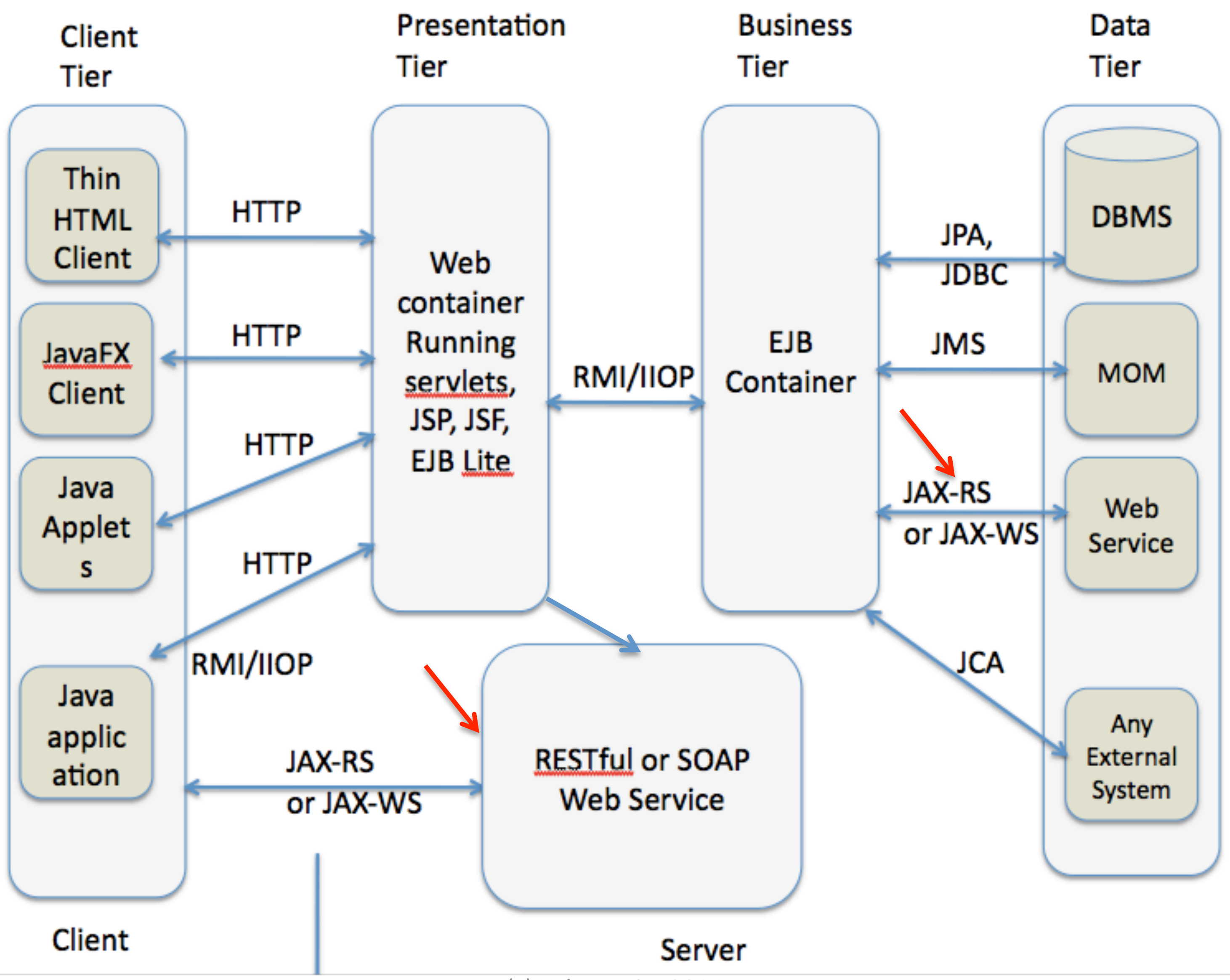
Java Programming
Unit 18

RESTful WebServices

JAX-RS is Java EE spec for REST



Why Web Services?

There is a need to expose corporate data to wider audiences, regardless of what OS and programming languages were used.

The first standard for publishing and consuming Web services was XML-based *SOAP protocol* (see http://www.w3schools.com/SOAP/soap_intro.asp).

While JSP and Servlets return both UI and data, Web Services return just data.

Some public Web services are here: http://www.programmableweb.com/apis/directory/

Java EE 7 Web Services Tutorial: http://bit.ly/1c4a3qg

Why RESTful Web Services?

REST is not a protocol, but a way of accessing data on the Web using simple HTTP calls

REST stands for *Representational State of Transfer*.

REST defines a set of constraints that an application must comply to.

It represents Web resources that the user may need to work with.

A resource is anything that you can access with a hyperlink.
Each resource has a Unified Resource Identifier (URI),
e.g. http://www.dice.com/yakovsresume.pdf

JAX-RS is Java EE API for RESTful Web Services

REST Principles (by Roy Fielding)

- Every resource on the Web has an ID (URI)
- You can link one resource to another(s)
- Use standard API: Get, Post, Put, Delete
- A resource can have multiple representations (text, JSON, pdf, etc.)
- Requests are stateless – no client-specific info is stored between requests
- Resources should be cacheable
- A REST app can be layered

REST vs. RPC-style services

With Remote Procedure Calls, a client makes one or more calls of the *arbitrary-named* methods (verbs) on the same URL.

In *RESTful services*, you always use only five verbs (HTTP methods GET, PUT, DELETE, HEAD, POST) and each resource has unique URI.

HTTP Methods

GET	Safe, Idempotent, cacheable
PUT	Idempotent
DELETE	Idempotent
HEAD	Safe, Idempotent
POST	None of the above

Idempotent: regardless of how many times a given method is invoked, the end result is the same.

GET is for retrieval, POST for creations, PUT – updates, DELETE - removals.

JAX RS 2.0

- Rest Endpoint - a POJO, typically deployed inside WAR
- Has Client API
- Message Filters and Entity Interceptors (e.g. Login Filter, encryptions et al.)
- Async processing on both client and server
- Validation

Some JAX-RS Annotations

- `@ApplicationPath` - defines the URL mapping for the application packaged in a war. It's the base URI for all @Path annotations.
- `@Path` - a root resource class that has at least one method annotated with @Path.
- `@PathParam` - injects values from request into a method parameter (e.g. Product ID)
- `@GET` - the class method that handles HTTP Get. You can have multiple methods annotated with @GET,

  and each produces different MIME type.
- `@POST` - the class method that handles HTTP Post
- `@PUT`- the class method that handles HTTP Put
- `@DELETE` - the class method that handles HTTP Delete
- `@Produces` - specifies the MIME type for response (e.g. "application/json"). The client's `Accept` header of the HTTP request declares what's acceptable.

Restful Stock Server

A Stock is an example of a resource.

Annotate a Java bean with `@XmlRootElement`, and the JAXB framework (a part of Java EE) will turn it into an XML or JSON document.

```
@XmlRootElement
public class Stock {
   private String symbol;
   private Double price;
   private String currency;
   private String country;

   public Stock() {
   }

   public Stock(String symbol,Double price, String currency,
                                                        String country) {
      this.symbol = symbol;
      this.price = price;
      this.currency = currency;
      this.country = country;
   }

   public String getSymbol() {
      return symbol;
   }

   public void setSymbol(String symbol) {
      this.symbol = symbol;
   }
…
}
```

Class StockResource

`@Path` can be used with the class and with the method. If the URI includes **/stock/IBM**, the request will be directed to class `Stock` and method `getStock()`.

`@PathParam("symbol")` injects stock symbol from URI into the arg of `getStock()`.

```
@Path("/stock")
public class StockResource {

   @Produces({"application/xml", "application/json"})
   @Path("{symbol}")
   @GET
   public Stock getStock(@PathParam("symbol") String symb) {

      Stock stock = StockService.getStock(symb);

      if (stock == null) {
         return new Stock("NOT FOUND", 0.0, "--", "--");
      }

      return stock;
   }
```

```
   @POST
   @Consumes("application/x-www-form-urlencoded")
   public Response addStock(@FormParam("symbol") String symb,
                       @FormParam("currency") String currency,
                       @FormParam("price") String price,
                       @FormParam("country") String country) {

      if (StockService.getStock(symb) != null)
         return Response.status(Response.Status.BAD_REQUEST)
           .entity("Stock " + symb + " already exists")
           .type("textplain").build();

      double priceToUse;
      try {
         priceToUse = new Double(price);
      }
      catch (NumberFormatException e) {
         priceToUse = 0.0;
      }

      StockService.addStock(new Stock(symb, priceToUse,
                                      currency, country));
      return Response.ok().build();
   }
}
```

Request URI, Get XML

If the class StockResource gets the following URI http://localhost:8080/Lesson34/resources/stock/IBM it may return resource representation as XML or JSON format based on the content of `Accept` field in HTTP request (see the annotation `@Produces` in previous slide):

```
<stock>
  <country>US</country>
  <currency>USD</currency>
  <price>43.12</price>
  <symbol>IBM</symbol>
</stock>
```

```
"stock": {
  "country": "US",
  "currency": "USD",
  "price": 43.12,
  "symbol”: "IBM"
}
```

Class StockService

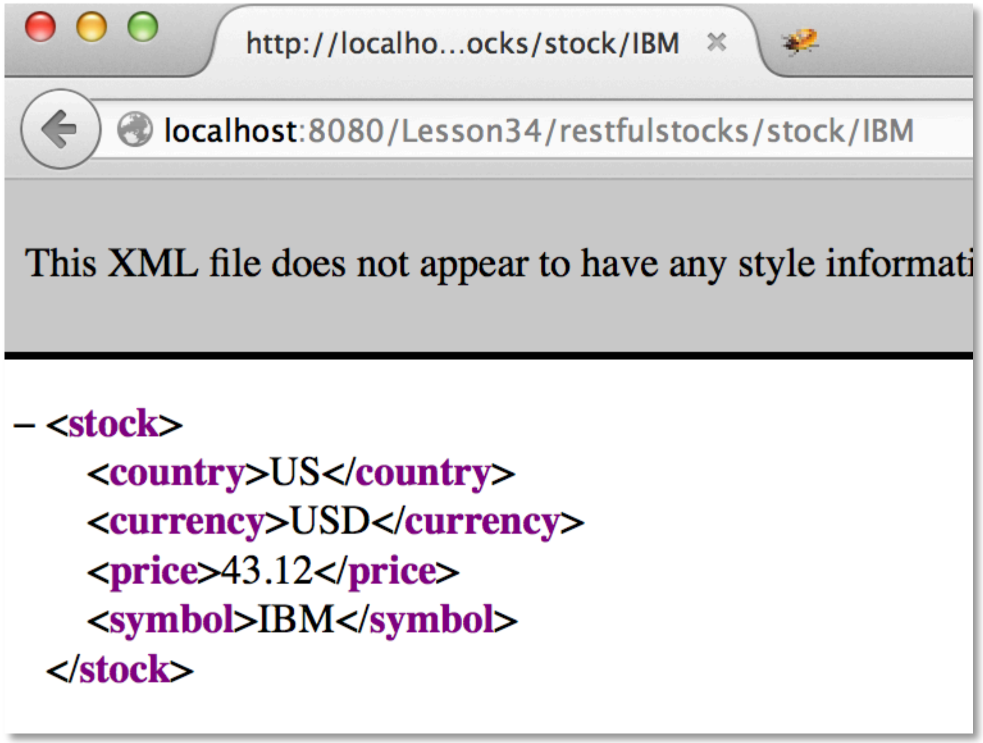
```
public class StockService {
    public static void addStock(Stock stock) {
        stocks.put(stock.getSymbol(), stock);
    }

    public static void removeStock(String symbol) {
        stocks.remove(symbol);
    }

    public static Stock getStock(String symbol) {
        return stocks.get(symbol);
    }

    private static Map<String, Stock> stocks = new HashMap<>();

    static {
        generateStocks();
    }

    private static void generateStocks() {
        addStock(new Stock("IBM", 43.12, "USD", "US"));
        addStock(new Stock("APPL", 320.0, "USD", "US"));
     }
}
```

Walkthrough (start)

- Create a dynamic web project called Lesson34 in Eclipse.
- Download and unzip *Lesson34* from http://myflex.org/yf/java/slides/Lesson34.zip
- Copy the content of the downloaded *src* folder into the *src* folder from Lesson34 project (the classes `StockApplication`, `Stock`, `StockResource`, and `StockService`).
- Deploy the project Lesson34 under GlassFish in Eclipse (the menu Add and Remove)
- Go to Lesson34 project property and in Project Facets select JAX-RS checkbox
- Start GlassFish server. The console should contain the message "Lesson34 was successfully deployed".

Walkthrough (end)

- Enter the URL http://localhost:8080/Lesson34/restfulstocks/stock. You'll get the error 405 – method not allowed.
- Enter the URL http://localhost:8080/Lesson34/restfulstocks/stock/IBM . You'll get something like this:


http://localho...ocks/stock/IBM
localhost:8080/Lesson34/restfulstocks/stock/IBM

This XML file does not appear to have any style informati

```
– <stock>
    <country>US</country>
    <currency>USD</currency>
    <price>43.12</price>
    <symbol>IBM</symbol>
  </stock>
```

- Download Postman Add-on for Chrome Browser from http://www.getpostman.com/ and learn how to use it as a test client for REST Web services.

Additional Reading

- REST Principles Explained: http://bit.ly/1exeCw9
- The HTTP methods defined: http://bit.ly/1nWmj5t
- An article on REST by Packetizer: http://bit.ly/T1cXK6

The Road Ahead

1. Install MySql Server or Oracle 11g Express Edition and migrate there all the examples that use Derby DB.
2. Evaluate MyBatis framework.
3. Pick any Java EE 7 compliant server listed at http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition (e.g. Wildfly 8) and deploy there all textbook code samples starting from Lesson 27.
4. Get familiar with Java practices: http://www.javapractices.com
5. Get familiar with Design Patterns (http://www.javacamp.org/designPattern)
6. Subscribe for the Java Specialists newsletter by Dr. Heinz Kabutz: http://www.javaspecialists.co.za

The End

For info about upcoming trainings visit
http://faratasystems.com/upcoming-training