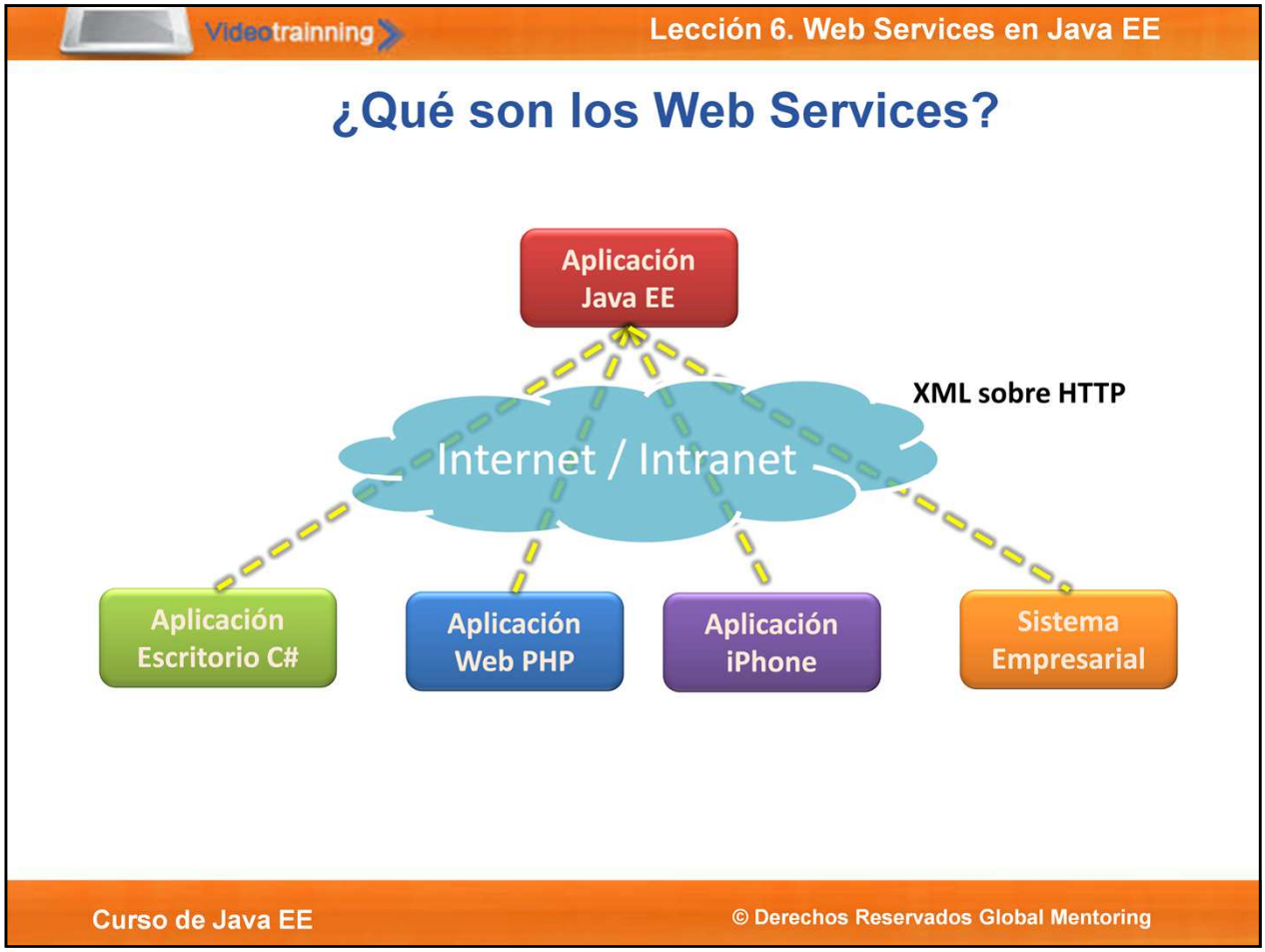


El tema de Web Services ha sido utilizado ya por varios años, a lo largo de estos años se han aprendido técnicas y nuevas formas de establecer una comunicación más eficiente entre sistemas creados en distintas plataformas o tecnologías.

Los Web Services son una tecnología orientada a la intercomunicación de sistemas. Algunas de sus características son:

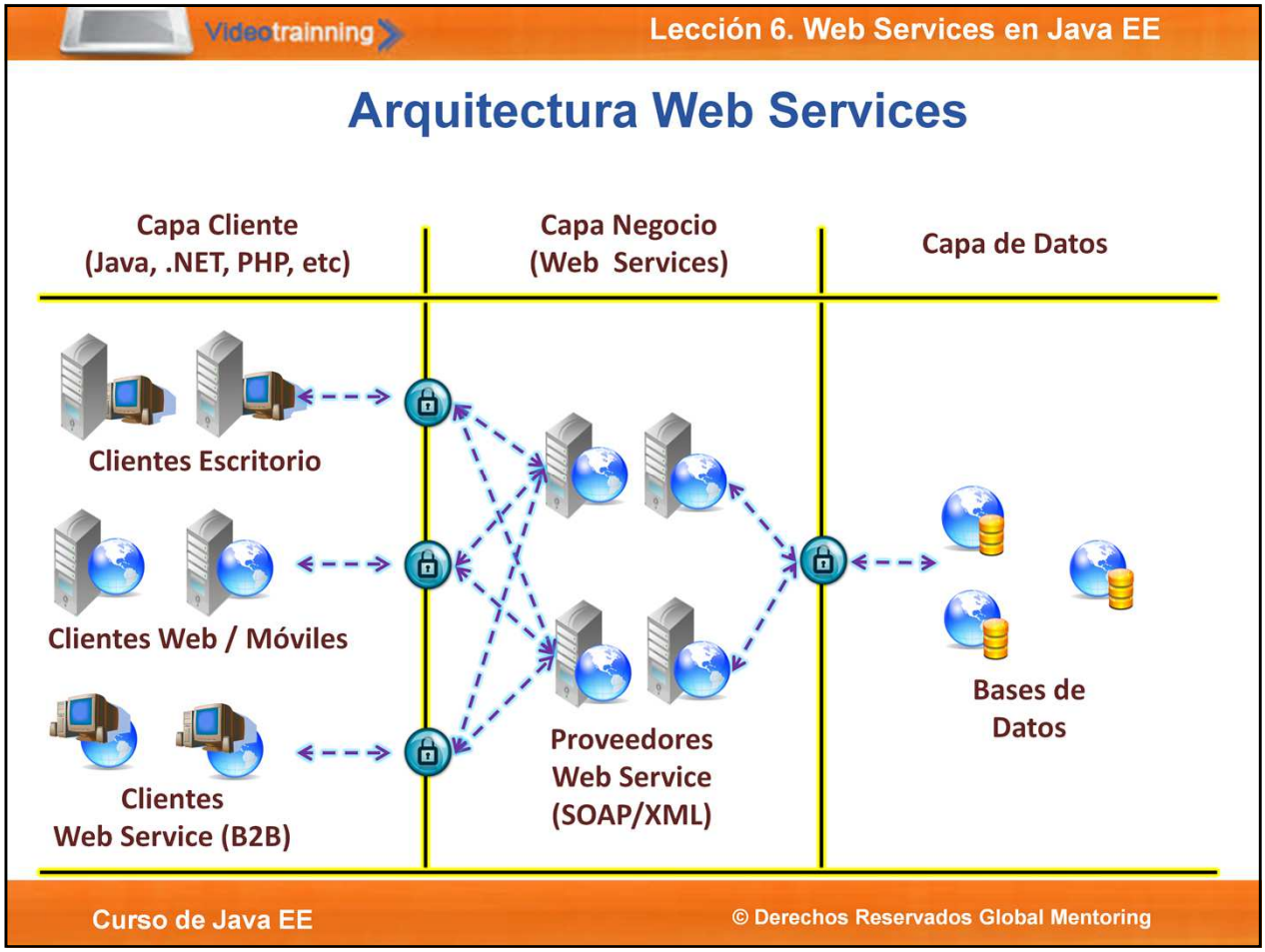
- **Interoperabilidad y Portabilidad**: Los sistemas que utilizan Servicios Web permiten intercambiar información entre distintas plataformas y lenguajes de programación utilizando la Web (Intranet/Internet) como base para la comunicación. Una de las formas es utilizar el protocolo en SOAP permitiendo la comunicación vía XML, logrando la independencia de plataforma. Todo esto apoyándonos del protocolo HTTP.

- **Reusabilidad**: Permite reutilizar mucha de la lógica de negocio proveniente de sistemas legados o de nuestros sistemas empresariales actuales.

- **Disponibilidad**: El objetivo de los servicios web es que se encuentran disponibles en cualquier momento y en cualquier lugar, para cualquier sistema y/o persona que necesite utilizarlos. Además, no requieren de intervención humana incluso en transacciones muy complejas.

Como podemos observar en la figura, en muchas ocasiones necesitamos intercomunicar sistemas de información, los cuales pueden haber sido desarrollados en la misma tecnología o no. La versión JEE 6 cuenta con dos APIs principales para el desarrollo de Web Services: Java API for XML Web Services (JAX-WS) y Java API for RESTful Web Services (JAX-RS), las cuales estudiaremos en esta lección.

Para más información de las tecnologías de Web Services en JEE 6:
http://docs.oracle.com/javaee/6/tutorial/doc/bnayk.html



En una aplicación Java Empresarial, los Web Services se pueden ver como una alternativa para exponer los EJBs sin necesidad de utilizar RMI o el uso de Java.

Los EJBs se pueden exponer como Servicios Web, con ello reutilizamos su lógica de negocio. Lo anterior permite a ***El Cliente*** ser escrito en cualquier lenguaje como Objective-C, .Net, PHP, etc, sin ninguna dependencia con el lenguaje Java.

Si eres nuevo en el tema de Web Services, los RESTful Web Services son más sencillos de aprender. Este tipo de Servicios Web los estudiaremos en la segunda parte de esta lección.

En esta primera parte estudiaremos los SOAP Web Services, los cuales requieren de un conocimiento básico de temas tales como XML, XSD (esquemas XML) y JAXB. Si no se tienes este conocimiento, mostraremos brevemente en qué consisten estas tecnologías, cómo las aplicaremos, así como una referencia para su estudio más profundo.

Cabe aclarar que debido a que la versión Java EE 6 ha simplificado enormemente el proceso de creación de Web Services, muchas de estas tecnologías estarán únicamente tras bambalinas, y cada que trabajemos en un nuevo Servicio Web se irá adquiriendo poco a poco más de experiencia en cada uno de estos temas y/o tecnologías.

Los Web Services son una parte primordial para el desarrollo de arquitecturas SOA (Service-Oriented Architecture) con el objetivo de integrar aplicaciones creadas en la misma o distintas plataformas.

En los siguientes ejercicios veremos cómo exponer EJB's de tipo Stateless SessionBeans como Servicios Web.

En Java EE 6, JAX-WS y JAX-RS son los dos estándares para crear Web Services:

- **JAX-WS: Java API for XML Web Services** permite definir Web Services también conocidos como ***SOAP Web Services***. JAX-WS 2.0 reemplaza el uso de JAX-RPC. La creación de Web Services utilizando JAX-RPC (Remote Procedure Call) fue el estándar en la versión 1.4, sin embargo en este curso no estudiaremos este tema. Para más información en la creación de este tipo de Web Services:
  http://docs.oracle.com/javaee/1.4/tutorial/doc/JAXRPC.html

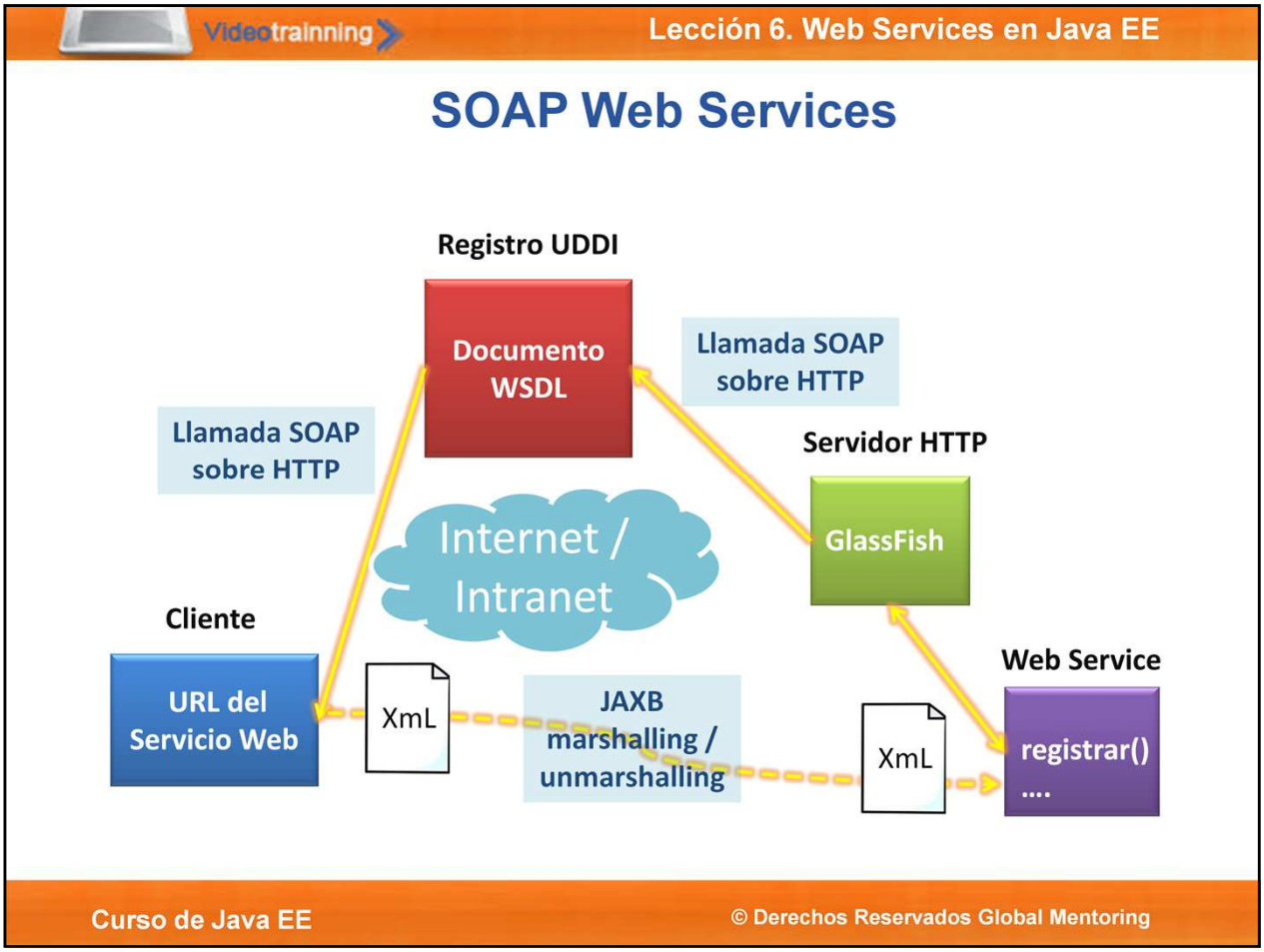
- **JAX-RS: Java API for RESTful Web Services** permite crear Web Services de acuerdo a la terminología Representational State Transfer (REST) bajo el protocolo HTTP. JAX-RS no es el estándar, pero ha ganado gran popularidad en los últimos años.

Cuando utilizamos Java EE 6, debemos decidir si utilizamos el API de JAX-WS o JAX-RS. Para tomar este decisión podemos tomar en cuenta los siguientes puntos:

- **JAX-WS:** Esta API permite abordar requerimientos más complejos y lleva más años en el mercado. Brinda soporte para los protocolos que son estándar en la industria de software al crear Web Services. Estos estándares proveen una manera muy robusta para agregar seguridad, transaccionalidad, interoperabilidad, entre varias características entre el cliente y el servidor al utilizar Web Services.

- **JAX-RS:** Esta API permite crear de manera más simple Web Services, solo que está restringido por el estilo REST, el cual utiliza el protocolo HTTP y sus métodos soportados y códigos de estado para establecer las operaciones básicas de un Servicio Web (GET, POST, PUT, DELETE, etc).

Estos tipos de Web Services son los que estudiaremos en esta lección. Para una revisión más detallada de cuando seleccionar un API u otra, pueden consultar el siguiente artículo (en inglés):
http://www2008.org/papers/pdf/p805-pautassoA.pdf



En términos simples un Web Service es una plataforma estándar que provee interoperabilidad entre distintas aplicaciones. Para la mayoría de los programadores esto significa envío y recepción de mensajes XML los cuales son transmitidos vía HTTP/HTTPS.

Tanto el lenguaje XML como el protocolo HTTP son un estándar y son ampliamente aceptados para el envío y recepción de información, así, esta información puede ser procesada por múltiples clientes con distintas tecnologías.

Como se observa en la figura, un Web Services se publica en un tipo directorio conocido como UDDI, el cual significa Universal Description, Discovery and Integration. UDDI es un estándar y tiene como objetivo publicar y permitir el acceso a los Web Services a través de mensajes SOAP.

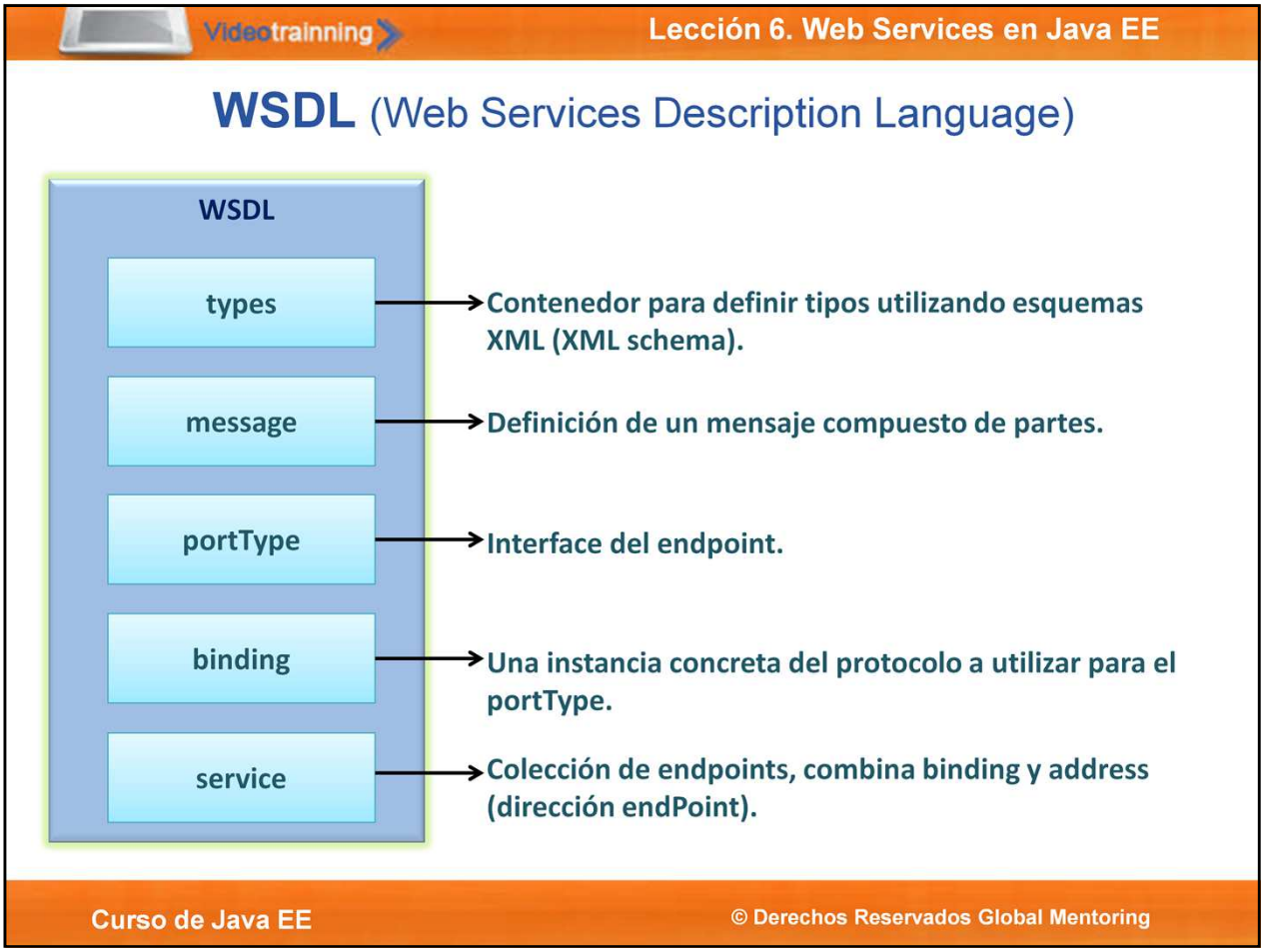
SOAP (Simple Object Access Protocol) es un protocolo estándar el cual define la forma en que se intercambia datos de tipo XML. Este protocolo lo analizaremos en la siguiente lámina.

WSDL (Web Services Description Language) nos permite describir los Web Services. WSDL describe la interface pública de nuestros Web Services y se utiliza XML para su descripción. Este descriptor lo analizaremos posteriormente.

Una vez que ya conocemos la ubicación del Web Services a través del WSDL (URI) del mismo, podemos enviar la petición SOAP y así ejecutar el método expuesto del Web Service. En este procedimiento el XML de petición y respuesta suele validarse utilizando XSD (schema).

Una vez llegado el mensaje XML al servidor Java, este debe convertirse en objetos Java. Para esto es común utilizar la tecnología JAXB, el cual nos permitirá convertir objetos Java en documentos XML (marshaling) y viceversa (unmarshaling). Esta tecnología la comentaremos más adelante.

Como podemos observar son varias las tecnologías relacionadas para entender a detalle la creación de SOAP Web Services, sin embargo la buena notifica es que el estándar JAX-WS nos permitirá ocultar y automatizar muchos de los pasos necesarios para crear un Servicio Web, así como la creación del Cliente del Web Services.



WSDL (Web Services Description Language) es un lenguaje basado en XML, el cual provee la descripción de un servicio Web. De hecho provee una descripción bastante detallada del Servicio Web y con la cual es posible generar tanto el código del Cliente como del Servicio Web asociado. El código generado manejará en automático la conversión de código Java a XML (marshaling) y viceversa (unmarshaling).

En la figura podemos observar la estructura general de un documento WSDL.

- **Types**: En la sección de Types se definen los tipos a utilizar en el mensaje XML. Estos tipos se definen utilizando esquemas XML (xsd).

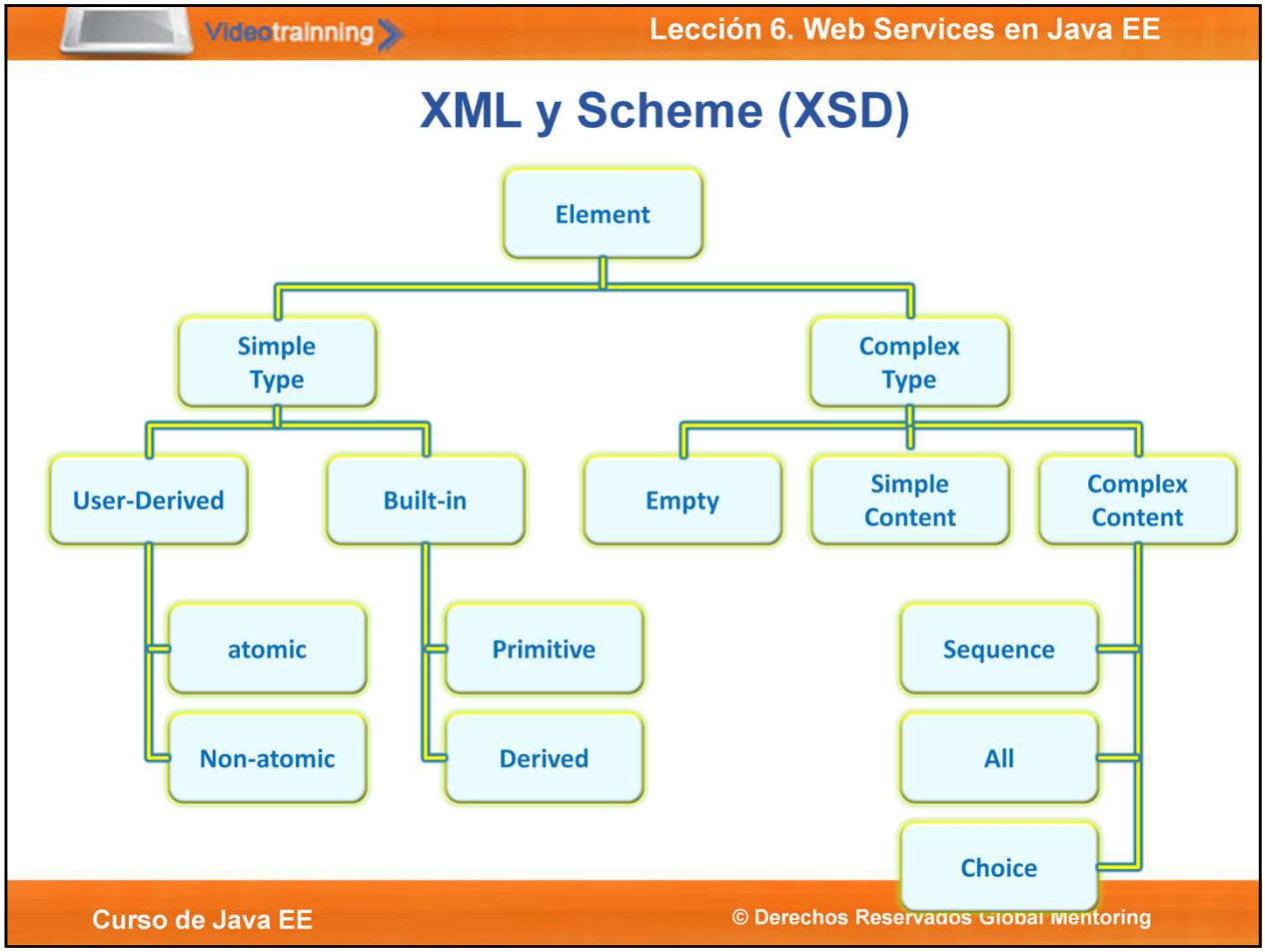
- **Message**: Esta sección define los tipos de mensajes que el Servicio Web soporta. Los mensajes pueden estar compuestos de una o más partes, las cuales pueden ser de entrada (input) o de salida (output).

- **PortType**: Esta sección define un grupo de operaciones (interface del web service). Cada operación debe tener un nombre único y contiene una combinación de elementos de entrada (input) y salida (output), los cuales hacen referencia a los elementos **Message**.

- **Binding**: Esta sección relaciona el portType (interface) con el protocolo a utilizar (ej. SOAP).

- **Service**: Esta sección relaciona la sección **Binding** como servicios y define un endpoint.

Aunque esta información parece en un inicio complicada, conforme se van realizando los primeros Web Services nos iremos familiarizando con estos términos. Sin embargo, si queremos manejar temas más avanzados será necesario estar familiarizados con este terminología y conceptos.



Existen varias formas de validar un documento XML. Esto es básico al momento de transmitir un mensaje a través de un Web Services. La validación nos permitirá automatizar los mensajes entre el Cliente y el Servidor sin necesidad de intervención humana.

Para validar un documento XML es posible realizarlo por medio de 2 técnicas. La primera conocida como DTD (Document Type Definition) y la segunda como XSD (Schema XML). La validación de archivos por DTD es cada vez menos utilizada, debido a que el lenguaje que se utiliza no es XML, y por lo tanto tiene varias limitantes al momento de validar documentos XML con restricciones más avanzadas.

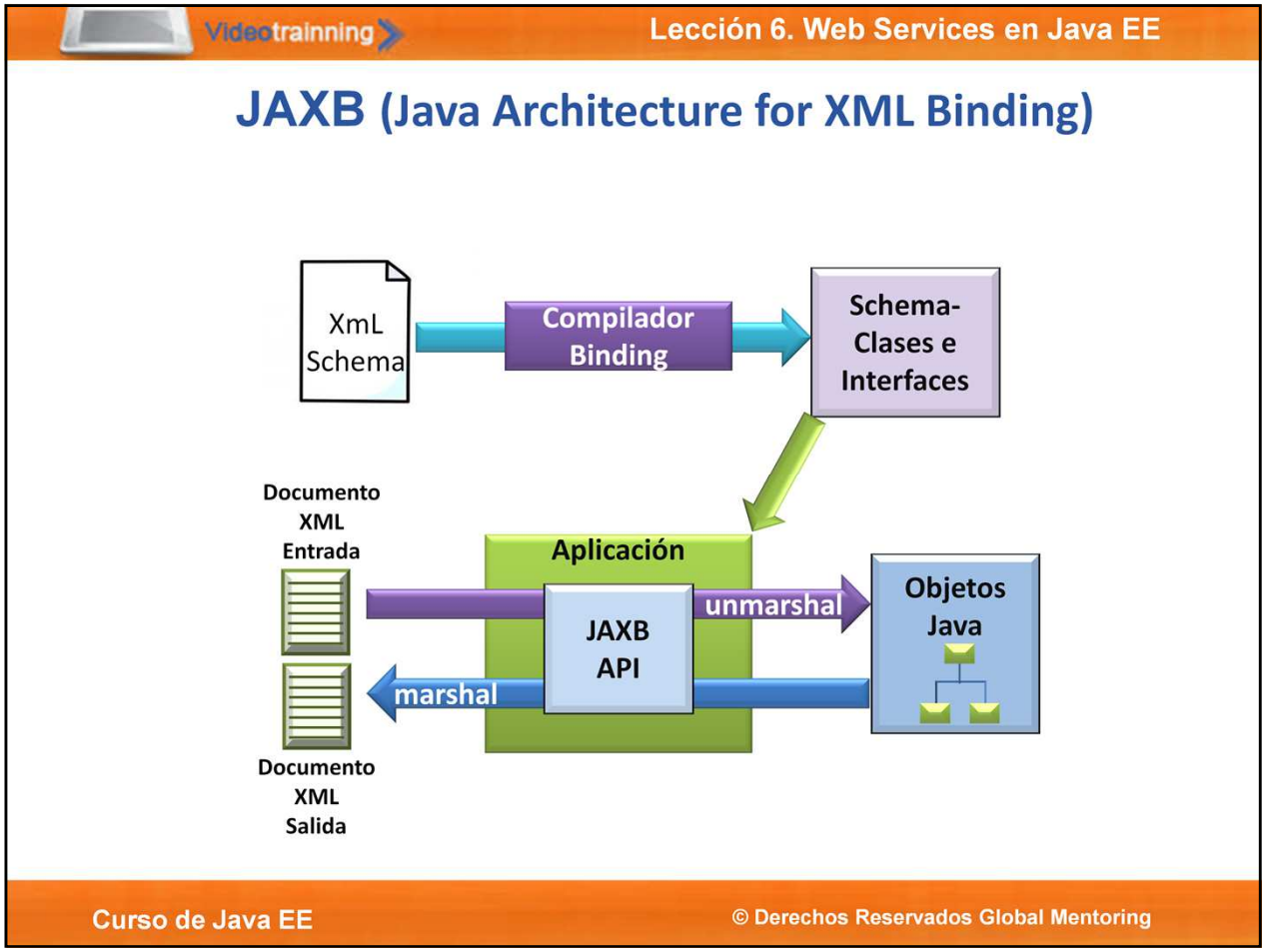
Por otro lado, la validación por XML Schema es cada vez más utilizado para validar la estructura y restricciones de un documento XML. Ya que permite agregar mucho más precisión al momento de validar elementos y atributos complejos de un documento XML, y debido a que está creado con el mismo lenguaje XML, favorece bastante para crear validaciones robustas y flexibles.

En la figura podemos observar las consideraciones a tomar en cuenta para la validación de un documento XML, y se puede observar que son tan detalladas como se necesite, por ejemplo:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="nota">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="para" type="xs:string"/>
            <xs:element name="de" type="xs:string"/>
            <xs:element name="titulo" type="xs:string"/>
            <xs:element name="contenido" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Un ejemplo de un XML validado por el esquema anterior puede ser el siguiente ejemplo:

```
<nota>
   <para>Juan</para>
   <de>Karla</de>
   <titulo>Recordatorio</titulo>
   <contenido>Nos vemos el viernes!</contenido>
</nota>
```

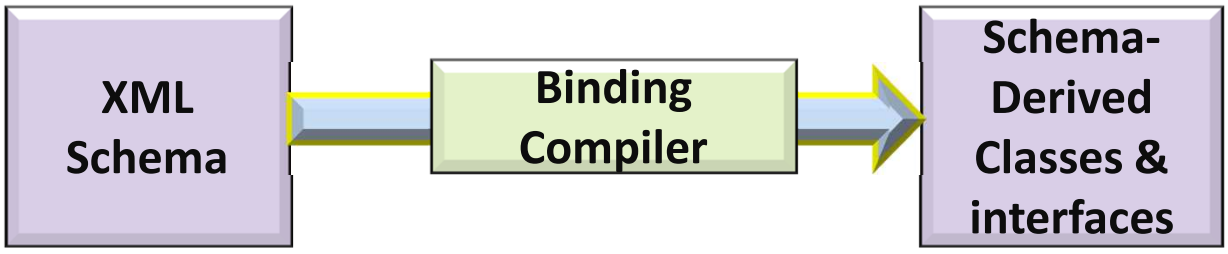


Al trabajar con Web Services, es necesario en algún momento convertir los mensajes XML en objetos Java y viceversa. Existen varias APIs para realizar esta labor, sin embargo el estándar en la versión Java EE es el API de JAXB (Java Architecture for XML Binding).

Como podemos observar en la figura, la tecnología JAXB provee dos principales características. La habilidad de convertir un objeto Java en un documento XML (marshal) y viceversa (unmarshaling).

JAXB permite de manera muy simple acceder y procesar documentos XML sin necesidad de conocer a detalle XML u otras API's de procesamiento.

JAXB asocia el esquema (XSD) del documento XML a procesar (binding), y posteriormente genera las clases Java que representan el documento XML (unmarshalling).



Después de generado el código, se puede acceder y desplegar los datos del documento XML asociado, simplemente utilizando las clases y objetos Java generados. No hay necesidad de utilizar un parser XML o incluso, no hay necesidad de conocer el documento XML original.

También es posible el proceso inverso (marshallling), en el cual a partir de código Java se generan el documento XSD que representa la estructura de los objetos Java y su relación.

Cuando creamos un nuevo Servicio Web, hay dos artefactos que deben generarse: El documento WSDL y las clases Java que implementan el Servicio Web. Además, se debe generar el documento XSD (esquema XML) asociado al mensaje XML a intercambiar por el Servicio Web.

Con el documento WSDL y el esquema XSD, un cliente de un Web Services puede ser autogenerado con ciertas herramientas, por el comando `wsimport` de Java. Este comando permite generar el código Java necesario para invocar un Web Service a partir de la URL del WSDL. Existen herramientas similares en otros lenguajes para crear los clientes de los SOAP Web Services de manera muy similar.

Según podemos observar en la figura y enfocándonos en los dos principales artefactos, WSDL y el código Java, podemos tomar diferentes caminos. La estrategia a seleccionar puede ser por cada Web Service de manera independiente, es decir, no es una decisión única.

Cuando ya tenemos listo el código Java, delegamos la generación del Servicio Web al API de JAX-WS, lo que generará de manera automática tanto el documento WSDL y el archivo XSD asociado al mensaje XML a intercambiar, todo esto basado en el código Java y la relación existente en el método Web Service a exponer.

JAX-WS puede exponer métodos de clases Java POJO's o de EJB's. Seleccionar uno u otro dependerá de si queremos aprovechar los beneficios de tener un EJB, o utilizar un POJO para exponer métodos Java como un Servicio Web.

Para exponer un Servicio Web desde una clase Java solamente debemos anotar la clase con @WebService y agregar la anotación @WebMethod al método a exponer como un Servicio Web. Esto en automático generará el WSDL y el esquema XSD asociado.

Esta es la estrategia que utilizaremos en nuestros ejercicios, ya que expondremos algunos métodos Java de nuestros EJBs como Servicios Web.

Ejercicio 11 y 12

- Abrir el documento PDF de Ejercicios del cursos de Java EE.
- Realizar las siguientes prácticas:

- **Ejercicio 11**: Creación Proyecto HolaMundo con Web Services
- **Ejercicio 12**: Creación Proyecto: SGA con Web Services

Referencias

- La referencia del tema de SOAP Web Services es la siguiente:
- http://docs.oracle.com/javaee/6/tutorial/doc/bnayk.html
- Para un estudio más profundo de las tecnologías utilizadas en los Web Services, se puede consultar la siguiente referencia:
- http://www.oracle.com/technetwork/java/javaee/tech/webservices-139501.html
- Referencia de JAXB:
- http://jaxb.java.net/
- http://docs.oracle.com/javaee/6/tutorial/doc/gkknj.html
- Monitor de conexiones TCP:
- http://code.google.com/p/tcpmon/

Referencias

- Otras referencias y tutoriales de las tecnologías base para el estudio de XML:
- XML: http://www.w3schools.com/xml/default.asp
- XML Schema: http://www.w3schools.com/schema/default.asp
- XML DOM: http://www.w3schools.com/dom/default.asp
- Web Services: http://www.w3schools.com/webservices/default.asp
- WSDL: http://www.w3schools.com/wsdl/default.asp
- SOAP: http://www.w3schools.com/soap/default.asp

Los SOAP Web Services utilizan HTTP como el mecanismo de transporte. Una petición GET o POST es ejecutada y un bloque de código XML es enviado al servidor. La URL que identifica al Servicio Web NO necesariamente indica qué tipo de operación debe realizarse del lado del servidor.

Los RESTful Web Services, por otro lado, se basan completamente en las operaciones soportadas por el protocolo HTTP para ejecutar la funcionalidad del lado del servidor. Cada llamada al Servicio Web, debe utilizar alguno de los siguientes métodos HTTP: GET, POST, PUT, DELETE, URL, HEAD u OPTIONS.

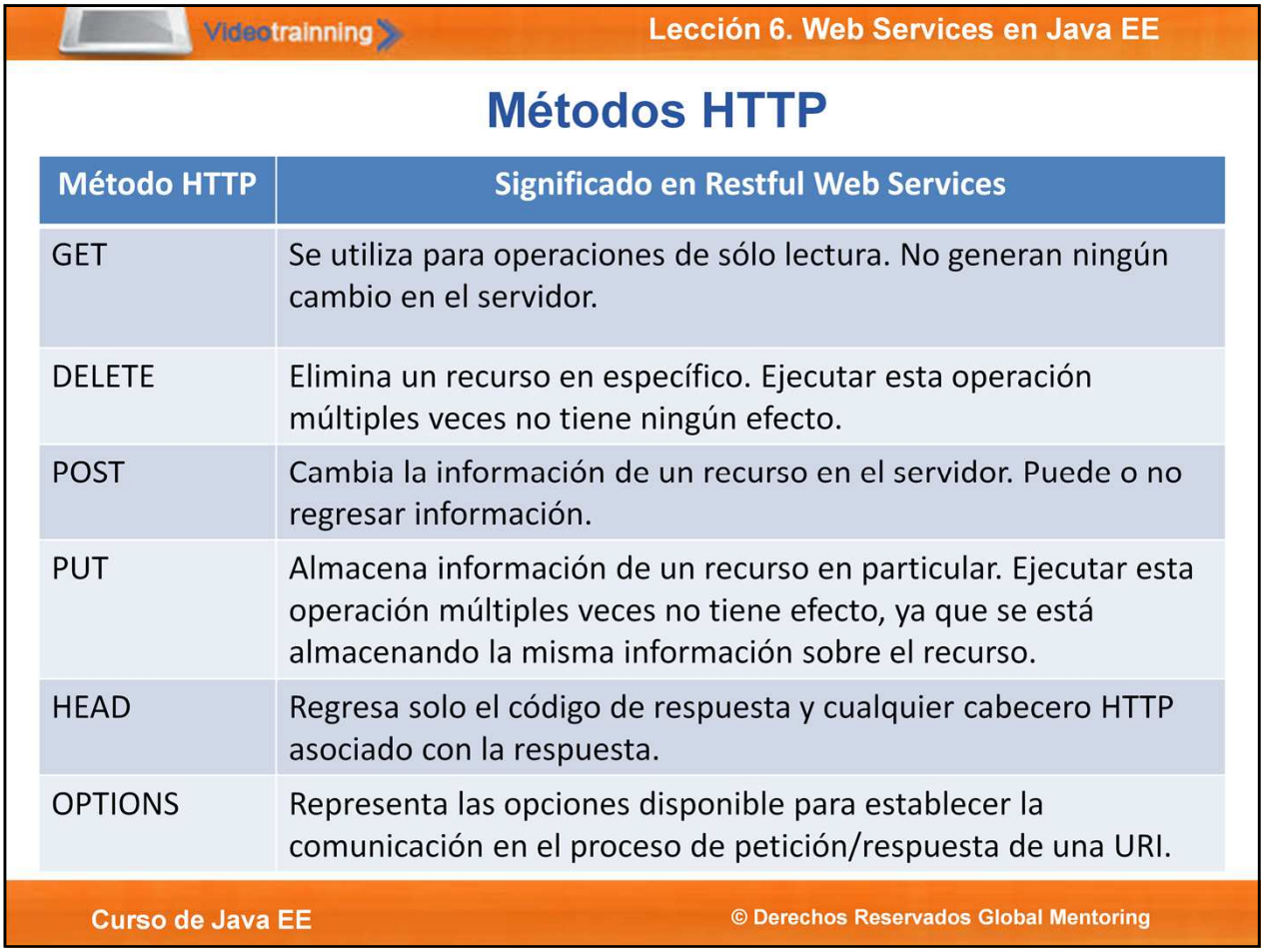
REST significa Representational State Transfer y nació por la necesidad de simplificar la creación de Web Services utilizando el protocolo HTTP como base. REST es una forma "ligera" y rápida de desarrollar y consumir Web Services. Debido a que el protocolo HTTP es utilizado prácticamente en todos lados donde utilicemos la Web, es posible reutilizar este mecanismo de comunicación como la base para la transmisión de Servicios Web.

La forma de crear un Web Services utilizando REST es a través de recursos (resources). Cada recurso tiene asociado una URI, y cada URI representa a su vez una operación del Web Service.

Ej: ***/personas*** - Es una URI que representa todas las entidades de tipo Persona de nuestro sistema.

Ej: ***/personas/{id}*** - Esta URI representa una orden en particular. A partir de esta URI, podremos leer (read), actualizar (update) y eliminar (delete) objetos de tipo Persona.

En esta lección estudiaremos a más detalle el tema de REST Web Services para exponer la lógica de negocio de nuestros EJB de Sesión de tipo Stateles.



El protocolo HTTP está definido por el consorcio www.w3.org. Un conocimiento detallado de este protocolo para el estudio de Web Services no es necesario, sin embargo conforme se involucren cada vez más en la creación de Rest Web Services, más se deberá conocer acerca de este protocolo.

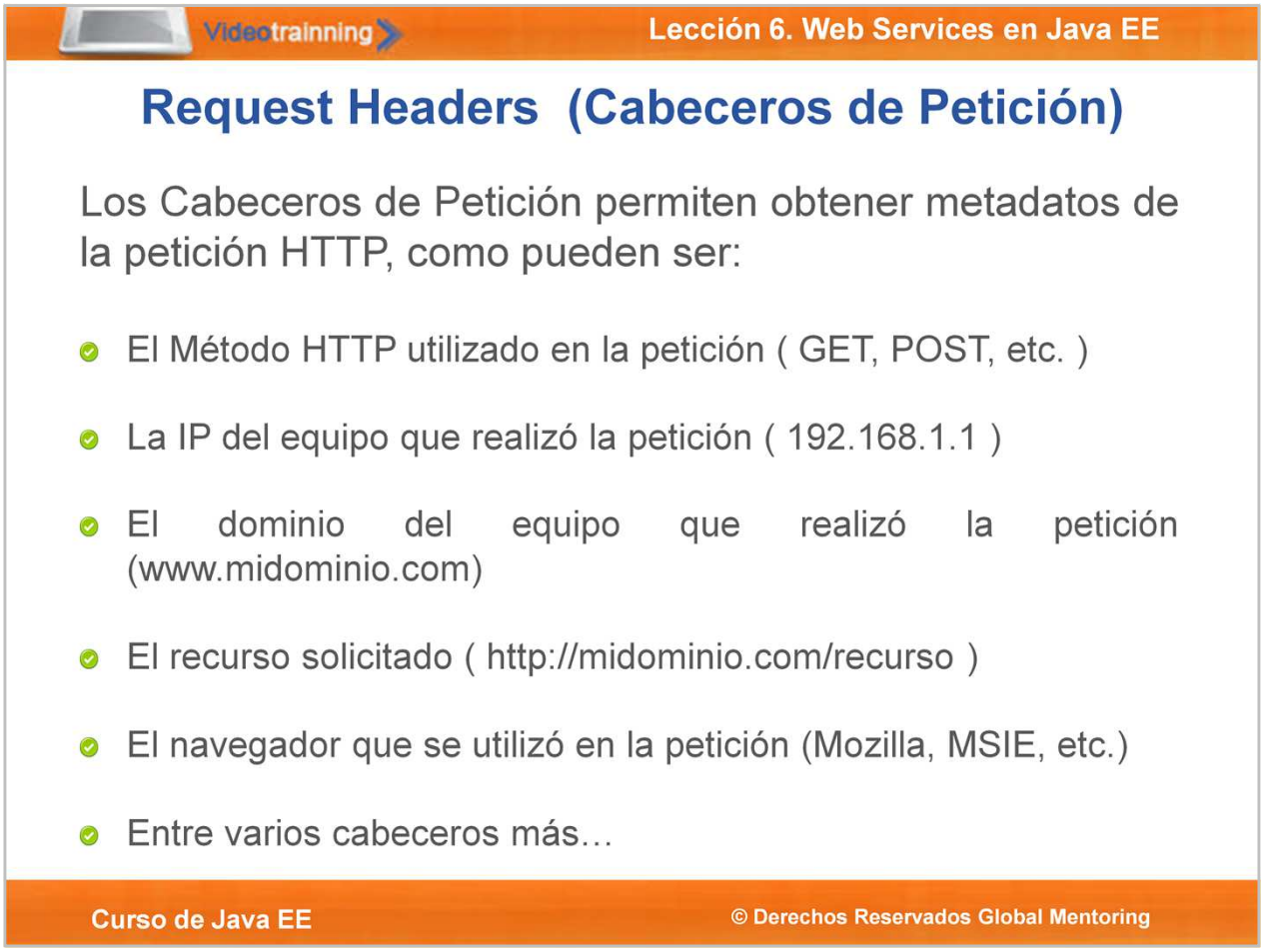
HTTP maneja 8 métodos, los cuales son: GET, DELETE, POST, PUT, HEAD, OPTIONS, TRACE y CONNECT. Cualquier petición enviada hacia un Servicio Web debe especificar cualquiera de los 6 métodos HTTP listados en la tabla. Se excluyen los métodos TRACE y CONNECT ya que no son relevantes para los Web Services de tipo RESTful.

- **GET**: Se utiliza para operaciones de sólo lectura. No generan ningún cambio en el servidor.
- **DELETE**: Elimina un recurso en específico. Ejecutar esta operación múltiples veces no tiene ningún efecto.
- **POST**: Cambia la información de un recurso en el servidor. Puede o no regresar información.
- **PUT**: Almacena información de un recurso en particular. Ejecutar esta operación múltiples veces no tiene efecto, ya que se está almacenando la misma información sobre el recurso.
- **HEAD**: Regresa solo el código de respuesta y cualquier cabecero HTTP asociado con la respuesta.
- **OPTIONS**: Representa las opciones disponible para establecer la comunicación en el proceso de petición/respuesta de una URI.

Con estos métodos debemos tomar en cuenta dos características muy importantes. Los métodos *seguros* e *idempotentes*. Los métodos *seguros (no modifican es estado del sistema)* son aquellos que no hace otra tarea que recuperar información, por ejemplo los métodos GET y HEAD. Los métodos *indempotentes* son aquellos que siempre se obtiene el mismo resultado, sin importar cuantas veces se realice cierta operación. Métodos que son *idempotentes* son: GET, HEAD, PUT y DELETE. El resto de los métodos no son ni *seguros* ni *idempotentes*.

Aunque a nivel de programación es posible realizar más de una operación al momento de enviar una petición, por ejemplo, actualizar y eliminar, esto NO debería programarse de esta manera, ya que rompe con la idea básica de los REST Web Services.

Una gran ventaja de que las operaciones REST sean sobre el protocolo HTTP es que los administradores de servidores, redes y encargados de seguridad de las mismas, saben como configurar este tipo de tráfico, por lo que no es algo nuevo para ellos.



Al enviar y recibir mensajes con REST Web Services, es necesario tener conocimiento de códigos de estado. Algunos de los códigos de estado más utilizados son:

•200 (Ok): Significa que la respuesta fue correcta, este el código de estado por default.

•204 (Sin Contenido): El navegador continua desplegando el documento previo.

•301 (Movido Permanentemente): El documento solicitado ha cambiado de ubicación, y posiblemente se indica la nueva ruta, en ese caso el navegador se redirecciona a la nueva página de manera automática.

•302 (Encontrado): El documento se ha movido temporalmente, y el navegador se mueve al nuevo url de manera automática.

•401 (Sin autorización): No se tiene permiso para visualizar el contenido solicitado, debido a que se trató de acceder a un recurso protegido con contraseña sin la autorización respectiva.

•404 (No encontrado): El recurso solicitado no se encuentra alojado en el servidor Web.

•500 (Error Interno del Servidor Web): El servidor web lanzó una excepción irrecuperable, y por lo tanto no se puede continuar procesando la petición.

Para una lista completa de los códigos de estado del protocolo HTTP se puede consultar el siguiente link:

http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Los tipos MIME(**Multipurpose Internet Mail Extensions**) son el estándar de internet para definir el tipo de información que recibirá el cliente (navegador Web) al realizar una petición al servidor Web.

Los REST Web Services pueden devolver distintos tipos de contenido para un mismo recurso, por ejemplo:

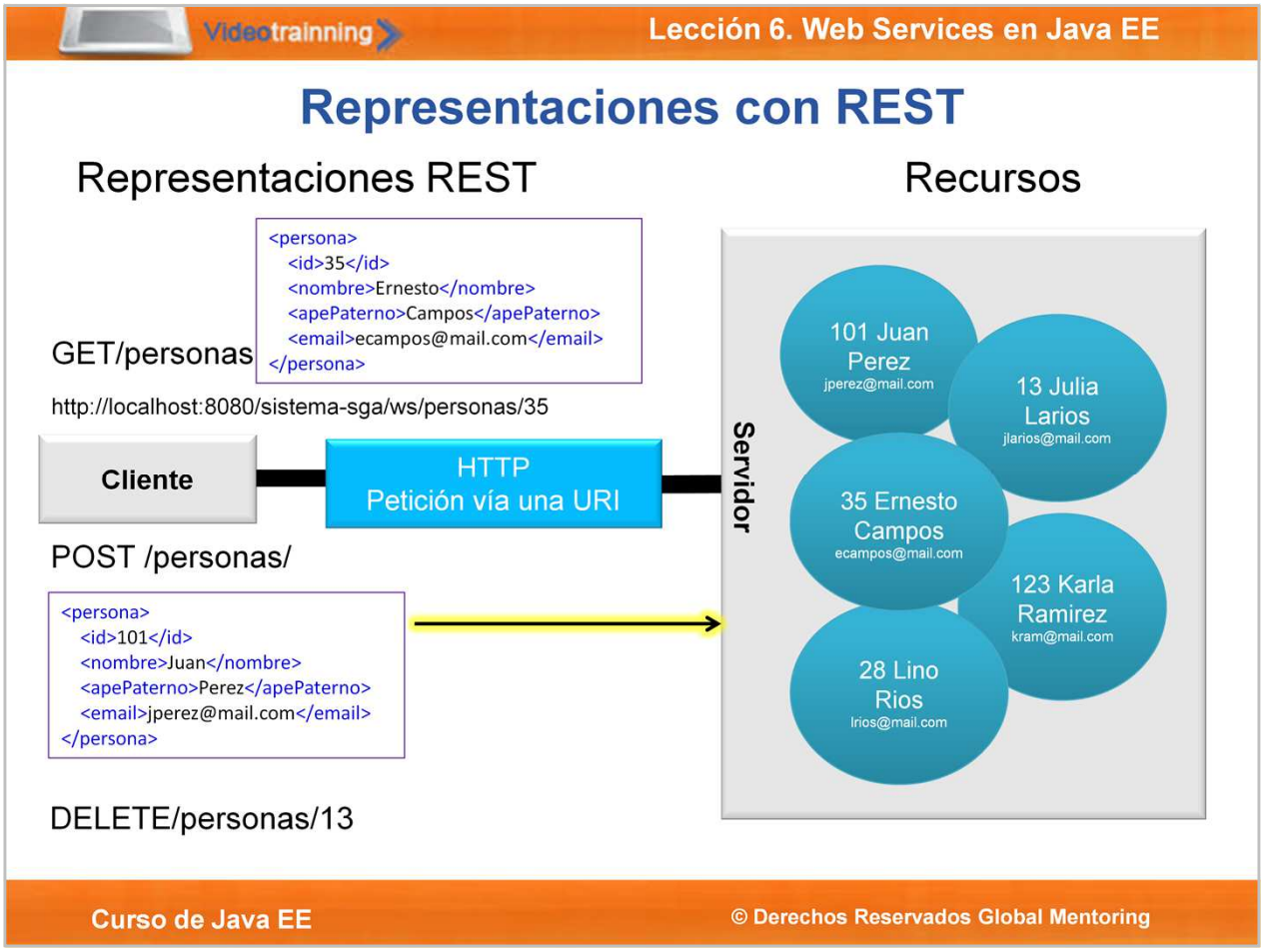
- application/xml – Mensaje XML
- application/json - Mensaje JSON
- text/html – Salida HTML
- text/plain – Salida en texto plano
- application/octet-stream – Datos binarios

Lo anterior son los tipos de recursos que más se utilizan para el envío de información en REST. Lo más común es utilizar xml, sin embargo no está limitado a este tipo de información, ya que si estamos utilizando un cliente con Jquery y AJAX, es común que utilicemos la anotación de JSON en lugar de XML.

Un listado más completo de tipos MIME es el siguiente:

- **application/msword**: Microsoft Word document
- **application/pdf**: Acrobat (.pdf) file
- **application/vnd.ms-excel**: Excel spreadsheet
- **application/vnd.ms-powerpoint**: Powerpoint presentation
- **application/zip**: Zip archive
- **audio/x-wav**: Microsoft Windows sound file
- **text/css**: HTML cascading style sheet
- **text/html**: HTML document
- **text/xml**: XML document
- **image/gif**: GIF image
- **image/jpeg**: JPEG image
- **image/png**: PNG image
- **video/mpeg**: MPEG video clip
- **video/quicktime**: QuickTime video clip

Para una lista más completa de tipos MIME, pueden consultar el siguiente link:
http://www.freeformatter.com/mime-types-list.html



Como podemos observar en la figura, REST utiliza el concepto de Representaciones, y cada representación apunta a un Recurso(s) del lado del Servidor Java.

Por ejemplo, para recuperar el objeto con id = 35, podemos utilizar el siguiente URI: http://localhost:8080/sistema-sga/ws/personas/35

Esto regresará la representación del objeto en el tipo MIME solicitado. Por ejemplo, si se solicitó una representación en XML, la respuesta debería ser:

```
<persona>
   <id>35</id>
   <nombre>Ernesto</nombre>
   <apePaterno>Campos</apePaterno>
   <email>ecampos@mail.com</email>
</persona>
```

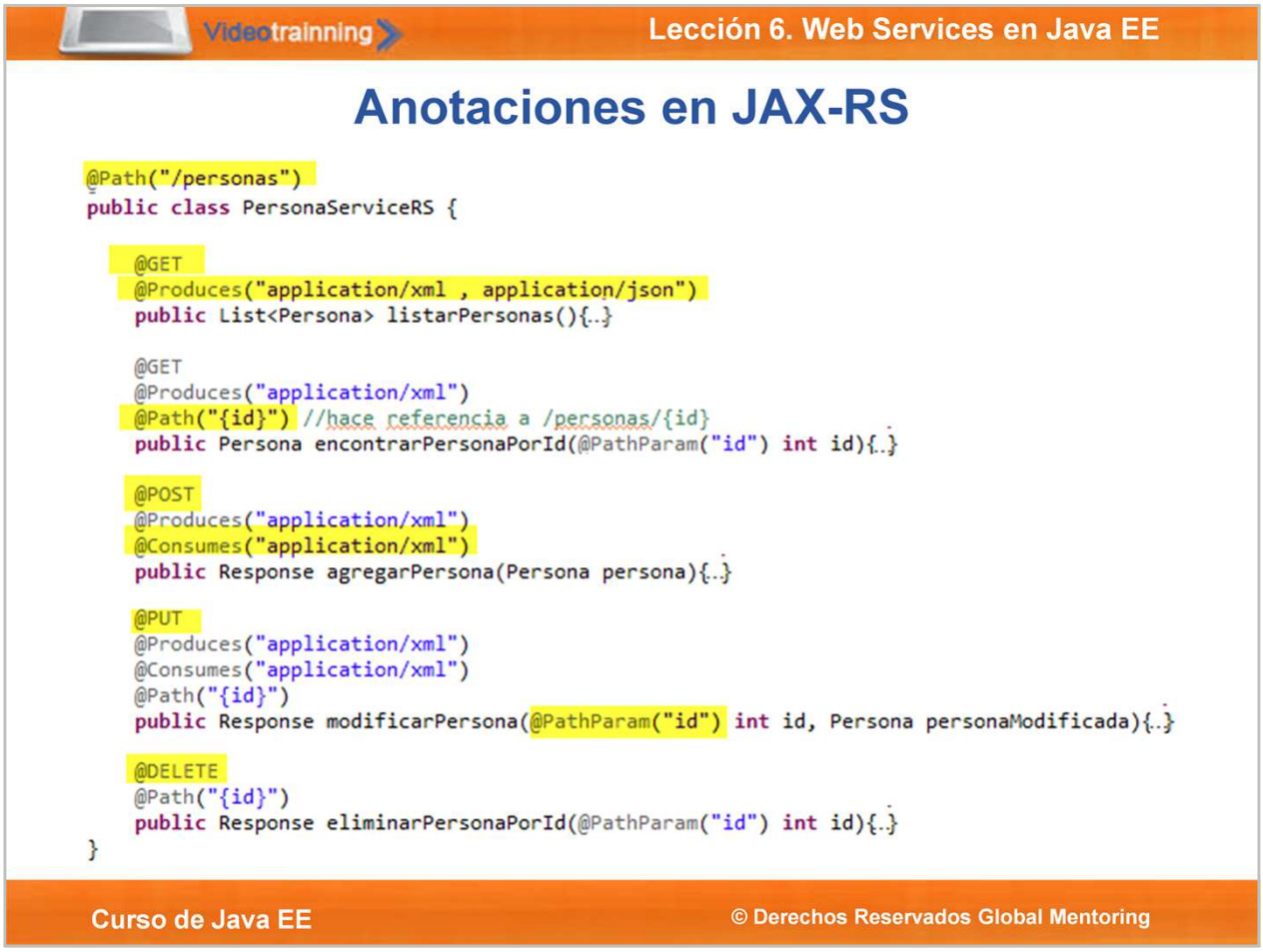
De manera similar, podemos tener las operaciones básicas para agregar, modificar y eliminar recursos del lado del servidor. Por ejemplo, las siguientes URL son ejemplos para cada una de las acciones mencinonadas:

Agregar: `POST /personas/` - Solicita agregar un nuevo recurso. Los datos se especifican en el documento XML a enviar. Ej.

```
<persona>
   <id>101</id>
   <nombre>Juan</nombre>
   <apePaterno>Perez</apePaterno>
   <email>jperez@mail.com</email>
</persona>
```

De manera similar, tenemos los ejemplos para modificar y eliminar:

Modificar: `PUT /personas/123` – Solicita modificar el recurso 123 con un XML respectivo.
Eliminar: `DELETE /personas/13` – Solicita eliminar el recurso 13



Como podemos observar en la figura, crear una clase que utilice JAX-RS para exponer un método como un servicio RESTfull Web Service en Java EE es muy simple. A continuación mencionaremos algunas de las anotaciones más utilizadas en JAX-RS, sin embargo, existen más anotaciones dependiendo del requerimiento a cubrir.

@Path: Esta anotación debe aparecer al inicio de la clase o en un método, e indica que esta clase/método se expondrá como un Servicio Web. Además, define la URI inicial del Servicio Web, la cual es relativa a la aplicación Web.

@GET, @POST, @PUT y @DELETE: Estas anotaciones se agregan a los métodos. Cada anotación representa el tipo de método HTTP que se va a utilizar. GET se utiliza para leer información, POST para agregar/modificar información. PUT se utilizar para agregar/modificar información y DELETE se utiliza para eliminar un elemento. En nuestro caso utilizaremos POST para agregar un nuevo recurso y PUT para modificar un recurso.

@PathParam: Para especificar parámetros se utilizan los signos { }. Los parámetros se adjuntan al método utilizando la anotación @PathParam. Puede haber múltiples parámetros. Ej. @Path("/personas/{tipo}/{id}")

@QueryParam: Permite procesar los parámetros del URL. Para especificar parámetros HTTP se agregan después de signo ?, y para agregar varios se utilizar el signo &. Ej. http://localhost:8080/webservice/personas?fechaInicio=01012012&fechaFin=31122012

@Produces: Indica el tipo MIME que enviará al cliente y se debe especificar por cada método. Por ejemplo: @Produces({"application/json","application/xml"})

@Consumes: Indica el tipo MIME que puede aceptar. Por ejemplo, en el caso de insertar una nueva Persona, podemos aceptar un mensaje XML indicando lo siguiente en el método a procesar la petición: @Consumes("application/xml"). JAX-RS utilizará JAXB para convertir el documento XML en una clase Java, para ello la clase Java de tipo Persona deberá tener la anotacion @XMLRootElement al inicio de la clase.

Si estamos utilizando las librerías del proyecto Jersey para desplegar REST Web Services, es necesario integrarlo con nuestra aplicación WEB, debido a que JAX-RS todavía no viene integrado de manera nativa con las aplicaciones Web.

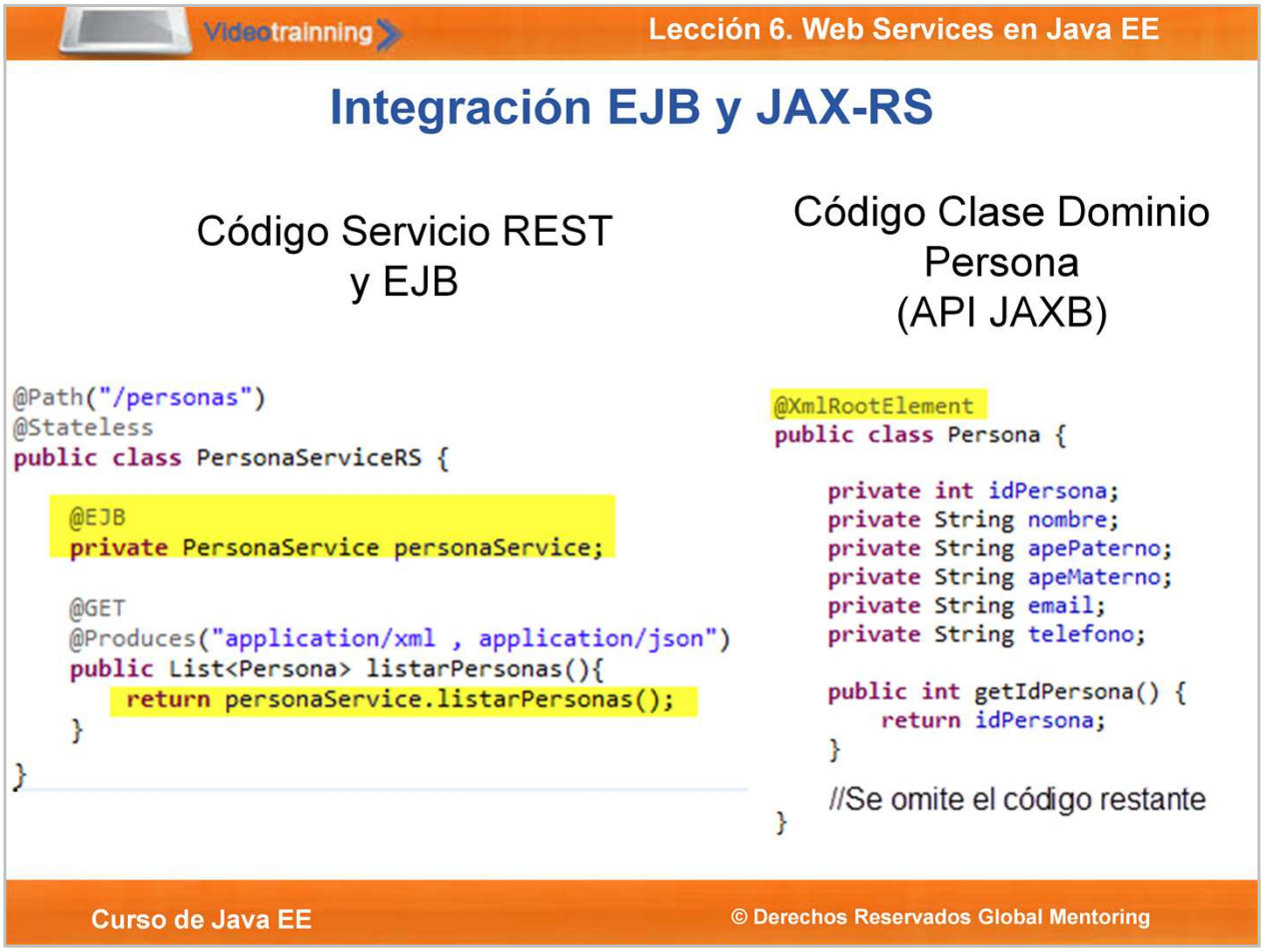
En la figura podemos observar la configuración necesaria del API del proyecto de Jersey para poder producir y consumir RESTful Web Services.

Para que la aplicación web reconozca las URI de los Servicios Web respectivos, es necesario configurar el Servlet del API de Jersey. Además, se debe especificar el elemento <servlet-mapping> con el url-pattern a utilizar.

Por ejemplo, para solicitar el recurso persona id = 101, se debe utilizar el siguiente URL:

http://localhost:8080/sistema-sga/webservices/personas/101

Podemos observar que el URI mostrado incluye el url-pattern configurado anteriormente. Sin esta configuración no es posible ejecutar los Servicios Web.



Como hemos visto los EJB proveen una serie de servicios como seguridad, transaccionalidad, entre otras características. Este simple hecho tiene varias ventajas sobre clases puras de Java.

El API JAX-RS en combinación con los EJB hace muy simple la integración entre estas tecnologías y así exponer la funcionalidad de los EJB por medio de RESTful Web Services.

Existen varias formas de lograr esta integración. En la figura se observa una forma de realizar esta integración.

Sin embargo, lo que sí se debe planear con cuidado son los URI a utilizar, ya que de esto dependerá la interface que utilizará el cliente para poder consumir los RESTful Web Services.

Como podemos observar en la figura, para exponer la funcionalidad de un método EJB podemos crear una clase enfocada a exponer únicamente los métodos de los EJB que necesitemos. Sin embargo, esta clase a su vez debe ser un EJB de tipo Stateless para poder inyectar la funcionalidad de los EJB que se desee.

Una vez realizada la inyección de dependencia del EJB, ya podemos utilizar los métodos del EJB, por ejemplo para desplegar el listado de personas.

Sin embargo, debido a que estamos regresando objetos de tipo Persona, esta clase se convierte de código Java a XML utilizando el API de JAXB. Para ello es necesario agregar la anotación @XmlRootElement a la clase Persona.

Con la configuración anterior ya tenemos listas nuestras clase Java para exponer funcionalidad de nuestros EJB como RESTful Web Services.

Cuando creamos SOAP Web Services, el documento WSDL era el elemento centrar para describir el Web Service a utilizar.

De manera similar, con RESTful Web Services tenemos un documento conocido como WADL (Web Application Description Language). Este documento XML permite describir los RESTful Web Services de manera muy similar a un documento WSDL.

Este tipo de documento XML está en sus primeras etapas de desarrollo, y no ha sido adoptado como un estándar, a diferencia de WSDL, sin embargo permite autodocumentar el Servicio WEB así como el XSD asociado a dicho Web Services.

Esto tiene varias ventajas, por ejemplo, existe una herramienta para generar el código del lado del cliente.

java -jar wadl2java.jar -o gen-src -p com.personas.rest http://localhost:8080/sistema-sga/webservice/application.wadl

Sin embargo, este tipo de herramientas han recibido varias críticas debido a que la simplicidad de la creación de clientes REST hace innecesario este tipo de herramientas.

En nuestro caso crearemos el cliente de manera manual con ayuda de las clases del proyecto Jersey, el cual hace muy simple la tarea de crear clientes Java.

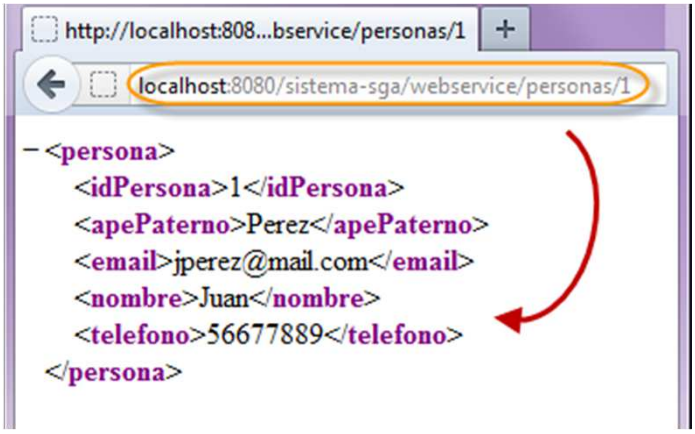
Como parte de la autodocumentación del Servicio Web, en el documento WADL se especifica el documento XSD que valida el documento XML a transmitir.

Con el siguiente link es posible acceder a este documento.

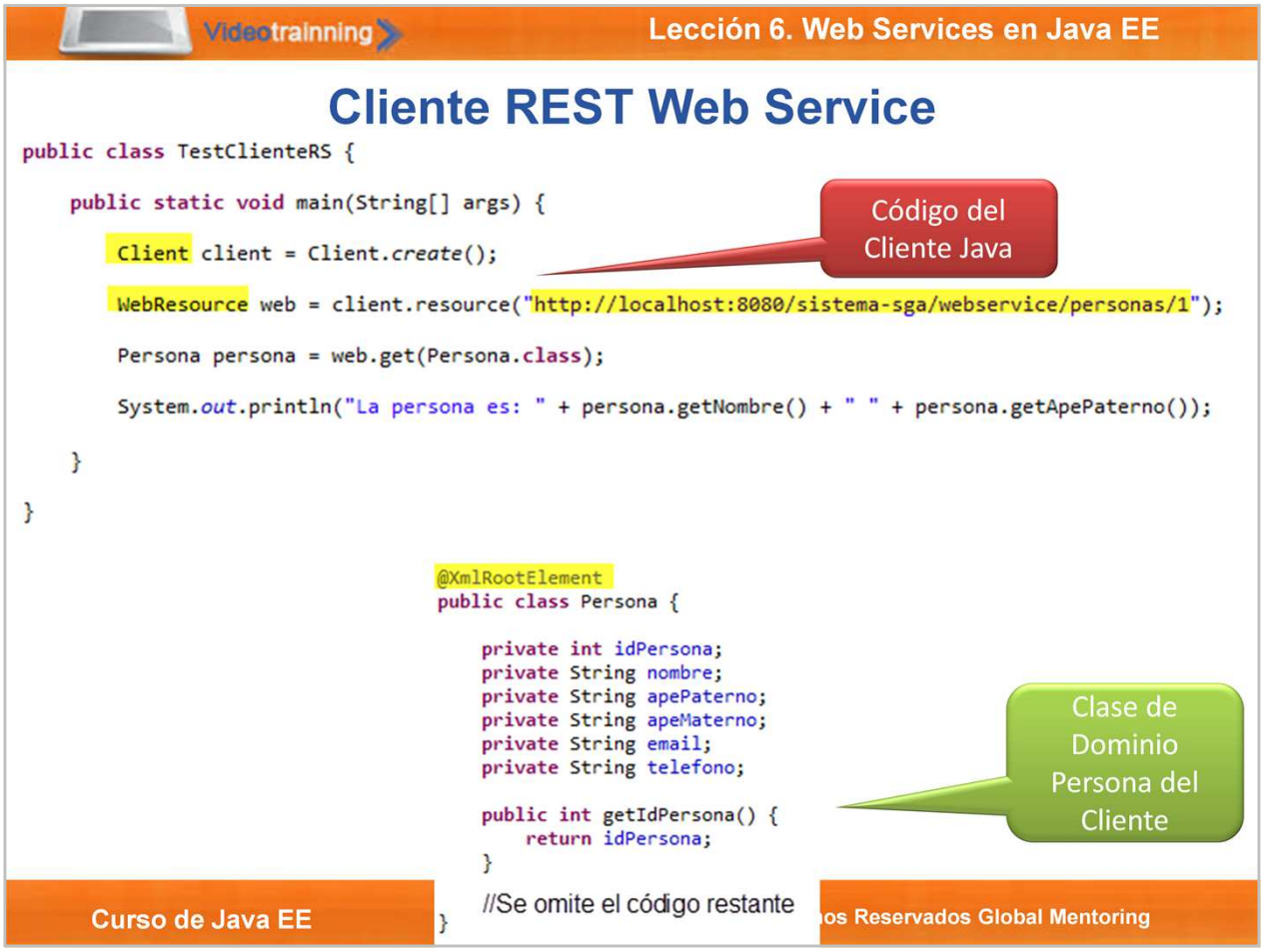
http://localhost:8080/sistema-sga/webservice/application.wadl/xsd0.xsd

Como podemos observar en la figura, debido a que se está transmitiendo entidades de tipo Persona, es necesario validar que la información XML a transmitir sea válida conforme a este documento XSD.

Un ejemplo de un XML a transmitir es el siguiente:



Con el documento XSD mostrado en la lámina, es posible validar los mensajes XML a transmitir. En nuestro ejercicio mostraremos tanto el documento WADL como el documento XSD asociado a nuestro RESTful Web Service a desplegar.



La ventaja de un servicio REST es que las peticiones GET el navegador WEB las soporta por default, así que al colocar la URL del recurso REST a buscar, obtendremos en automático la respuesta respectiva. Por ejemplo al colocar la siguiente URI y estar levantado el servidor de GlassFish o el Servicio Web ya publicado, obtendremos la respuesta siguiente:

http://localhost:8080/sistema-sga/webservice/personas/1

```
<?xml  version="1.0" encoding="UTF-8" standalone="yes" ?>
<persona>
   <idPersona>1</idPersona>
   <apePaterno>Perez</apePaterno>
   <email>jperez@mail.com</email>
   <nombre>Juan</nombre>
   <telefono>56677889</telefono>
</persona>
```

Para crear un cliente Java, el proyecto http://jersey.java.net/ proporciona varias clases para hacer muy simple el consumo de REST Web Services. En la figura se puede observar un ejemplo para consultar la persona con el `idPersona = 1`.

Al crear el ejercicio de SOAP Web Services, en automático nos creo las clases de dominio, por ejemplo, la clase de Persona. En este caso, debido a que no estamos utilizando herramientas de generación de código, es necesario crearlas manualmente. Para ello nos apoyaremos del documento WSDL, y del XSD asociado. Según hemos visto, para consultar estos documentos se debe utilizar la siguiente URL:

http://localhost:8080/sistema-sga/webservice/application.wadl

http://localhost:8080/sistema-sga/webservice/application.wadl/xsd0.xsd

Basado en el documento XSD podemos observar los atributos que necesitaremos para nuestros objetos de Entidad, necesarios para procesar la petición REST. Para ello, deberemos crear la clase Persona y agregar la anotación @XmlRootElement de JAXB en la definición de la clase Java. Estos temas los revisaremos a más detalle en el ejercicio que desarrollaremos a continuación.

Ejercicio 13

- Abrir el documento PDF de Ejercicios del cursos de Java EE.
- Realizar las siguientes prácticas:
- **Ejercicio 13**: Creación Proyecto SGA con Rest Web Services

Referencias

- La documentación del tema de Rest Web Services es la siguiente:

- http://jersey.java.net/nonav/documentation/latest/index.html
- http://docs.oracle.com/javaee/6/tutorial/doc/giliz.html

- Para un estudio más profundo del protocolo HTTP y sus métodos se pueden consultar los siguientes links:

- http://atomenabled.org/developers/protocol/atom-protocol-spec.php

- Referencia de JAXB:

- http://jaxb.java.net/
- http://docs.oracle.com/javaee/6/tutorial/doc/gkknj.html
- http://jersey.java.net/nonav/documentation/latest/xml.html#d4e820

- Lectura extra recomendada:
- http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf

En Global Mentoring promovemos la *Pasión por la Tecnología Java*.

Te invitamos a visitar nuestro sitio Web donde encontrarás cursos Java Online desde Niveles Básicos, Intermedios y Avanzados.

Además agregamos nuevos cursos para que continúes con tu preparación como consultor Java de manera profesional.

A continuación te presentamos nuestro listado de cursos en constante crecimiento:

- Fundamentos de Java
- Programación con Java
- Java con JDBC
- HTML, CSS y JavaScript
- Servlets y JSP's
- Struts Framework
- Hibernate Framework
- Spring Framework
- JavaServer Faces
- Java EE (EJB, JPA y Web Services)
- JBoss Administration

Datos de Contacto:

Sitio Web: www.globalmentoring.com.mx

Email: informes@globalmentoring.com.mx

Ayuda en Vivo: www.globalmentoring.com.mx/chat.html