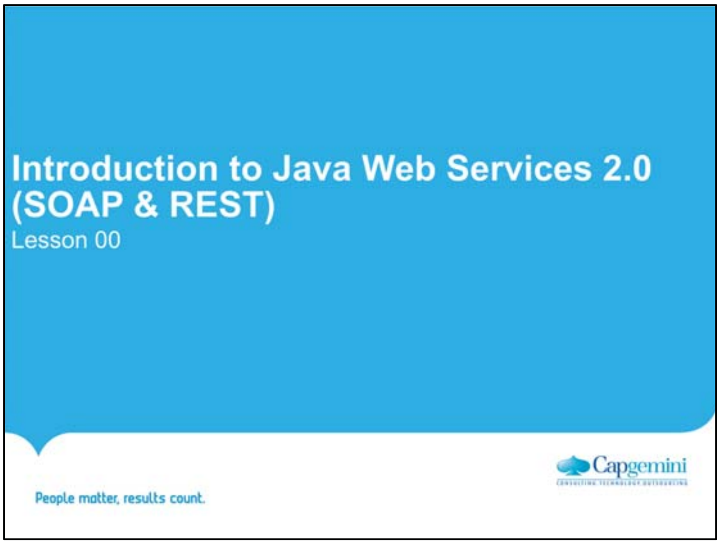
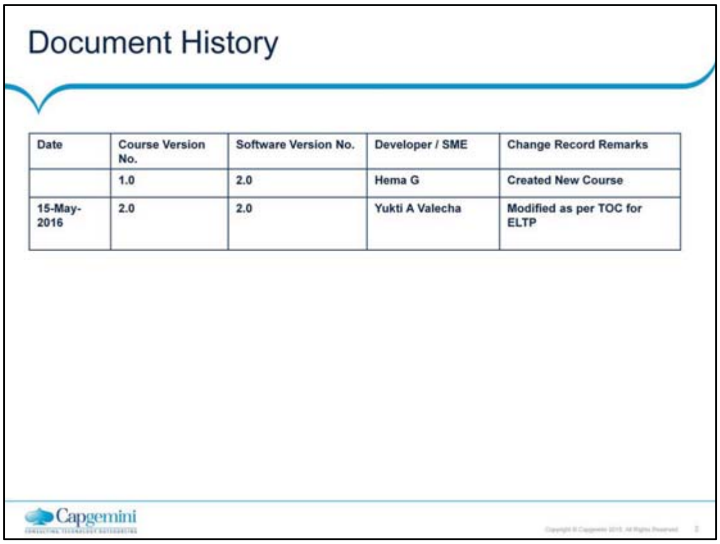


Introduction to Java Web Services 2.0 (SOAP & REST)

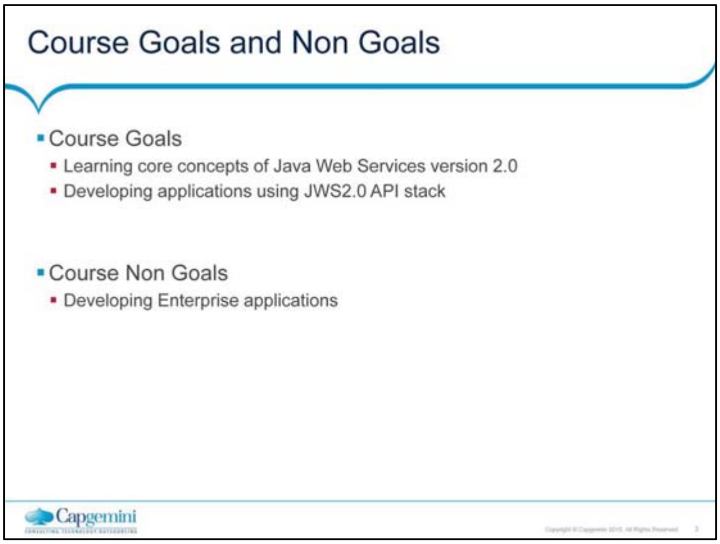
Lesson 00

People matter, results count.



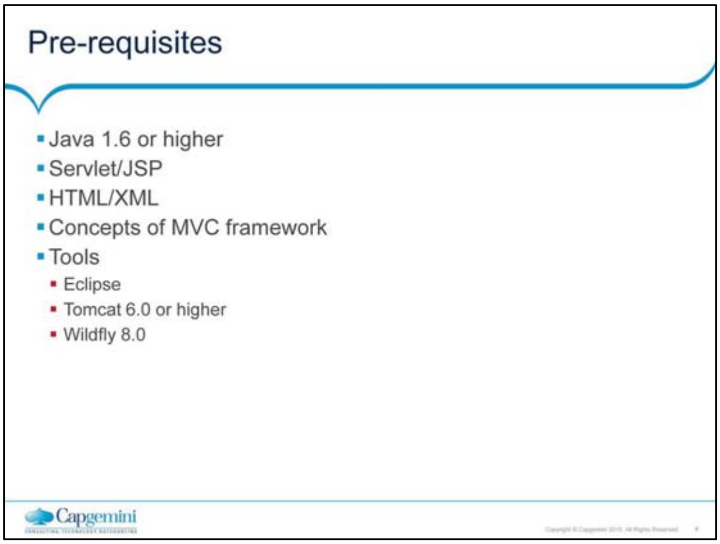
Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
	1.0	2.0	Hema G	Created New Course
15-May-2016	2.0	2.0	Yukti A Valecha	Modified as per TOC for ELTP



Course Goals and Non Goals

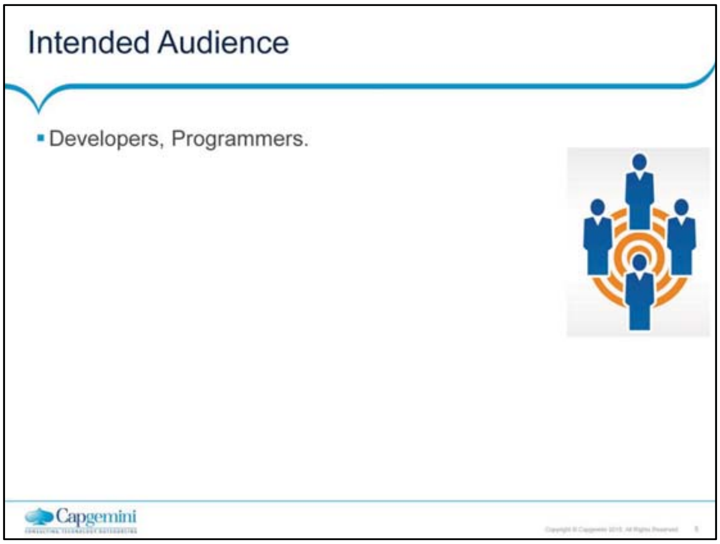
- Course Goals
  - Learning core concepts of Java Web Services version 2.0
  - Developing applications using JWS2.0 API stack
- Course Non Goals
  - Developing Enterprise applications



Pre-requisites

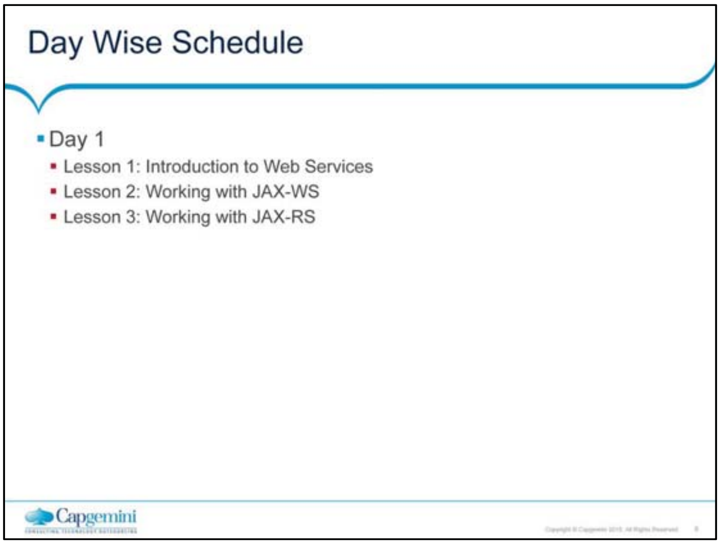
- Java 1.6 or higher
- Servlet/JSP
- HTML/XML
- Concepts of MVC framework
- Tools
  - Eclipse
  - Tomcat 6.0 or higher
  - Wildfly 8.0

Capgemini



Intended Audience

- Developers, Programmers.



Day Wise Schedule

- Day 1
  - Lesson 1: Introduction to Web Services
  - Lesson 2: Working with JAX-WS
  - Lesson 3: Working with JAX-RS

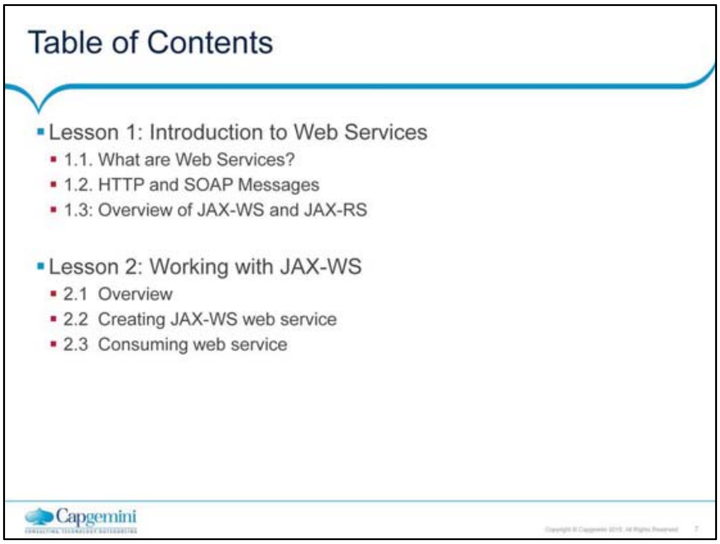


Table of Contents

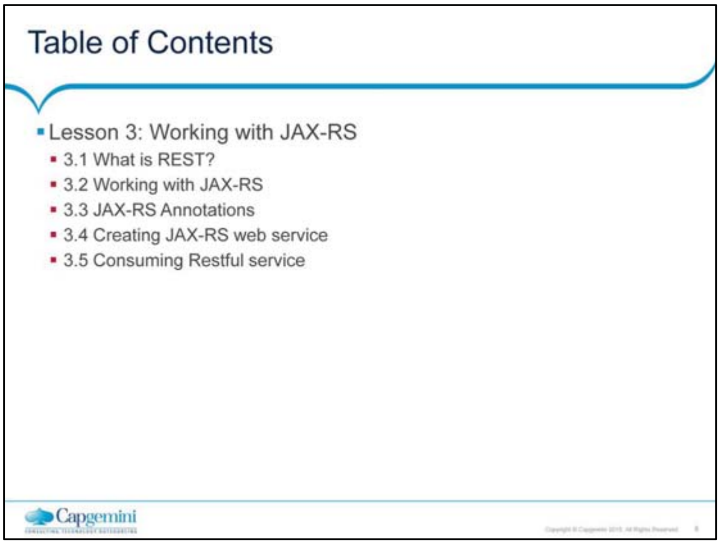
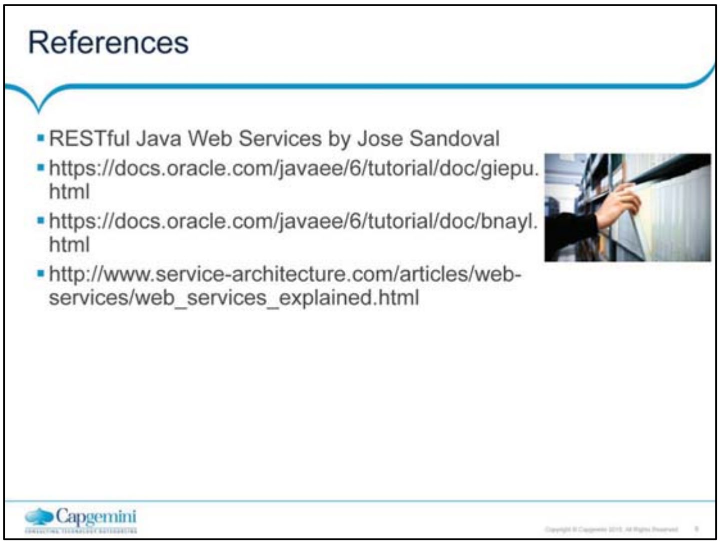
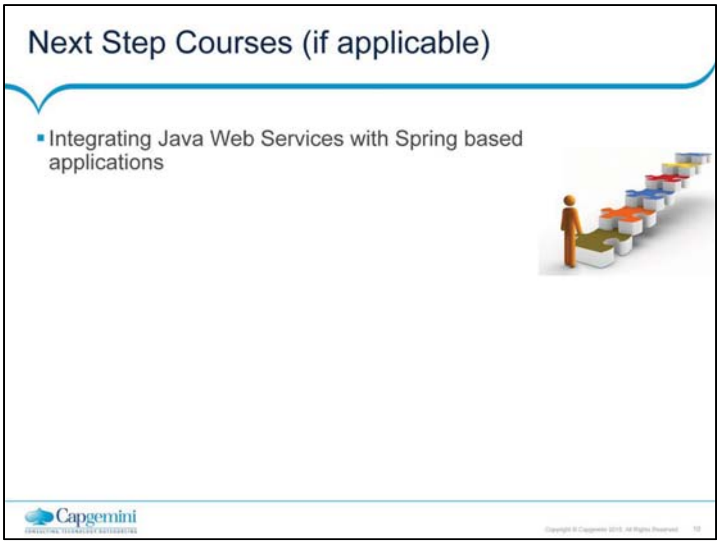


Table of Contents



References

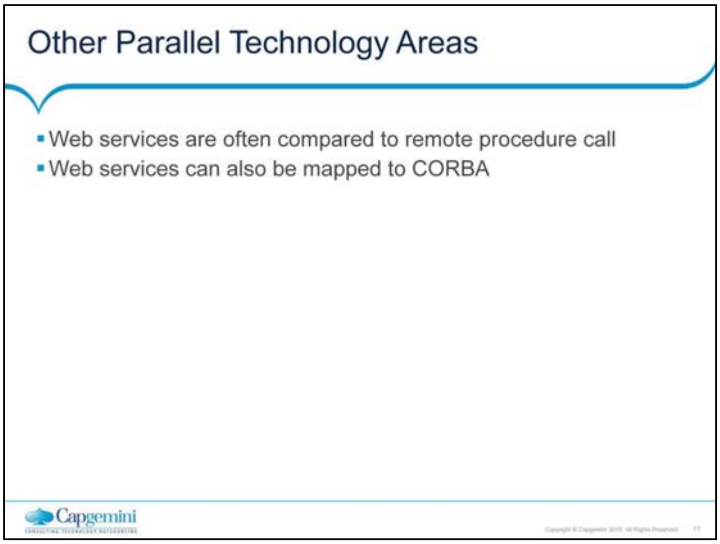
- RESTful Java Web Services by Jose Sandoval
- https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html
- https://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html
- http://www.service-architecture.com/articles/web-services/web_services_explained.html



Next Step Courses (if applicable)

- Integrating Java Web Services with Spring based applications

Capgemini

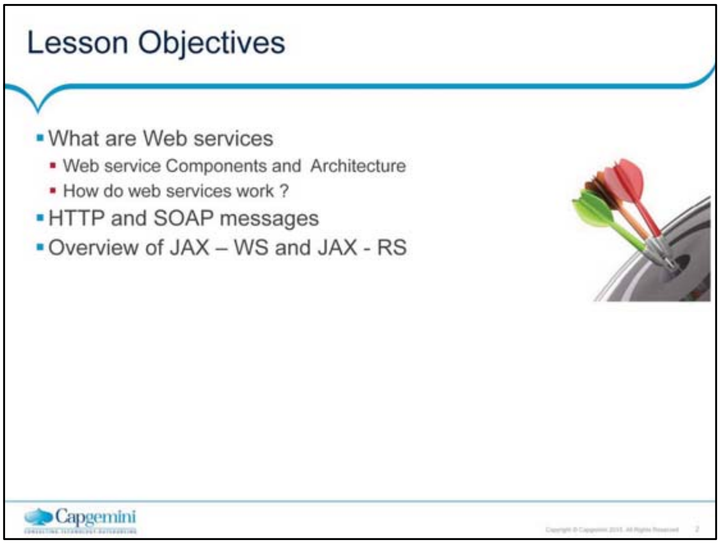


Other Parallel Technology Areas

- Web services are often compared to remote procedure call
- Web services can also be mapped to CORBA

Introduction to Web Services (SOAP & REST)

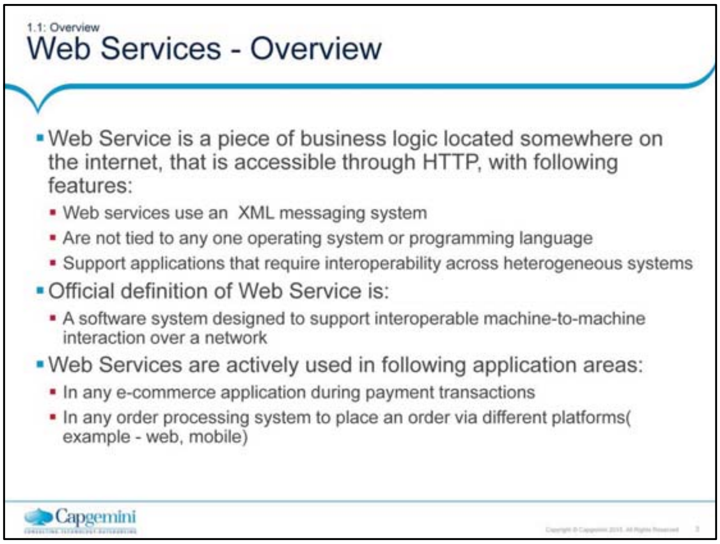
Introduction to Web Services



Lesson Objectives

- What are Web services
  - Web service Components and Architecture
  - How do web services work ?
- HTTP and SOAP messages
- Overview of JAX – WS and JAX - RS

Following contents would be covered:
1.1 : What are Web services
1.1.1 Web service components and architecture
1.1.2 How do Web services work
1.2: HTTP and SOAP messages
1.3: Overview of JAX – WS and JAX – RS



1.1: Overview

Web Services - Overview

- Web Service is a piece of business logic located somewhere on the internet, that is accessible through HTTP, with following features:
  - Web services use an XML messaging system
  - Are not tied to any one operating system or programming language
  - Support applications that require interoperability across heterogeneous systems
- Official definition of Web Service is:
  - A software system designed to support interoperable machine-to-machine interaction over a network
- Web Services are actively used in following application areas:
  - In any e-commerce application during payment transactions
  - In any order processing system to place an order via different platforms(example - web, mobile)

Capgemini

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction

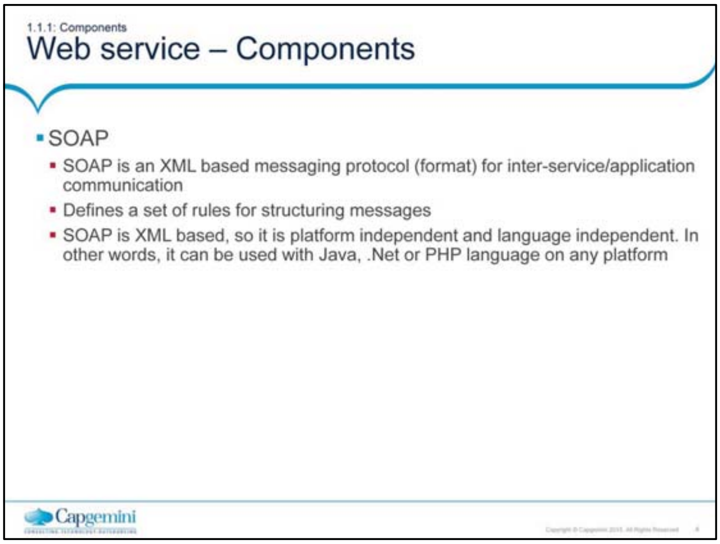
A web service is a collection of open protocols and standards (promotes interoperability between clients / servers without the need of proprietary or trademark software)

Software applications written in various programming languages and running on various platforms can use web services to exchange data. For example, interoperability between Java and Python, or Windows and Linux applications can be facilitated through web services.

Web services has the ability to go through firewalls.

Web services are available anytime, anywhere and on any device.

Web services can be used, if clients are scattered across the web.



1.1.1: Components

Web service – Components

- SOAP
  - SOAP is an XML based messaging protocol (format) for inter-service/application communication
  - Defines a set of rules for structuring messages
  - SOAP is XML based, so it is platform independent and language independent. In other words, it can be used with Java, .Net or PHP language on any platform

SOAP commonly uses HTTP.

SOAP can be used to exchange complete documents or to call a remote procedure

SOAP is a communication protocol

SOAP is for communication between applications

SOAP is a format for sending messages

SOAP is designed to communicate via Internet

SOAP is platform independent

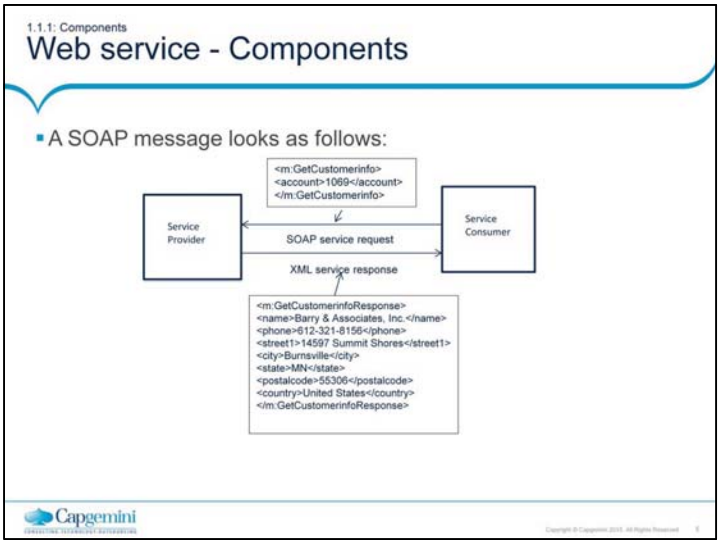
SOAP is language independent

SOAP is simple and extensible

SOAP allows you to get around firewalls

SOAP is developed as a W3C standard

Basics of WSDL will be covered in next subsequent slides



1.1.1: Components

Web service - Components

- A SOAP message looks as follows:

```
<m:GetCustomerinfo>
<account>1069</account>
</m:GetCustomerinfo>
```

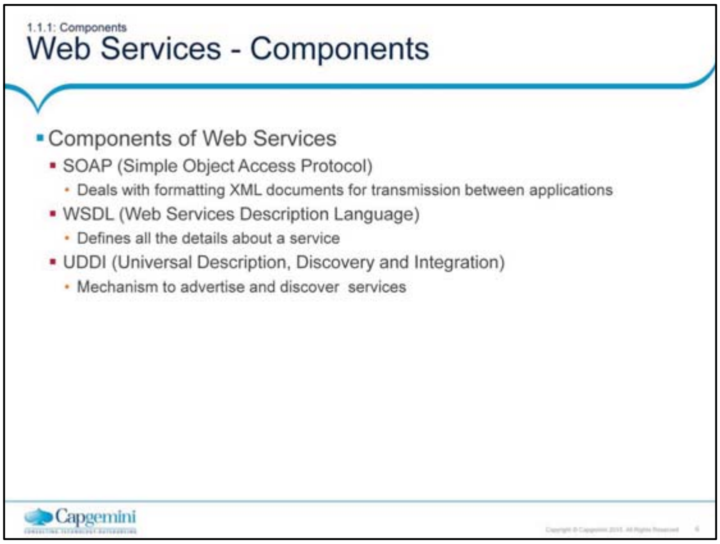
```
<m:GetCustomerinfoResponse>
<name>Barry & Associates, Inc.</name>
<phone>612-321-8156</phone>
<street1>14597 Summit Shores</street1>
<city>Burnsville</city>
<state>MN</state>
<postalcode>55306</postalcode>
<country>United States</country>
</m:GetCustomerinfoResponse>
```

Service Provider is an interface created for the user. Based on this interface, service is configured and can be called at runtime by the consumer.

Service Consumer (also possible to generate a proxy). The service can be called from a program.

Both the service consumer and producer can be configured at runtime.

Note: ‘m’ used here in diagram for SOAP request and response is a prefix denotes message passing between Service provider and Service consumer.



1.1.1: Components

Web Services - Components

- Components of Web Services
  - SOAP (Simple Object Access Protocol)
    - Deals with formatting XML documents for transmission between applications
  - WSDL (Web Services Description Language)
    - Defines all the details about a service
  - UDDI (Universal Description, Discovery and Integration)
    - Mechanism to advertise and discover services

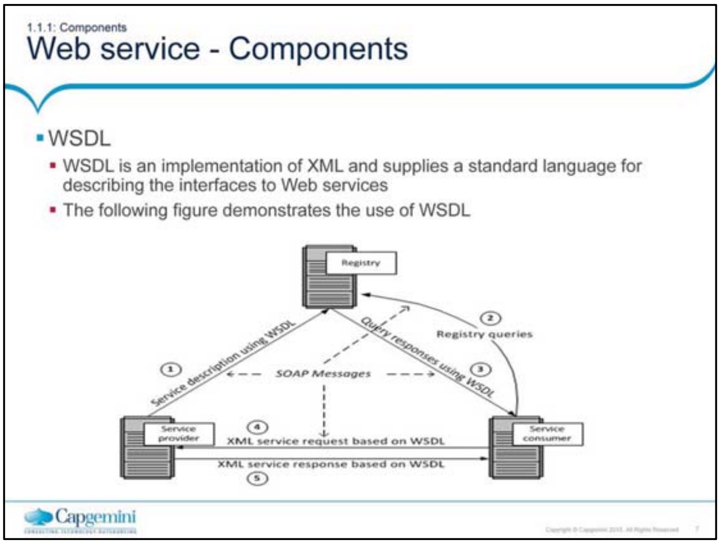
These are the components of Web services.

WSDL defines how incoming information, such as queries, need to be structured for the service application to make sense of it, and how outgoing data will be structured so that the requesting application can understand it.

These definitions are stored as XML (Extensible Markup Language) specifications

A common structure for WSDL information is the Simple Object Access Protocol (SOAP) that allows communication between applications

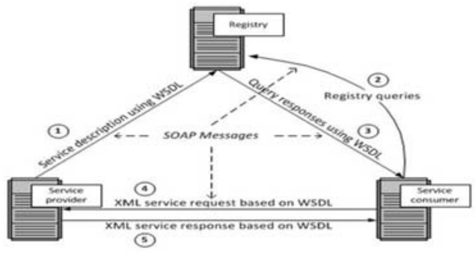
A UDDI is a directory structure to locate the web service



1.1.1: Components

Web service - Components

- WSDL
  - WSDL is an implementation of XML and supplies a standard language for describing the interfaces to Web services
  - The following figure demonstrates the use of WSDL


Registry
① Service description using WSDL
② Registry queries
③ Query responses using WSDL
SOAP Messages
Service provider
Service consumer
④ XML service request based on WSDL
XML service response based on WSDL ⑤

Capgemini

The messaging exchange format between producer and consumer is based on SOAP.

Step 1: A service provider describes its service using WSDL. This definition is published to a repository of services; which is the Universal Description, Discovery, and Integration (UDDI). This is Registry in diagram.

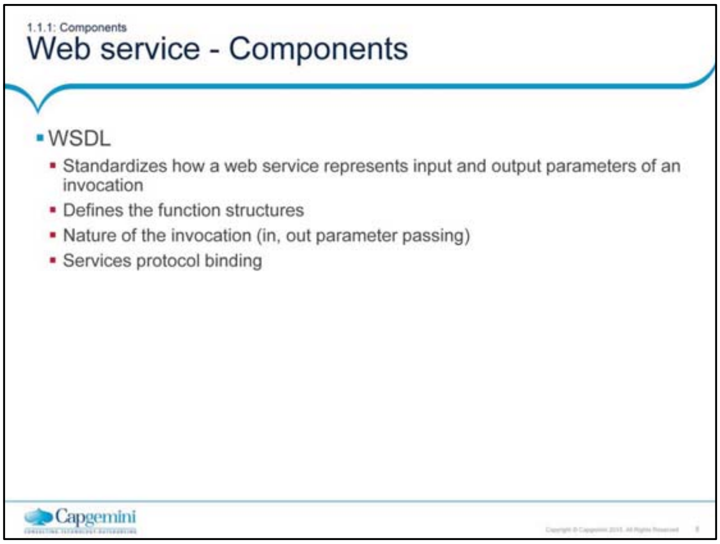
Step 2: A service consumer issues one or more queries to the repository (UDDI) to locate a service and determine how to communicate with that service

Step 3: Part of the WSDL provided by the service provider is passed to the service consumer. This tells the service consumer what are the requests and responses needed by the service provider.

Step 4: The service consumer uses the WSDL to send a request to the service provider.

Step 5: The service provider provides the expected response to the service consumer.

For example:
Web services can be used to retrieve information about books at Amazon
Similarly another web service can be used to place an order for a book to Amazon



1.1.1: Components

Web service - Components

- WSDL
  - Standardizes how a web service represents input and output parameters of an invocation
  - Defines the function structures
  - Nature of the invocation (in, out parameter passing)
  - Services protocol binding

WSDL is an acronym for Web Services Description Language.

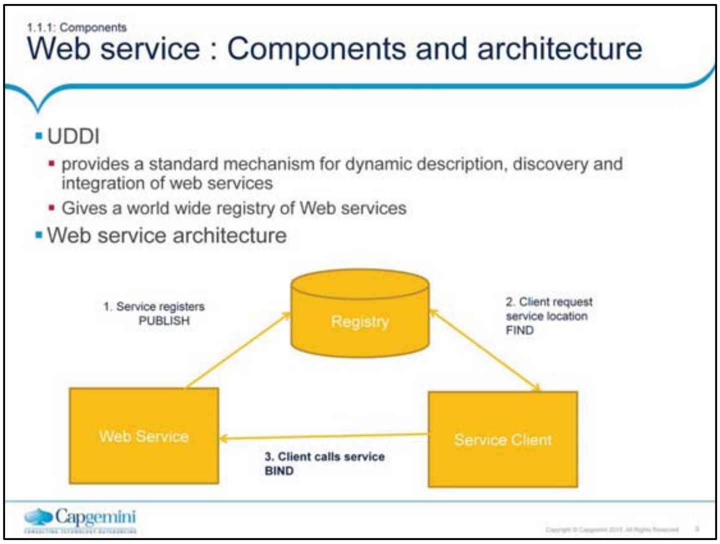
WSDL is a xml document containing information about web services such as method name, method parameter and how to access it.

WSDL is a part of UDDI. It acts as a interface between web service applications.

WSDL is pronounced as ”wiz-dull”.

More on WSDL in the next chapter.

In -> input arguments that need to be passed to a web service
Out -> output argument (return type) that a web service will return



1.1.1: Components

Web service : Components and architecture

- UDDI
  - provides a standard mechanism for dynamic description, discovery and integration of web services
  - Gives a world wide registry of Web services
- Web service architecture

Registry
Web Service
Service Client
1. Service registers PUBLISH
2. Client request service location FIND
3. Client calls service BIND
Capgemini

UDDI is an acronym for Universal Description, Discovery and Integration.

UDDI is a XML based framework for describing, discovering and integrating web services.

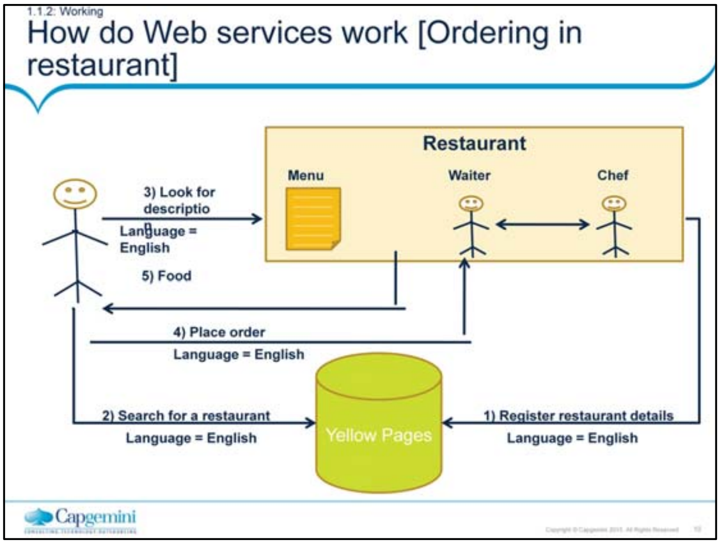
UDDI is a directory of web service interfaces described by WSDL, containing information about web services.

There are three major roles within the web service architecture:

Service Provider (Web service)
This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

Service Requestor (Service client)
This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Service Registry
This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

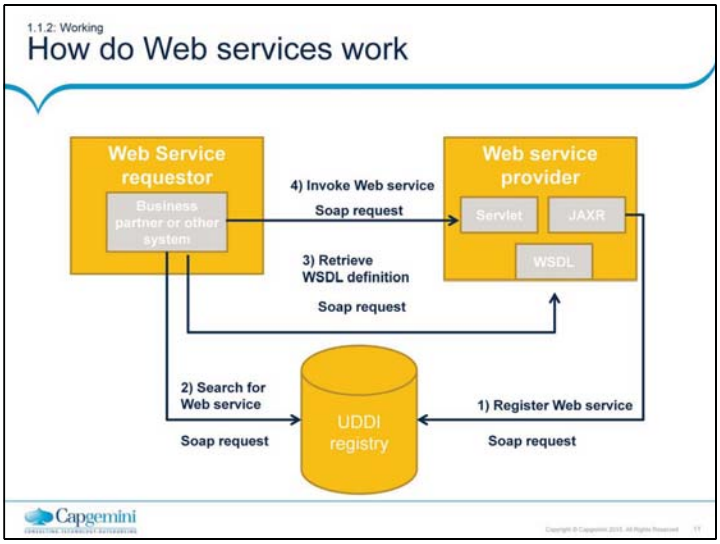


1.1.2: Working

How do Web services work [Ordering in restaurant]

Restaurant
Menu
Waiter
Chef
3) Look for descriptio Language = English
5) Food
4) Place order Language = English
2) Search for a restaurant Language = English
Yellow Pages
1) Register restaurant details Language = English
Capgemini

The restaurant would register itself under Yellow pages (Registry)
A customer would search for the restaurant under Yellow pages
Customer would then look for food options [descriptions] available in menu card (WSDL)
Customer places order (requesting for web service)
Waiter serves food to customer (response)



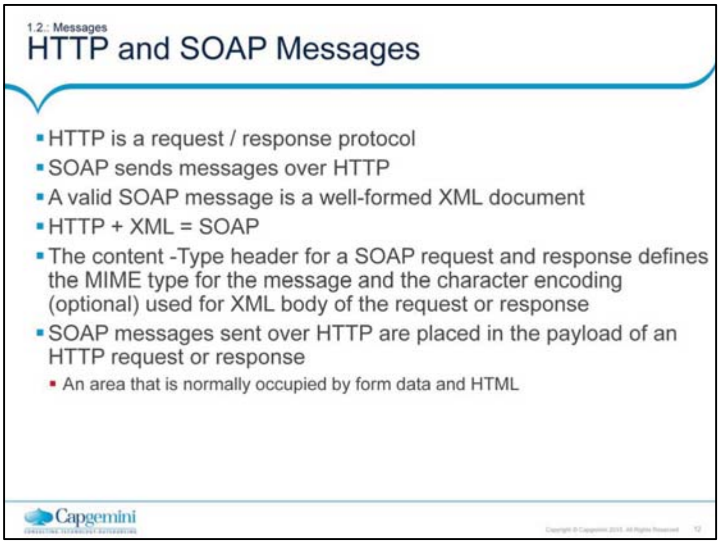
1.1.2: Working

How do Web services work

Web service provider needs to register the service under UDDI
Web service consumer (business partner, other system) would search for the web service in the registry
Web service consumer would then retrieve the web service definitions from provider (WSDL descriptions)
Now the service consumer can consume the web service given by producer

The communication between producers and consumers happen via SOAP messages.

JAXR : is one of the Java XML programming APIs. JAXR provides a uniform and standard Java API for accessing different kinds of XML-based registries, for example here it would be UDDI. Thus it would support in JAX – WS.



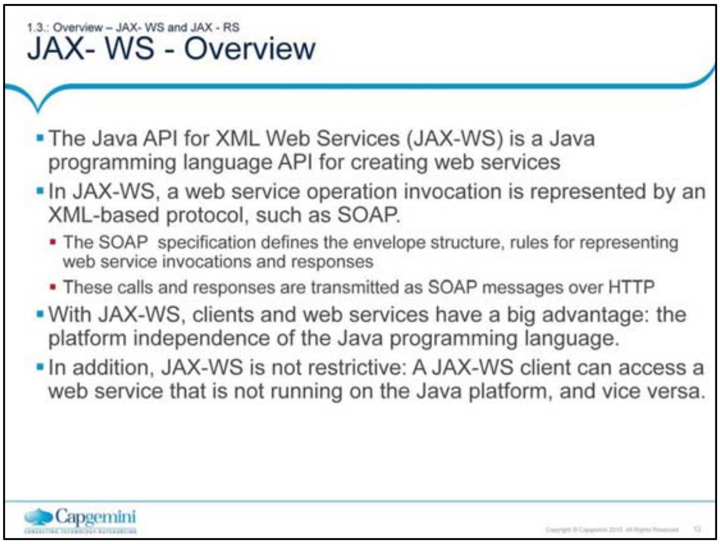
1.2.: Messages

HTTP and SOAP Messages

- HTTP is a request / response protocol
- SOAP sends messages over HTTP
- A valid SOAP message is a well-formed XML document
- HTTP + XML = SOAP
- The content -Type header for a SOAP request and response defines the MIME type for the message and the character encoding (optional) used for XML body of the request or response
- SOAP messages sent over HTTP are placed in the payload of an HTTP request or response
  - An area that is normally occupied by form data and HTML

You can serve any content over HTTP such as HTML, images, sound, video, etc. SOAP is an XML-based encoding of messages that are typically sent over HTTP.

Just like HTTP sits on top of TCP (TCP over IP), SOAP sits on top of HTTP.


1.3.: Overview – JAX- WS and JAX - RS

JAX- WS - Overview

- The Java API for XML Web Services (JAX-WS) is a Java programming language API for creating web services
- In JAX-WS, a web service operation invocation is represented by an XML-based protocol, such as SOAP.
  - The SOAP specification defines the envelope structure, rules for representing web service invocations and responses
  - These calls and responses are transmitted as SOAP messages over HTTP
- With JAX-WS, clients and web services have a big advantage: the platform independence of the Java programming language.
- In addition, JAX-WS is not restrictive: A JAX-WS client can access a web service that is not running on the Java platform, and vice versa.

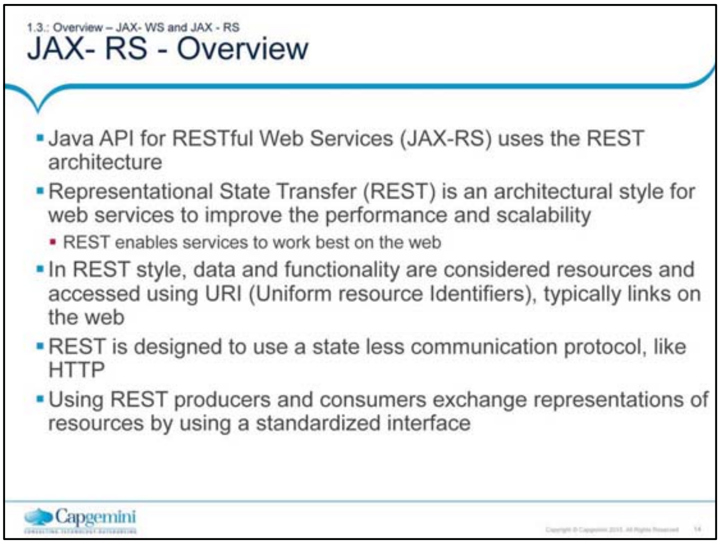
The JAX-WS API hides this complexity from the application developer. On the server side, the developer specifies the web service operations by defining methods in an interface written in the Java programming language.
The developer also codes one or more classes that implement those methods.

Client programs are also easy to code.

With JAX-WS, the developer does not generate or parse SOAP messages.

It is the JAX-WS runtime system that converts the API calls and responses to and from SOAP messages.

Details would be seen in the next subsequent chapters



1.3.: Overview – JAX- WS and JAX - RS

JAX- RS - Overview

- Java API for RESTful Web Services (JAX-RS) uses the REST architecture
- Representational State Transfer (REST) is an architectural style for web services to improve the performance and scalability
  - REST enables services to work best on the web
- In REST style, data and functionality are considered resources and accessed using URI (Uniform resource Identifiers), typically links on the web
- REST is designed to use a state less communication protocol, like HTTP
- Using REST producers and consumers exchange representations of resources by using a standardized interface

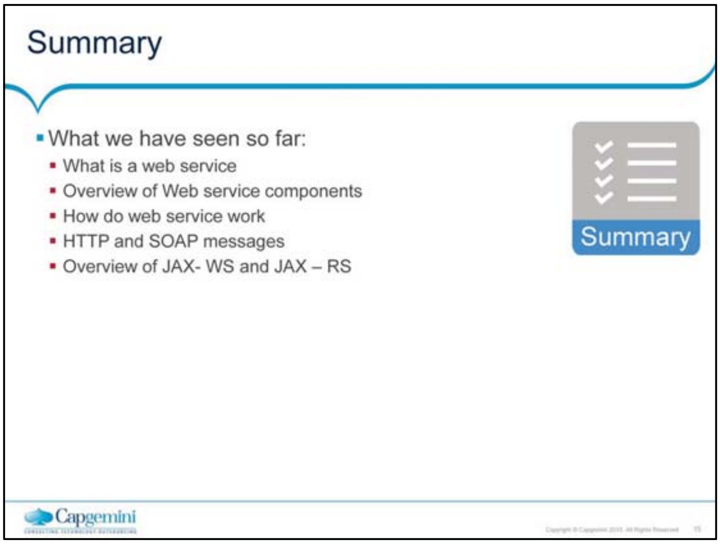
RESTful web services are built to work best on the Web.

Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web.

In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web.

The resources are acted upon by using a set of simple, well-defined operations.

REST is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

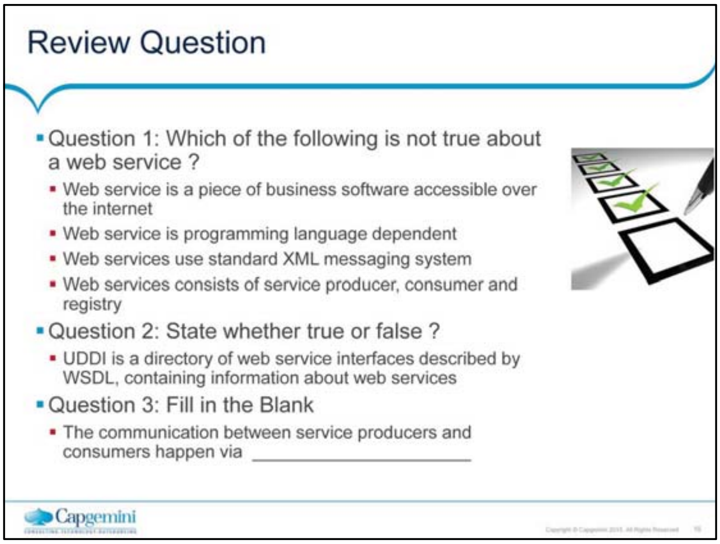


Summary

- What we have seen so far:
  - What is a web service
  - Overview of Web service components
  - How do web service work
  - HTTP and SOAP messages
  - Overview of JAX- WS and JAX – RS

Summary

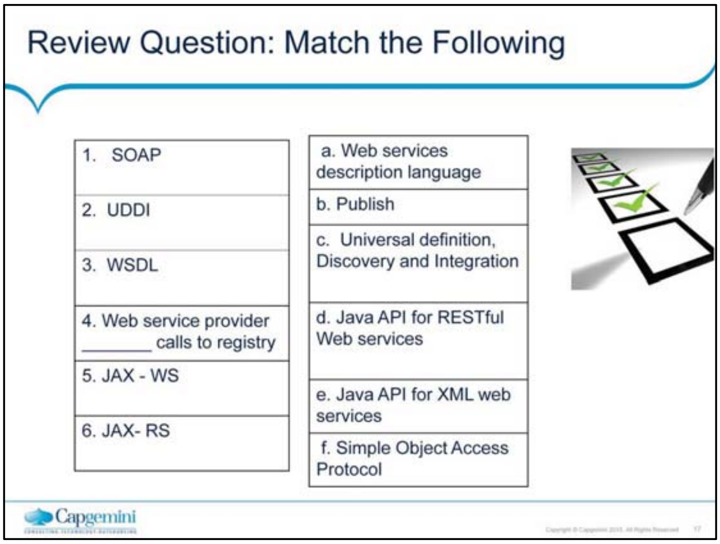
Add the notes here



Review Question

- Question 1: Which of the following is not true about a web service ?
  - Web service is a piece of business software accessible over the internet
  - Web service is programming language dependent
  - Web services use standard XML messaging system
  - Web services consists of service producer, consumer and registry
- Question 2: State whether true or false ?
  - UDDI is a directory of web service interfaces described by WSDL, containing information about web services
- Question 3: Fill in the Blank
  - The communication between service producers and consumers happen via ______________________

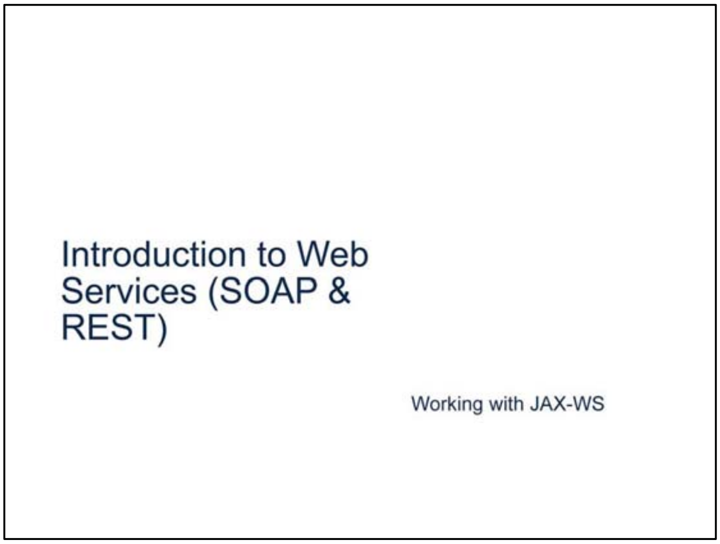
Add the notes here.



Review Question: Match the Following

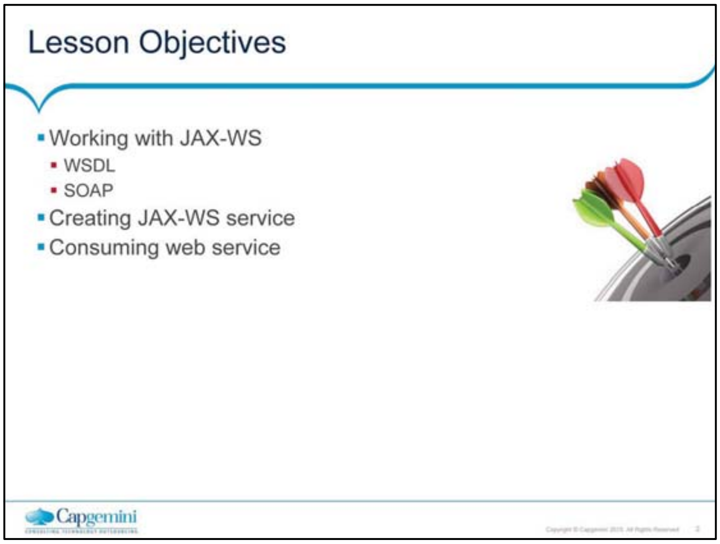
1. SOAP	a. Web services description language
2. UDDI	b. Publish
3. WSDL	c. Universal definition, Discovery and Integration
4. Web service provider ________ calls to registry	d. Java API for RESTful Web services
5. JAX - WS	e. Java API for XML web services
6. JAX- RS	f. Simple Object Access Protocol

Add the notes here.



Introduction to Web Services (SOAP & REST)

Working with JAX-WS



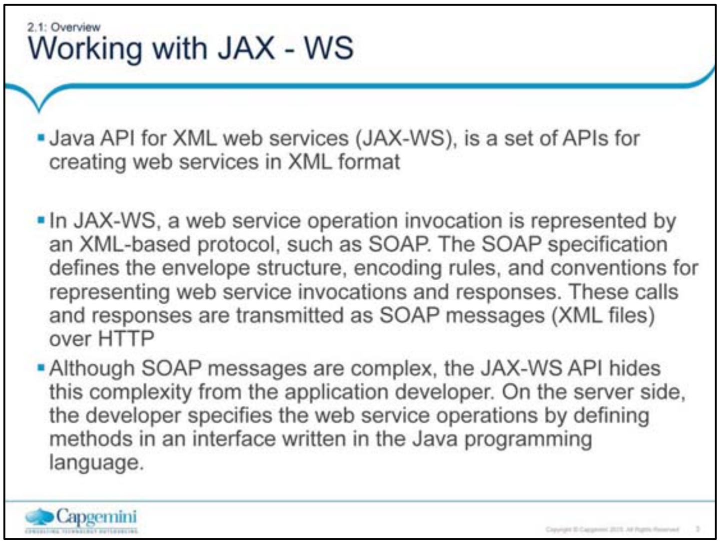
Lesson Objectives

- Working with JAX-WS
  - WSDL
  - SOAP
- Creating JAX-WS service
- Consuming web service

Capgemini

Copyright © Capgemini 2015. All Rights Reserved 2

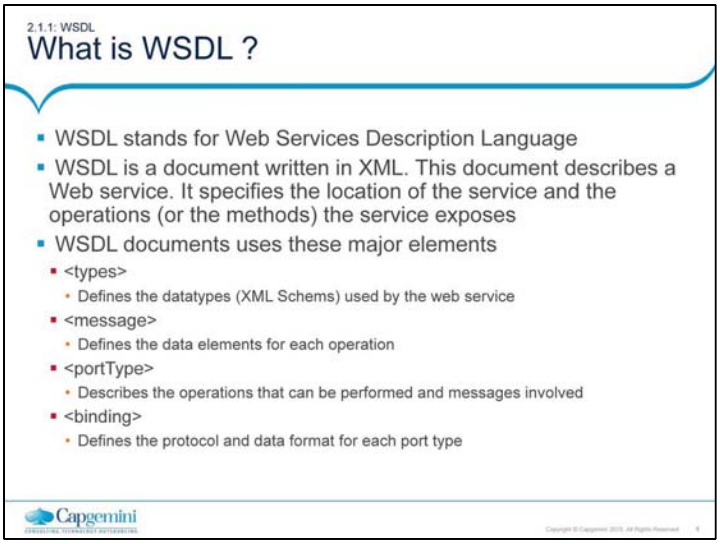
Following contents would be covered:
2.1: Working with JAX-WS
 2.1.1: What is WSDL
 2.1.2: Structure of WSDL
 2.1.3: Generating WSDL
 2.1.4: What is SOAP
 2.1.5: Structure of SOAP
2.2: Creating JAX-WS service
2.3: Consuming JAX-WS service



2.1: Overview

Working with JAX - WS

- Java API for XML web services (JAX-WS), is a set of APIs for creating web services in XML format
- In JAX-WS, a web service operation invocation is represented by an XML-based protocol, such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP
- Although SOAP messages are complex, the JAX-WS API hides this complexity from the application developer. On the server side, the developer specifies the web service operations by defining methods in an interface written in the Java programming language.



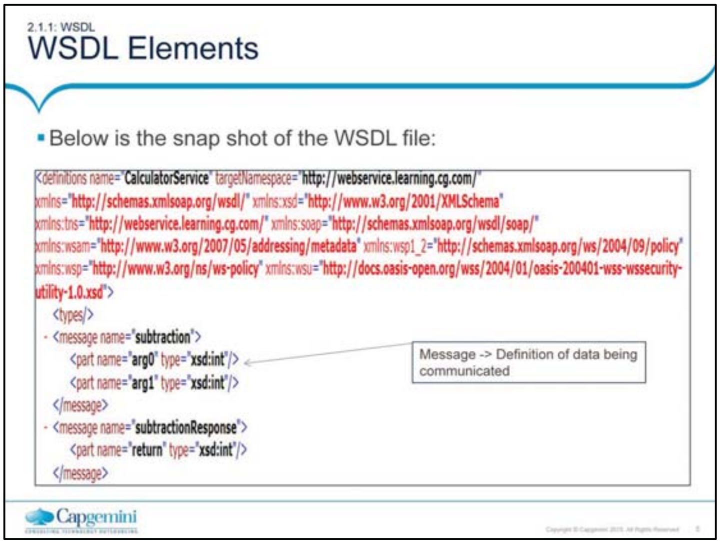
2.1.1: WSDL

What is WSDL ?

- WSDL stands for Web Services Description Language
- WSDL is a document written in XML. This document describes a Web service. It specifies the location of the service and the operations (or the methods) the service exposes
- WSDL documents uses these major elements
  - <types>
    - Defines the datatypes (XML Schems) used by the web service
  - <message>
    - Defines the data elements for each operation
  - <portType>
    - Describes the operations that can be performed and messages involved
  - <binding>
    - Defines the protocol and data format for each port type

The main structure of WSDL document looks as below:

```
<definitions>

<types>
  data type definitions........
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations......
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```



2.1.1: WSDL

WSDL Elements

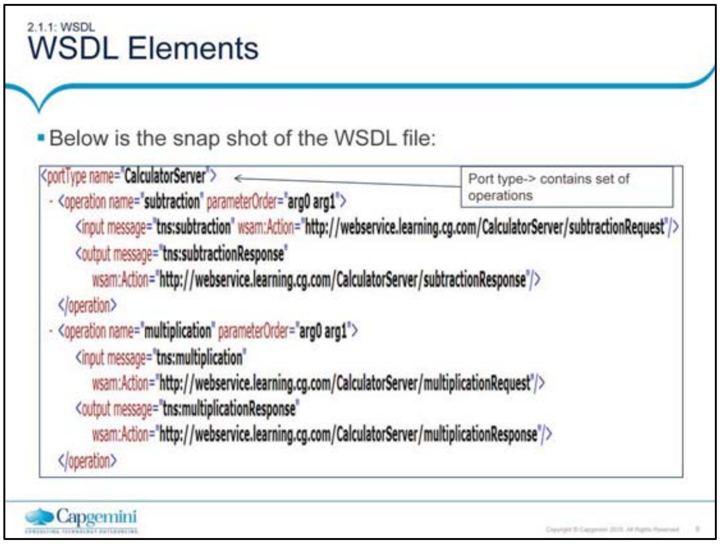
- Below is the snap shot of the WSDL file:

```
<definitions name="CalculatorService" targetNamespace="http://webservice.learning.cg.com/"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://webservice.learning.cg.com/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
   <types/>
 - <message name="subtraction">
      <part name="arg0" type="xsd:int"/>
      <part name="arg1" type="xsd:int"/>
   </message>
 - <message name="subtractionResponse">
      <part name="return" type="xsd:int"/>
   </message>
```

Message -> Definition of data being communicated

Capgemini

Add the notes here.



2.1.1: WSDL

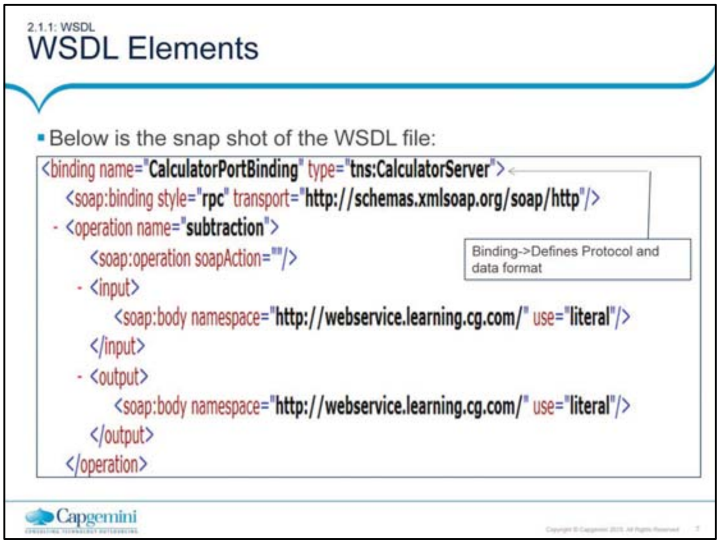
WSDL Elements

- Below is the snap shot of the WSDL file:

```
<portType name="CalculatorServer">
- <operation name="subtraction" parameterOrder="arg0 arg1">
    <input message="tns:subtraction" wsam:Action="http://webservice.learning.cg.com/CalculatorServer/subtractionRequest"/>
    <output message="tns:subtractionResponse"
      wsam:Action="http://webservice.learning.cg.com/CalculatorServer/subtractionResponse"/>
  </operation>
- <operation name="multiplication" parameterOrder="arg0 arg1">
    <input message="tns:multiplication"
      wsam:Action="http://webservice.learning.cg.com/CalculatorServer/multiplicationRequest"/>
    <output message="tns:multiplicationResponse"
      wsam:Action="http://webservice.learning.cg.com/CalculatorServer/multiplicationResponse"/>
  </operation>
```

Port type-> contains set of operations

Add the notes here.



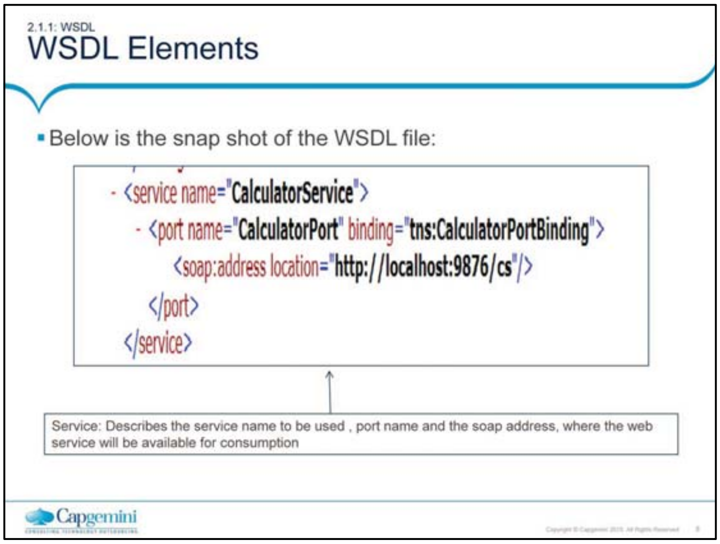
2.1.1: WSDL

WSDL Elements

- Below is the snap shot of the WSDL file:

```
<binding name="CalculatorPortBinding" type="tns:CalculatorServer">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
- <operation name="subtraction">
    <soap:operation soapAction=""/>
  - <input>
      <soap:body namespace="http://webservice.learning.cg.com/" use="literal"/>
    </input>
  - <output>
      <soap:body namespace="http://webservice.learning.cg.com/" use="literal"/>
    </output>
  </operation>
```

Add the notes here.



2.1.1: WSDL

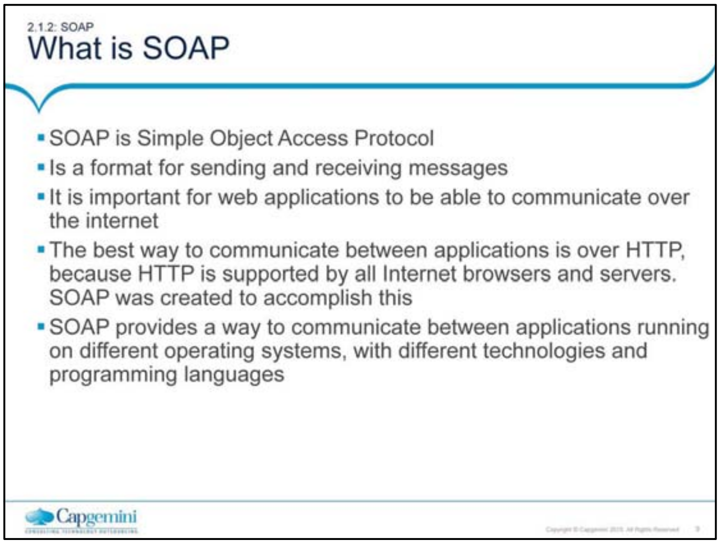
WSDL Elements

- Below is the snap shot of the WSDL file:

```
- <service name="CalculatorService">
  - <port name="CalculatorPort" binding="tns:CalculatorPortBinding">
      <soap:address location="http://localhost:9876/cs"/>
    </port>
  </service>
```

Service: Describes the service name to be used , port name and the soap address, where the web service will be available for consumption

Add the notes here.

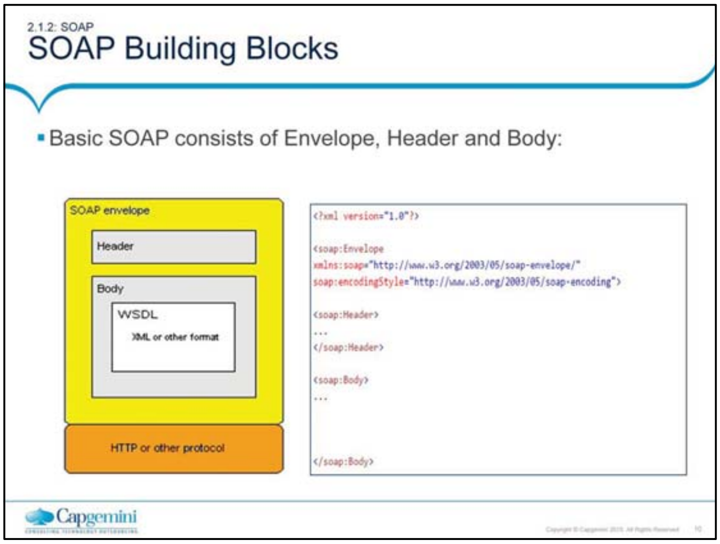


2.1.2: SOAP

What is SOAP

- SOAP is Simple Object Access Protocol
- Is a format for sending and receiving messages
- It is important for web applications to be able to communicate over the internet
- The best way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this
- SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages

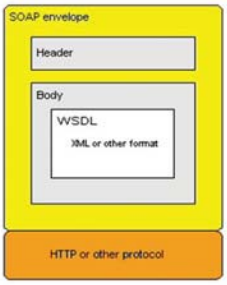
SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol
SOAP is an application communication protocol
SOAP is a format for sending and receiving messages
SOAP is platform independent
SOAP is based on XML
SOAP is a W3C recommendation



2.1.2: SOAP

SOAP Building Blocks

- Basic SOAP consists of Envelope, Header and Body:

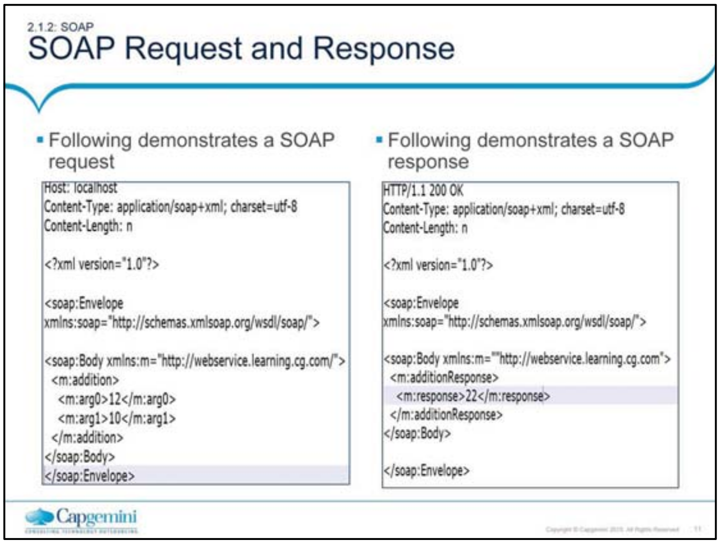


```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...


</soap:Body>
```

A SOAP message is an ordinary XML document containing the following elements:
An Envelope element that identifies the XML document as a SOAP message
A Header element that contains header information
A Body element that contains call and response information

A few syntax rules…..

A SOAP message MUST be encoded using XML
A SOAP message MUST use the SOAP Envelope namespace
A SOAP message MUST use the SOAP Encoding namespace
A SOAP message must NOT contain a DTD reference
A SOAP message must NOT contain XML Processing Instructions



2.1.2: SOAP

SOAP Request and Response

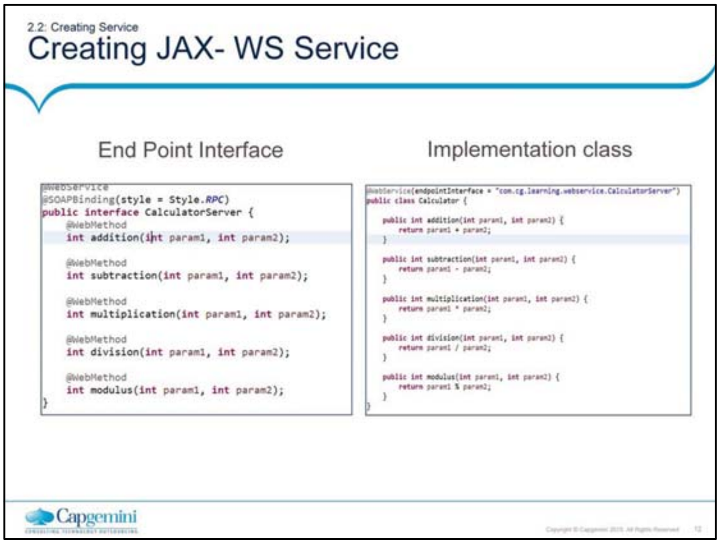
- Following demonstrates a SOAP request

```
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: n

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

<soap:Body xmlns:m="http://webservice.learning.cg.com/">
 <m:addition>
  <m:arg0>12</m:arg0>
  <m:arg1>10</m:arg1>
 </m:addition>
</soap:Body>
</soap:Envelope>
```

- Following demonstrates a SOAP response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: n

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

<soap:Body xmlns:m=""http://webservice.learning.cg.com">
 <m:additionResponse>
  <m:response>22</m:response>
 </m:additionResponse>
</soap:Body>

</soap:Envelope>
```

In the example above, an addition request is sent to a server. The request has 2 parameters as number1 and number2 and addition of 2 numbers is returned in the response. The namespace for the function is defined in "http://webservice.learning.cg.com/".



2.2: Creating Service

Creating JAX- WS Service

End Point Interface

```
@WebService
@SOAPBinding(style = Style.RPC)
public interface CalculatorServer {
    @WebMethod
    int addition(int param1, int param2);

    @WebMethod
    int subtraction(int param1, int param2);

    @WebMethod
    int multiplication(int param1, int param2);

    @WebMethod
    int division(int param1, int param2);

    @WebMethod
    int modulus(int param1, int param2);
}
```

Implementation class

```
@WebService(endpointInterface = "com.cg.learning.webservice.CalculatorServer")
public class Calculator {

    public int addition(int param1, int param2) {
        return param1 + param2;
    }

    public int subtraction(int param1, int param2) {
        return param1 - param2;
    }

    public int multiplication(int param1, int param2) {
        return param1 * param2;
    }

    public int division(int param1, int param2) {
        return param1 / param2;
    }

    public int modulus(int param1, int param2) {
        return param1 % param2;
    }
}
```

The starting point for developing a JAX-WS web service is a Java class annotated with the javax.jws.WebService annotation.

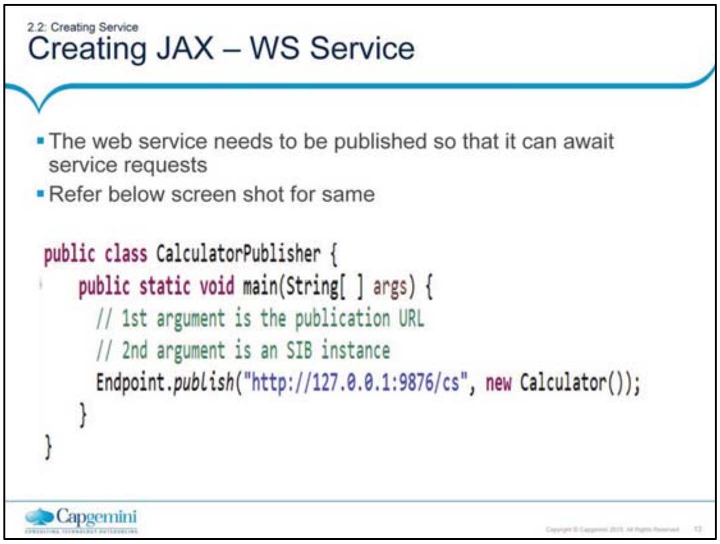
The WebService annotation defines the class as a web service endpoint. SOAP binding style is given as RPC.

A *service endpoint interface* (SEI) is a Java interface that declares the methods that a client can invoke on the service

The web service implementation class implicitly defines a SEI.

You may specify an explicit SEI by adding the endpointInterface element to the WebService annotation in the implementation class.

You must then provide a SEI that defines the public methods made available in the endpoint implementation class.



2.2: Creating Service

Creating JAX – WS Service

- The web service needs to be published so that it can await service requests
- Refer below screen shot for same

```
public class CalculatorPublisher {
    public static void main(String[ ] args) {
      // 1st argument is the publication URL
      // 2nd argument is an SIB instance
      Endpoint.publish("http://127.0.0.1:9876/cs", new Calculator());
    }
}
```

Currently the service is published at network address 127.0.0.1.which is localhost, and at port number 9876,

The application path /cs is an arbitary name.

The Endpoint class has an overloaded publish method. In this two-argument version, the first argument is the publication URL as a string and the second argument is an instance of the service SIB, in this case com.cg.learning.Calculator.

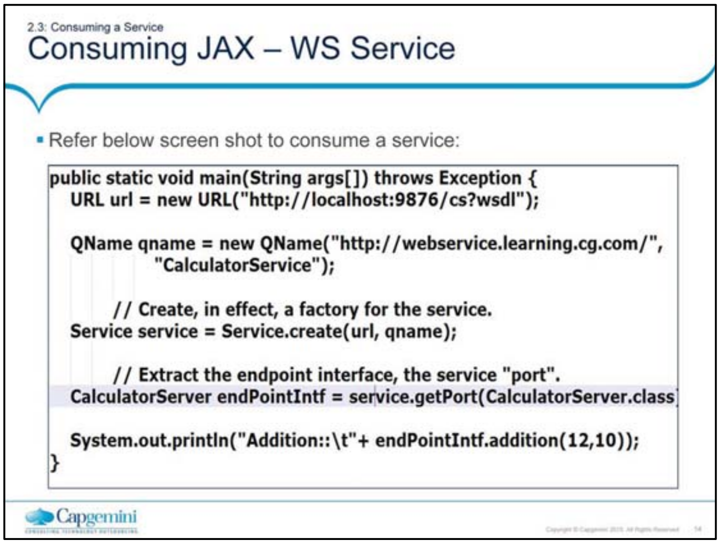
The application runs indefinitely, awaiting service requests.

It needs to be terminated at the command prompt with control-C or the equivalent.

You can check the following URL in browser:

http://127.0.0.1:9876/cs?wsdl

To view the service contract, the WSDL document.



2.3: Consuming a Service

Consuming JAX – WS Service

- Refer below screen shot to consume a service:

```
public static void main(String args[]) throws Exception {
    URL url = new URL("http://localhost:9876/cs?wsdl");

    QName qname = new QName("http://webservice.learning.cg.com/",
                "CalculatorService");

        // Create, in effect, a factory for the service.
    Service service = Service.create(url, qname);

        // Extract the endpoint interface, the service "port".
    CalculatorServer endPointIntf = service.getPort(CalculatorServer.class

    System.out.println("Addition::\t"+ endPointIntf.addition(12,10));
}
```

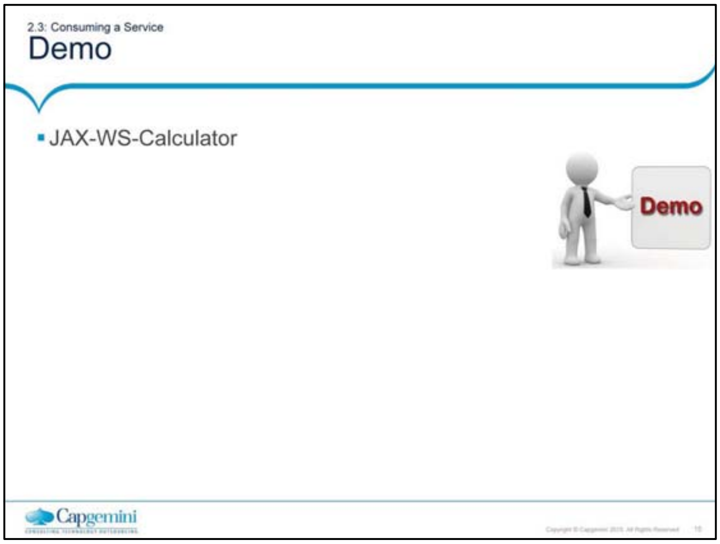
Here basic XML API is being used to consume a service.

The URL defines the location of wsdl document.

The Qname (qualified name) prints the service URI (as to where the service will be available). “CalculatorService” is the service name to be exposed in wsdl document.

Next we create a factory for the service with the URL and Qname at a port number where SEI (Service Endpoint Interface) is available.

Lastly by using endPoint interface we can consume the service.

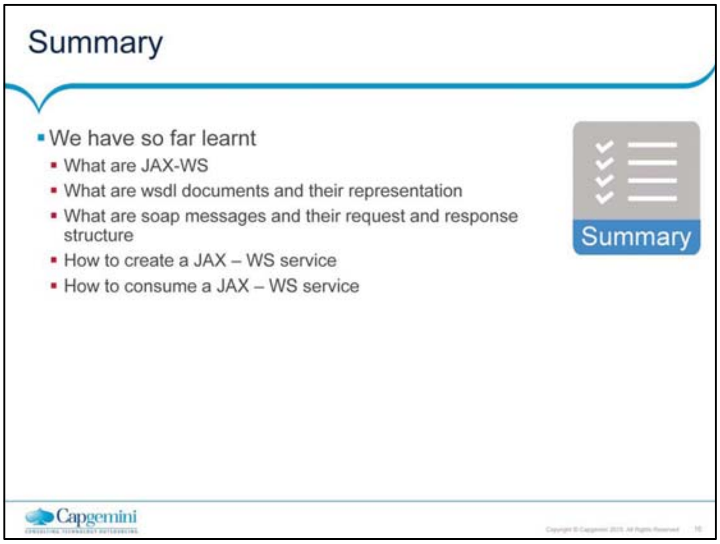


2.3: Consuming a Service

Demo

- JAX-WS-Calculator

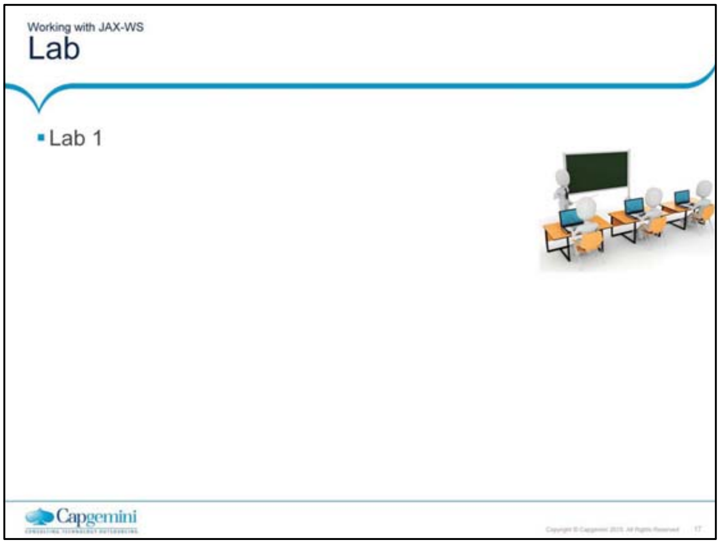
Add the notes here.



Summary

- We have so far learnt
  - What are JAX-WS
  - What are wsdl documents and their representation
  - What are soap messages and their request and response structure
  - How to create a JAX – WS service
  - How to consume a JAX – WS service

Add the notes here.

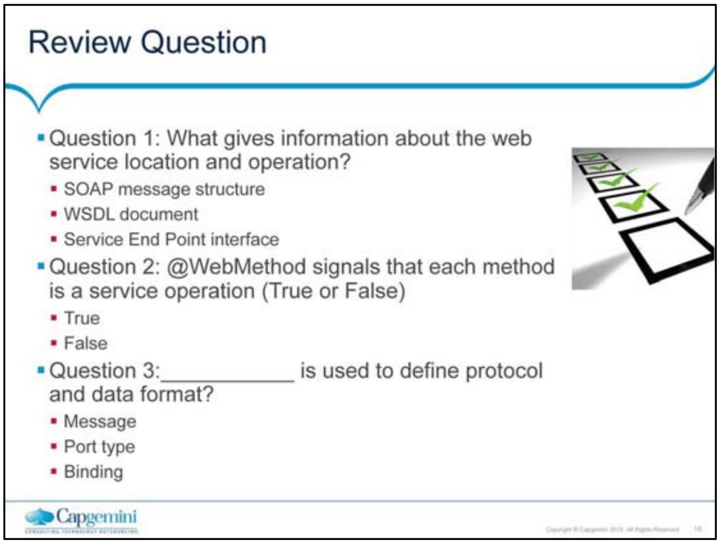


Working with JAX-WS

Lab

- Lab 1

Add the notes here.

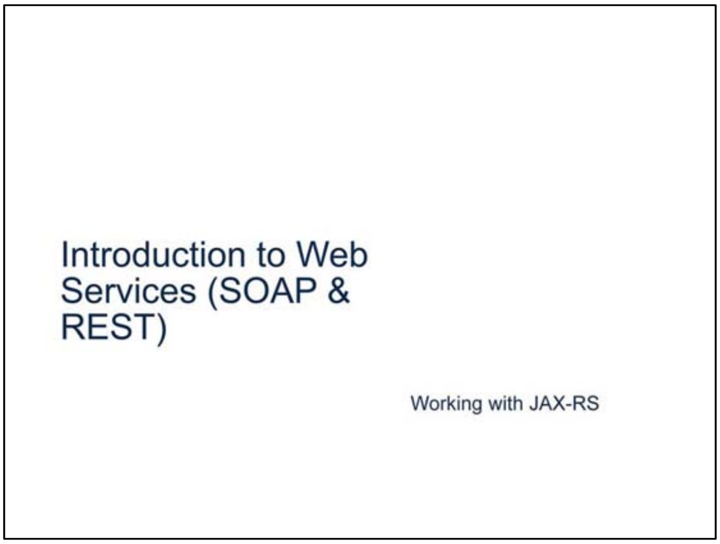


Review Question

- Question 1: What gives information about the web service location and operation?
  - SOAP message structure
  - WSDL document
  - Service End Point interface
- Question 2: @WebMethod signals that each method is a service operation (True or False)
  - True
  - False
- Question 3:___________ is used to define protocol and data format?
  - Message
  - Port type
  - Binding

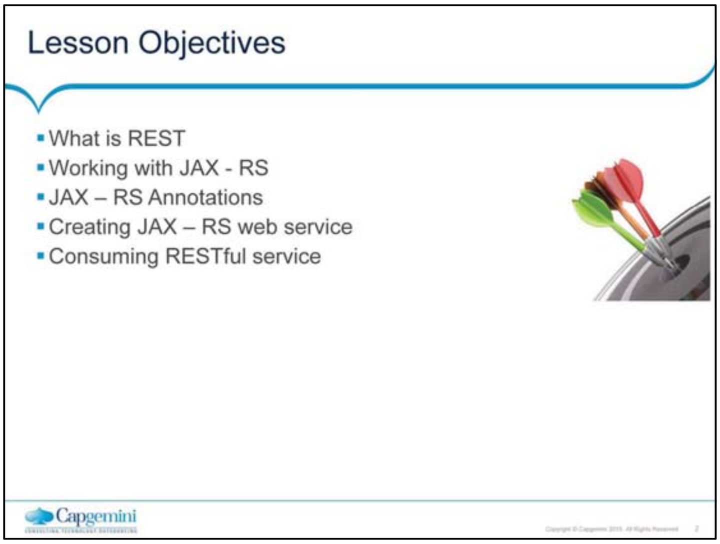
Capgemini

Add the notes here.



Introduction to Web Services (SOAP & REST)

Working with JAX-RS

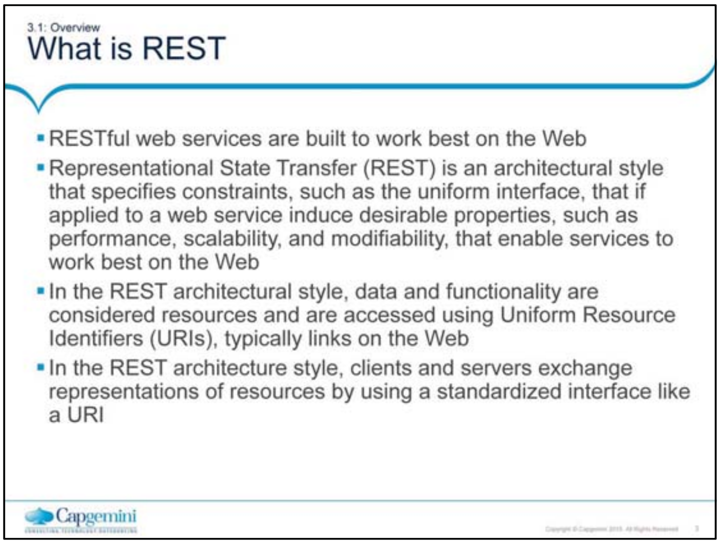


Lesson Objectives

- What is REST
- Working with JAX - RS
- JAX – RS Annotations
- Creating JAX – RS web service
- Consuming RESTful service

Following contents would be covered:

3.1: What is REST
    3.1.1 : SOAP and REST
3.2: Working with JAX - RS
3.3: JAX-RS annotations
3.4: Creating JAX – RS web service
3.5: Consuming RESTful service



3.1: Overview

What is REST

- RESTful web services are built to work best on the Web
- Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web
- In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web
- In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface like a URI

REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP)

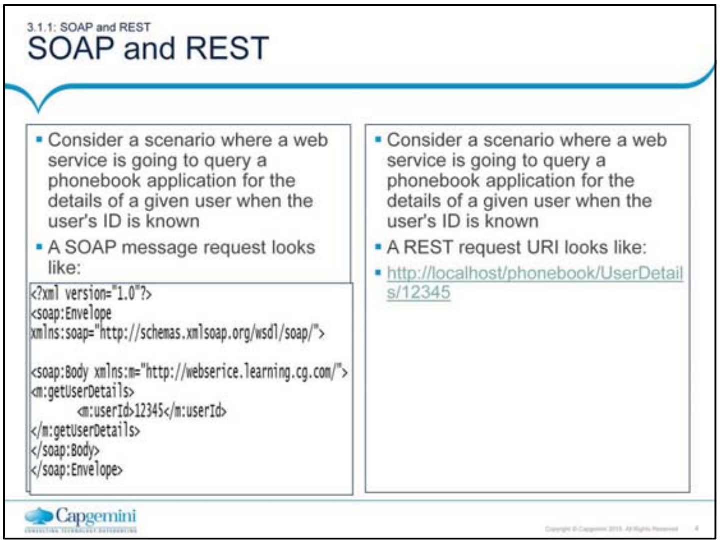
The following principles encourage RESTful applications to be simple, lightweight, and fast:

Resource identification through URI: A RESTful web service exposes a set of resources.
Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT updates an existing resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.

Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control.

Stateful interactions through hyperlinks: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction



3.1.1: SOAP and REST

SOAP and REST

- Consider a scenario where a web service is going to query a phonebook application for the details of a given user when the user's ID is known
- A SOAP message request looks like:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

<soap:Body xmlns:m="http://webserice.learning.cg.com/">
<m:getUserDetails>
        <m:userId>12345</m:userId>
</m:getUserDetails>
</soap:Body>
</soap:Envelope>
```

- Consider a scenario where a web service is going to query a phonebook application for the details of a given user when the user's ID is known
- A REST request URI looks like:
- http://localhost/phonebook/UserDetails/12345

Capgemini

As seen in above slide same scenario is represented in both forms SOAP and REST
REST turns out to be an easier implementation.

REST vs. SOAP is like mailing a letter:
with SOAP, you're using an envelope;
with REST, it's a postcard. Postcards are easier to handle (by the receiver), waste less paper (i.e., consume less bandwidth), and have a short content.

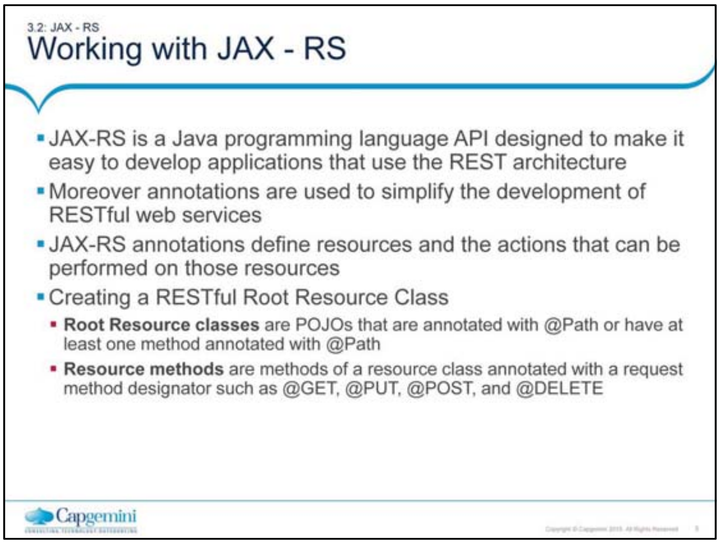
With REST, the semantics are specified entirely by the URI
With REST, the URI is the end
REST: oriented around nouns – Resources

Following examples can be considered:
http://example.com/customer/123
http://example.com/order/555/customer
{POST, GET, DELETE}
REST services are built around Resources

REST services are Stateless and use a Uniform Interface. Resources are manipulated through Representations. Messages are Self-Describing

Differences between SOAP and REST

SOAP	REST
1. Is Simple Object Access Protocol	1. REST is Representational State Transfer
2. SOAP requires more bandwidth	2. REST requires less bandwidth
3. SOAP permits only XML	3. REST permits different data formats like text, XML, JSON (Java script Object Notation) data formats
4. SOAP is Transport layer independent (HTTP). It can be used over SMTP as well.	4. REST requires use of HTTP
5. SOAP is a standard to interact with Web service	5. REST has small learning curve and uses URI to Interact with Web services



3.2: JAX - RS

Working with JAX - RS

- JAX-RS is a Java programming language API designed to make it easy to develop applications that use the REST architecture
- Moreover annotations are used to simplify the development of RESTful web services
- JAX-RS annotations define resources and the actions that can be performed on those resources
- Creating a RESTful Root Resource Class
  - **Root Resource classes** are POJOs that are annotated with @Path or have at least one method annotated with @Path
  - **Resource methods** are methods of a resource class annotated with a request method designator such as @GET, @PUT, @POST, and @DELETE

There are many implementations of JAX – RS

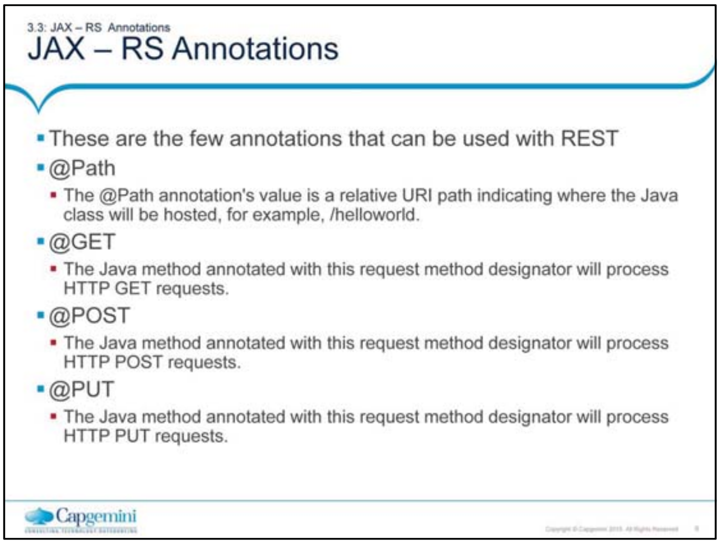
Apache CFX, an Open source Web service framework

Jersey , the reference implementation from Oracle. We would be using Jersey implementation for our REST web services

Jersey is used because it makes development of RESTful web services easier.

RESTful web services created under Jersey can be deployed to any application server.

Moreover JAX – RS is a part of Java EE and can be used with other Java EE technologies.



3.3: JAX – RS Annotations

JAX – RS Annotations

- These are the few annotations that can be used with REST
- @Path
  - The @Path annotation's value is a relative URI path indicating where the Java class will be hosted, for example, /helloworld.
- @GET
  - The Java method annotated with this request method designator will process HTTP GET requests.
- @POST
  - The Java method annotated with this request method designator will process HTTP POST requests.
- @PUT
  - The Java method annotated with this request method designator will process HTTP PUT requests.

The @Path annotation's value is a relative URI path indicating where the Java class will be hosted: for example, /helloworld. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user and pass it to the application as a variable in the URI: /helloworld/{username}.

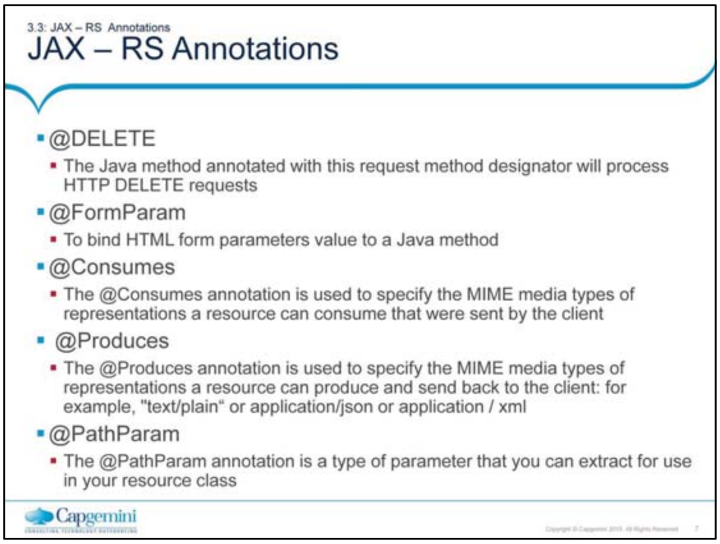
The @GET annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.

URI for GET request in JAX-RS-CRUD Demo shared (http://localhost:9090/JAX-RS-CRUD/rest/countries)

The @POST annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.

URI for POST request in JAX-RS-CRUD Demo shared (http://localhost:9090/JAX-RS-CRUD/post.jsp) (Here the user enters the details to create a country. The values of which are passed to the controller via @FormParam-discussed in next slide)

The @PUT annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.



3.3: JAX – RS Annotations

JAX – RS Annotations

- @DELETE
  - The Java method annotated with this request method designator will process HTTP DELETE requests
- @FormParam
  - To bind HTML form parameters value to a Java method
- @Consumes
  - The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client
- @Produces
  - The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client: for example, "text/plain" or application/json or application / xml
- @PathParam
  - The @PathParam annotation is a type of parameter that you can extract for use in your resource class

The @DELETE annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding

URI for DELETE request in JAX-RS-CRUD Demo shared (http://localhost:9090/JAX-RS-CRUD/rest/countries/delete)

@FormParam: Used to bind HTML form parameters value to a Java method

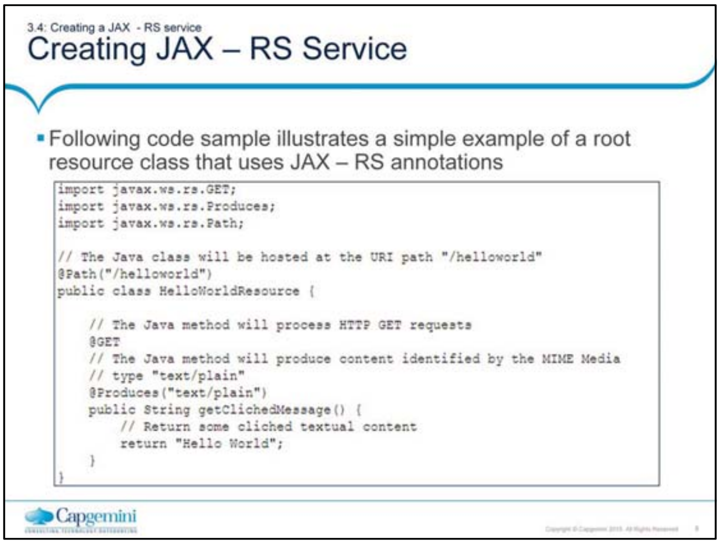
With respect to the JAX-RS-CRUD demo shared - the values entered in post.jsp mentioned in previous slide are passed to controller method. Since we are using HTTP POST method the URI is not appended with parameters.
Following is the URI (http://localhost:9090/JAX-RS-CRUD/rest/countries) to demonstrate that new country is added.

The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client.
@Consumes(MediaType.APPLICATION_JSON) specifies that the controller method will need an input parameter of type object of type JSON

The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client: for example, "text/plain"..
In JAX-RS-CRUD demo shared - @Produces(MediaType.APPLICATION_JSON) specifies that the controller method will return an object of type JSON

The @PathParam annotation is a type of parameter that you can extract for use in your resource class. URI path parameters are extracted from the request URI, and the parameter names correspond to the URI path template variable names specified in the @Path class-level annotation

URI for GET request with Path parameter in JAX-RS-CRUD Demo shared (http://localhost:9090/JAX-RS-CRUD/rest/countries/3) (3 is the path parameter appended to the URI). Thus details with respect to country Id = 3 is fetched and displayed in browser.



3.4: Creating a JAX - RS service

Creating JAX – RS Service

- Following code sample illustrates a simple example of a root resource class that uses JAX – RS annotations

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

The @Path annotation's value is a relative URI path. In the preceding example, the Java class will be hosted at the URI path /helloworld. This is an extremely simple use of the @Path annotation, with a static URI path.

Variables can be embedded in the URIs. URI path templates are URIs with variables embedded within the URI syntax.

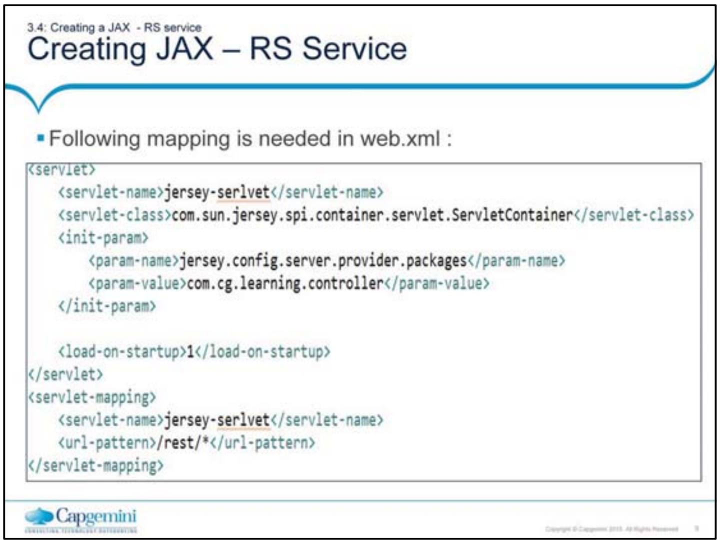
The @GET annotation is a request method designator, along with @POST, @PUT, @DELETE, and @HEAD, defined by JAX-RS and corresponding to the similarly named HTTP methods. In the example, the annotated Java method will process HTTP GET requests.
The behavior of a resource is determined by the HTTP method to which the resource is responding.

The @Produces annotation is used to specify the MIME media types a resource can produce and send back to the client. In this example, the Java method will produce representations identified by the MIME media type "text/plain".

The @Consumes annotation is used to specify the MIME media types a resource can consume that were sent by the client.

The example could be modified to set the message returned by the getClichedMessage method, as shown in this code example:

```
@POST
@Consumes("text/plain")
public void postClichedMessage(String message)
{
                    // Store the message
}
```


3.4: Creating a JAX - RS service

Creating JAX – RS Service

- Following mapping is needed in web.xml :

```
<servlet>
    <servlet-name>jersey-serlvet</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>com.cg.learning.controller</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>jersey-serlvet</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

From the above mapping all URL patterns will map through URLs with rest being a part of the URI.

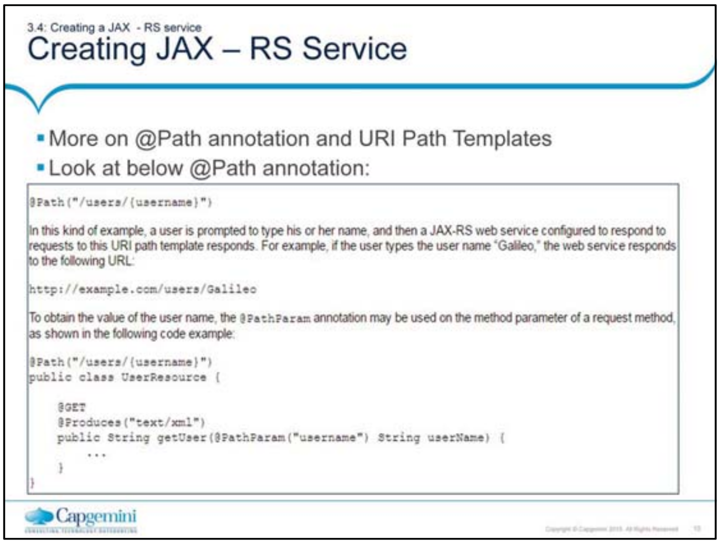
An inbuilt servlet -> com.sun.jersey.spi.container.servlet.ServletContainer is a servlet (Filter) for deploying root resource classes.

Provides support for servlet-based and filter-based Web applications.

The above mapping will deploy Jersey and automatically register any root resource or provider classes present in the directory "/WEB-INF/classes" or jar files present in the directory "/WEB-INF/lib“

The init-param -> jersey.config.server.provider.packages defines one or more packages that contain application-specific resources and providers.

If the property is set, the specified packages will be scanned for JAX-RS root resources and providers.



3.4: Creating a JAX - RS service

Creating JAX – RS Service

- More on @Path annotation and URI Path Templates
- Look at below @Path annotation:

```
@Path("/users/{username}")
```

In this kind of example, a user is prompted to type his or her name, and then a JAX-RS web service configured to respond to requests to this URI path template responds. For example, if the user types the user name "Galileo," the web service responds to the following URL:

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

To obtain the value of the user name, the `@PathParam` annotation may be used on the method parameter of a request method, as shown in the following code example:

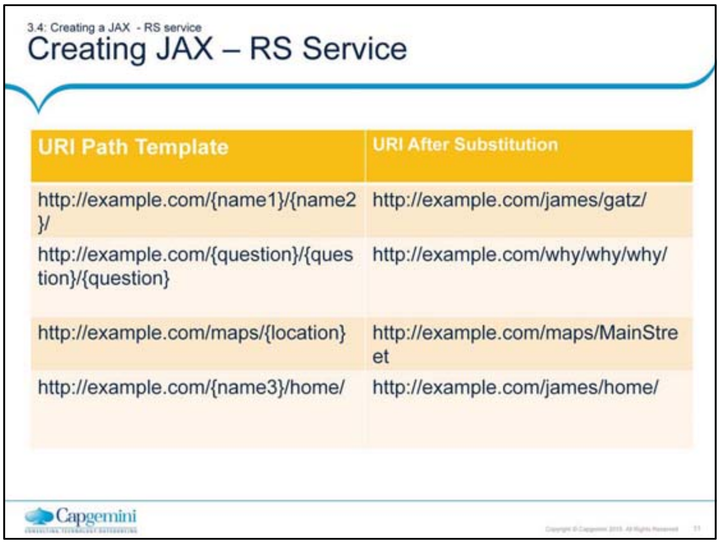
```
@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String userName) {
        ...
    }
}
```

Capgemini

The @Path annotation identifies the URI path template to which the resource responds and is specified at the class or method level of a resource.

The @Path annotation's value is a partial URI path template relative to the base URI of the server on which the resource is deployed, the context root of the application, and the URL pattern to which the JAX-RS runtime responds.

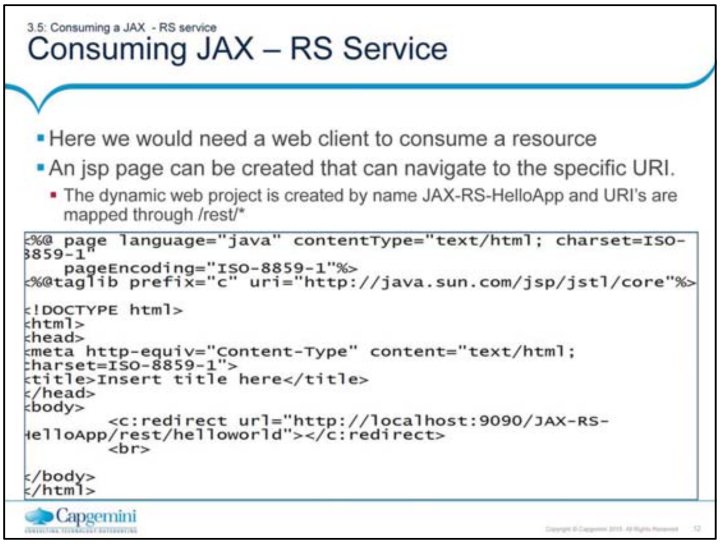
URI path templates are URIs with variables embedded within the URI syntax. These variables are substituted at runtime in order for a resource to respond to a request based on the substituted URI. Variables are denoted by braces ({ and }). For example, look at the following @Path annotation:
@Path("/users/{username}")



3.4: Creating a JAX - RS service

Creating JAX – RS Service

URI Path Template	URI After Substitution
http://example.com/{name1}/{name2}/	http://example.com/james/gatz/
http://example.com/{question}/{question}/{question}	http://example.com/why/why/why/
http://example.com/maps/{location}	http://example.com/maps/MainStreet
http://example.com/{name3}/home/	http://example.com/james/home/



3.5: Consuming a JAX - RS service

Consuming JAX – RS Service

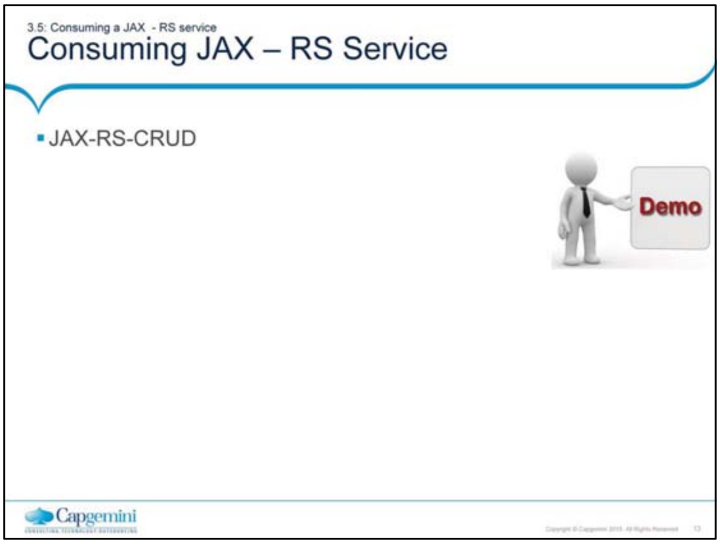
- Here we would need a web client to consume a resource
- An jsp page can be created that can navigate to the specific URI.
  - The dynamic web project is created by name JAX-RS-HelloApp and URI's are mapped through /rest/*

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
        <c:redirect url="http://localhost:9090/JAX-RS-HelloApp/rest/helloworld"></c:redirect>
        <br>

</body>
</html>
```

Capgemini

When above jsp is executed, REST Service will look for Root resource class with @Path mapping as "/helloworld" and on finding the mapping will execute that corresponding method and return result in MIME type specified.

(Refer slide number 8)

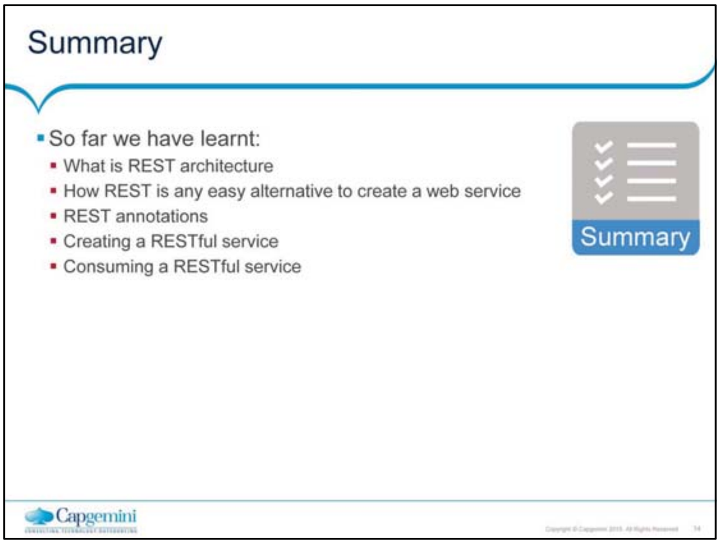


3.5: Consuming a JAX - RS service

Consuming JAX – RS Service

- JAX-RS-CRUD

Add the notes here.



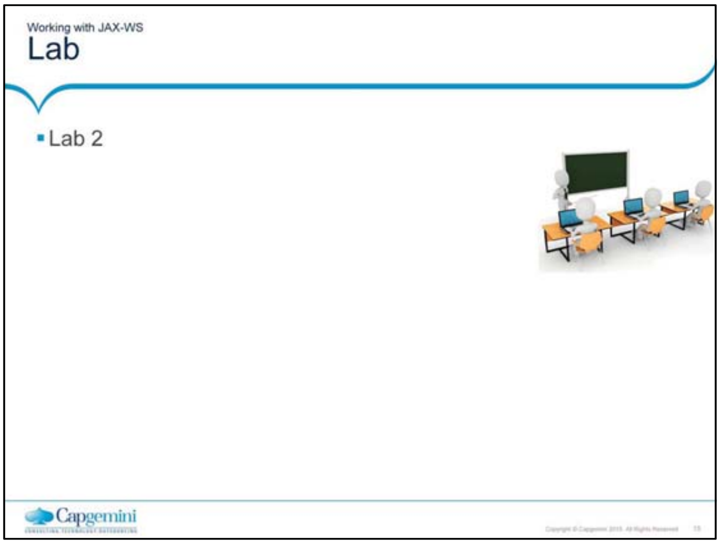
Summary

- So far we have learnt:
  - What is REST architecture
  - How REST is any easy alternative to create a web service
  - REST annotations
  - Creating a RESTful service
  - Consuming a RESTful service

Summary

Capgemini

Add the notes here.

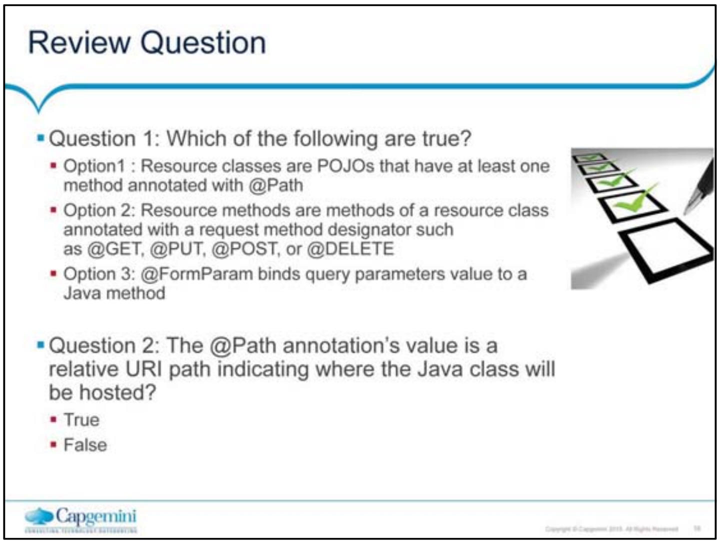


Working with JAX-WS

Lab

- Lab 2

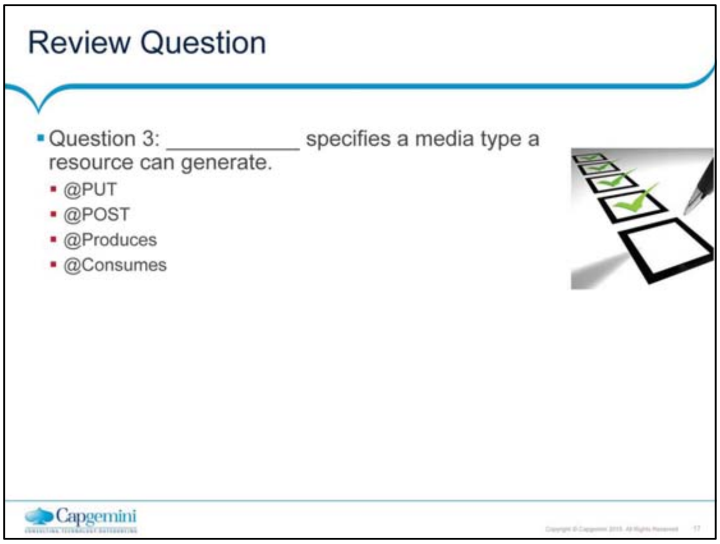
Add the notes here.



Review Question

- Question 1: Which of the following are true?
  - Option1 : Resource classes are POJOs that have at least one method annotated with @Path
  - Option 2: Resource methods are methods of a resource class annotated with a request method designator such as @GET, @PUT, @POST, or @DELETE
  - Option 3: @FormParam binds query parameters value to a Java method
- Question 2: The @Path annotation's value is a relative URI path indicating where the Java class will be hosted?
  - True
  - False

Add the notes here.



Review Question

- Question 3: ___________ specifies a media type a resource can generate.
  - @PUT
  - @POST
  - @Produces
  - @Consumes

Add the notes here.

Introduction to Web Services (SOAP and REST)– 2.0 Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
May 2016	1.0	Yukti A Valecha	Created new lab book as per revised course contents

Table of Contents

Getting Started

Overview

This lab book is a guided tour for learning Introduction to Web Services(SOAP and REST) version 2.0. It comprises 'To Do' assignments.

Setup Checklist for Web Services 2.0

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)
- Internet Explorer 6.0 or higher
- Wildfly Server 8.0.

Please ensure that the following is done:

- A text editor like Notepad or Eclipse is installed.
- JDK 1.8 is installed. (This path is henceforth referred as <java_install_dir>)
- Wildfly is installed but not started.

Instructions

- For all naming conventions refer Appendix A.All lab assignments should refer coding standards.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory webservices_assgn. For each lab exercise create a directory as lab <lab number>.
- You may also look up the on-line help if necessary.

Lab 1. Web Services (JAX-WS)

Goals	Understand the process of creating and Consuming a Java Web Service
Time	50 minutes

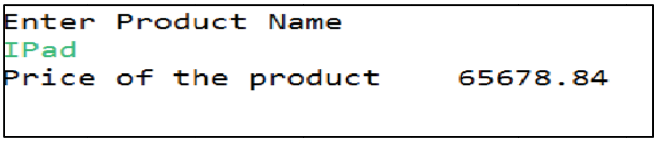
1. Refer below Java files:
    - Product.java
    - ProductDB.java

Create a Product Web service that will accept the product name and return the price of the product to the web service consumer.

If the product is not available then the web service should return a message to the consumer specifying that the product is not available.

Note: Make use of the static DB given in ProductDB.java

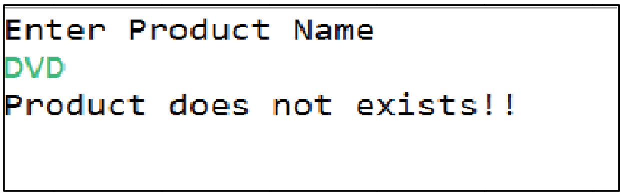
Refer below screen shots:



```
Enter Product Name
IPad
Price of the product     65678.84
```

Figure 1 : Screenshot

If the user enters a product that does not exists then, refer below screen shot



```
Enter Product Name
DVD
Product does not exists!!
```

Figure 2 : Screenshot

Lab 2. Web Services (JAX-RS)

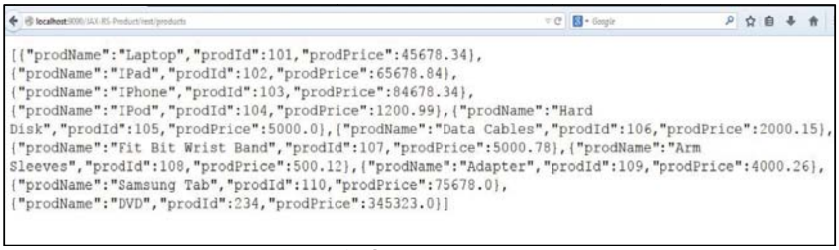
Goals	Understand the process of creating and Consuming a RESTful Java Web Service
Time	90 minutes

1. Refer below Java files:
   Product.java
   ProductDB.java

   Create a Product RESTful web service that will display all products to the Web service consumer.

   Note: Make use of the static DB given in ProductDB.java

   Refer below screen shot:



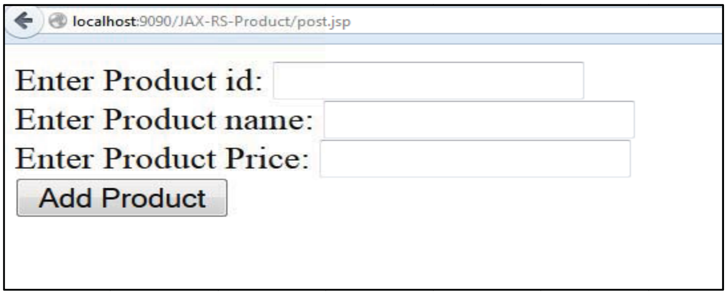
localhost:9090/JAX-RS-Product/rest/products

```
[{"prodName":"Laptop","prodId":101,"prodPrice":45678.34},
{"prodName":"IPad","prodId":102,"prodPrice":65678.84},
{"prodName":"IPhone","prodId":103,"prodPrice":84678.34},
{"prodName":"IPod","prodId":104,"prodPrice":1200.99},{"prodName":"Hard
Disk","prodId":105,"prodPrice":5000.0},{"prodName":"Data Cables","prodId":106,"prodPrice":2000.15},
{"prodName":"Fit Bit Wrist Band","prodId":107,"prodPrice":5000.78},{"prodName":"Arm
Sleeves","prodId":108,"prodPrice":500.12},{"prodName":"Adapter","prodId":109,"prodPrice":4000.26},
{"prodName":"Samsung Tab","prodId":110,"prodPrice":75678.0},
{"prodName":"DVD","prodId":234,"prodPrice":345323.0}]
```

Figure 3 : Screenshot

The consumer should also be able to add a product to the existing list of products.

Refer below screen shots:

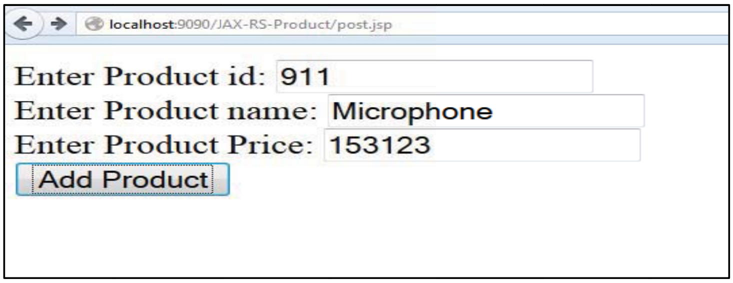


localhost:9090/JAX-RS-Product/post.jsp

Enter Product id:
Enter Product name:
Enter Product Price:

Figure 4 : Input Screen

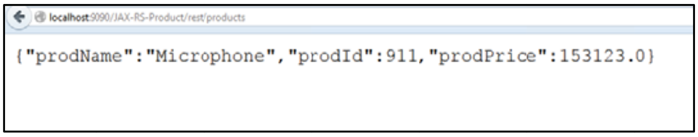
After entering product details, refer below screen shot



Enter Product id: 911
Enter Product name: Microphone
Enter Product Price: 153123

Figure 5 : Input Screen with data

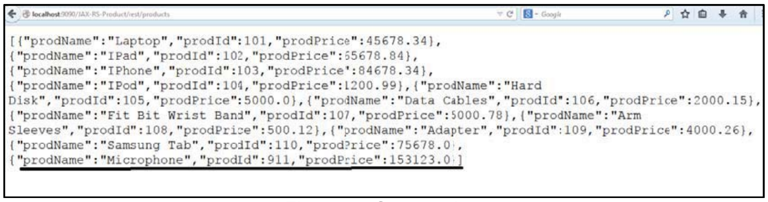
Product is added to existing list of products



```
{"prodName":"Microphone","prodId":911,"prodPrice":153123.0}
```

Figure 6 : Screen shot

Now, new product is added to existing list of products



```
[{"prodName":"Laptop","prodId":101,"prodPrice":45678.34},
{"prodName":"IPad","prodId":102,"prodPrice":65678.84},
{"prodName":"IPhone","prodId":103,"prodPrice":84678.34},
{"prodName":"IPod","prodId":104,"prodPrice":1200.99},{"prodName":"Hard
Disk","prodId":105,"prodPrice":5000.0},{"prodName":"Data Cables","prodId":106,"prodPrice":2000.15},
{"prodName":"Fit Bit Wrist Band","prodId":107,"prodPrice":5000.78},{"prodName":"Arm
Sleeves","prodId":108,"prodPrice":500.12},{"prodName":"Adapter","prodId":109,"prodPrice":4000.26},
{"prodName":"Samsung Tab","prodId":110,"prodPrice":75678.0},
{"prodName":"Microphone","prodId":911,"prodPrice":153123.0}]
```

Figure 7 : Screen shot

Appendices

Appendix A: Naming Conventions

Package names are written in all lower case to avoid conflict with the names of classes or interfaces.Companies use their reversed Internet domain name to begin their package names—for example, com.cg.learning.mypackage for a package named learning.mypackage created by a programmer at cg.com.
Packages in the Java language itself begin with **java. Orjavax**.

Classes and interfaces the first letter should be capitalized, and if several words are linked together to form the name, the first letter of the inner words should be uppercase (a format that's sometimes called "camelCase").
For classes, the names should typically be nouns. For example:
Dog
Account
PrintWriter
For interfaces, the names should typically be adjectives like
Runnable
Serializable
Methods The first letter should be lowercase, and then normal camelCaserules should be used. In addition, the names should typically be verb-noun pairs. For example:
getBalance
doCalculation
setCustomerName
Variables Like methods, the camelCase format should be used, starting with a lowercase letter. Sun recommends short, meaningful names, which sounds good to us. Some examples:
buttonWidth
accountBalance
myString
Constants Java constants are created by marking variables static and final. They should be named using uppercase letters with underscore characters as separators:
MIN_HEIGHT

Appendix B: Table of Figures