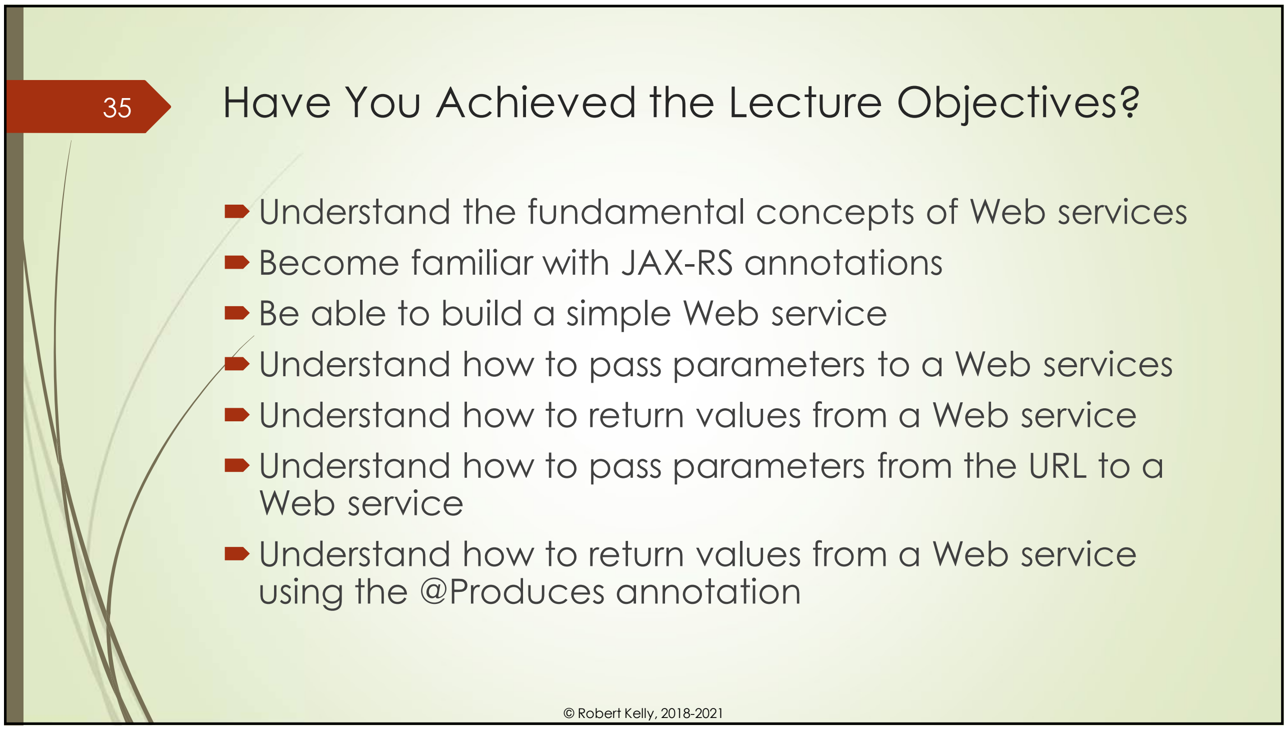
34

@Produces Example

- If we again change the @Produces annotation, when called with localhost:8080/CSE336-Services/library/librarycards/125 it returns the JSON string

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public String getCard(@PathParam("cnum") int cardNumber) {
  String s1 = "<html><body><h1>";
  String s2 = "</h1></body></html>";
  String message="{num:123, nickname:'Alonzo' type:'Adult'}";
  if (cardNumber==123){
    return s1+message+s2;
  }
  else {
    return message;
  } }
```

34



35

Have You Achieved the Lecture Objectives?

- Understand the fundamental concepts of Web services
- Become familiar with JAX-RS annotations
- Be able to build a simple Web service
- Understand how to pass parameters to a Web services
- Understand how to return values from a Web service
- Understand how to pass parameters from the URL to a Web service
- Understand how to return values from a Web service using the @Produces annotation

35