Prof. MSc. Albert da Costa Xavier
albert.xavier@fiap.com.br

EJB3 – Conceitos de WebServices

Conceitos de XML

"XML (eXtensible Markup Language) é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais.

Seu propósito principal é a facilidade de compartilhamento de informações através da Internet.

Um documento XML é uma árvore rotulada onde um nó externo consiste de:

- dados de caracteres (uma sequência de texto)
- instruções de processamento (anotações para os processadores), tipicamente no cabeçalho do documento
- um comentário (nunca com semântica acompanhando)
- uma declaração de entidade (simples macros)

Um nó interno é um elemento, o qual é rotulado com:

- um nome ou
- um conjunto de atributos, cada qual consistindo de um nome e um valor.

Os documentos XML são sensíveis à letras maiúsculas e minúsculas.

Um documento XML é bem formatado quando segue algumas regras básicas. Tais regras são mais simples do que para documentos HTML e permitem que os dados sejam lidos e expostos sem nenhuma descrição externa ou conhecimento do sentido dos dados XML.

Documentos bem estruturados:

- tem casamentos das tags de início e fim
- as tags de elemento tem que ser apropriadamente posicionadas
- **XML Schema**

É uma linguagem baseada no formato XML para definição de regras de validação ("esquemas") em documentos no formato XML.

Foi a primeira linguagem de esquema para XML a obter o *status* de recomendação por parte do W3C.

Um arquivo contendo as definições na linguagem XML Schema é chamado de **XSD** (XML Schema Definition), este descreve a estrutura de um documento XML

- **Estrutura de um document XSD**

Em sua essência é um documento XML. Isso é, deve obedecer as mesmas regras que um documento XML. Um documento XSD também possui outras necessidades que um documento XML não necessita. Para que essas necessidades sejam atendidas é preciso definir as partes de um documento XSD.

Os elementos são declarados utilizando-se a tag **"element"**. Os principais atributos da tag são:

- **name**: especifica o nome do elemento
- **type**: especifica o tipo de dados do elemento
- **minOccurs**: especifica o mínima de vezes que o elemento pode aparecer
- **maxOccurs**: especifica o máxima de vezes que o elemento pode aparecer (a palavra unbounded pode ser utilizada para especificar uma quantidade ilimitada).

A declaração abaixo cria um elemento chamado "endereco" tipo string, que pode aparecer no mínimo zero (0) vezes e no máximo uma (1) vez:

```
<xsd:element name="endereco" type="xsd:string" minOccurs="0" maxOccurs="1"/>
```

- **Declaração de atributos**

De uma forma geral as declarações de atributos se parecem muito com as declarações de elementos. Essas declarações possuem alguns atributos. Os principais são:

- **name**: especifica o nome do atributo
- **type**: especifica o tipo de dados do atributo
- **use**: especifica a utilização do atributo (requerido, opcional ou proibido)

Ex.: declara dois atributos. O primeiro é do tipo data e é opcional. O segundo é do tipo string e é obrigatório.

```
<xsd:attribute name="datacadastro" type="xsd:dateTime" use="optional"/>

<xsd:attribute name="estadocivil" type="xsd:string" use="required"/>
```

- **Tipos de Dados**

O XML Schema possui diversos tipos de dados ,além da possibilidade de criar tipos próprios, os mais comuns são:

- xsd:string – string de caracteres de comprimento ilimitado
- xsd:boolean – valor booleano (true, false - 1 ou 0)
- xsd:decimal – número decimal
- xsd:float – ponto flutuante
- xsd:date – Uma data no calendário gregoriano
- xsd:dateTime – Um instante específico no calendário gregoriano
- xsd:integer – Um número inteiro

Obs.: esses dados podem ser utilizados tanto com os elementos quanto com os atributos.

- **Grupos de modelos**

Permitem que elementos sejam especificados dentro de outros elementos e, obedeçam a uma ordem ou escolha específica através de conectores (opcional). Os três conectores permitidos são: sequence, all e choice.

Ex. 1: a utilização do conector sequence, nesse caso é especificado que todos os três elementos apareçam na ordem correta. Se a ordem não for obedecida ocorrerá erro.

```
<xsd:sequence>
 <xsd:element name="nome" type="xsd:string"/>
 <xsd:element name="datanasc" type="xsd:date"/>
 <xsd:element name="telefone" type="xsd:string"/>
</xsd:sequence>
```

Ex. 2: a utilização do conector all, nesse caso é obrigado apenas que os elementos constem no documento, mas não necessariamente em uma ordem. Qualquer ordem especificada é permitida.

```
<xsd:all>
 <xsd:element name="nome" type="xsd:string"/>
 <xsd:element name="datanasc" type="xsd:date"/>
 <xsd:element name="telefone" type="xsd:string"/>
</xsd:all>
```

Ex. 3: a utilização do conector Choice, nesse caso é obrigado apenas que e somente um dos elementos especificados conste no documento. Se os dois elementos aparecerem, ocorrerá um erro.
<xsd:choice>

```
 <xsd:element name="CPF" type="xsd:string"/>
 <xsd:element name="RG" type="xsd:string"/>
</xsd:choice>
```

A utilização de conectores é necessária para a construção de elementos compostos. Esses tipos de elementos são combinações de tags aninhadas chamados de elementos complexos. Para que esses elementos sejam declarados é necessário utilizar tag “complexType”.

```
<xsd:element name="cliente">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="nome" type="xsd:string"/>
   <xsd:element name="endereco" type="xsd:string"/>
   <xsd:element name="cep" type="xsd:integer"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Existem cinco conjuntos de tipos de dados pré-definidos em XSD. Esses conjuntos de tipos de dados são:

Conjunto	Tipo	Descrição
Numéricos	xsd:float	Números reais (32bits)
	xsd:double	Números reais (64bits)
	xsd:decimal	Número decimal
	xsd:integer	Número inteiro
	xsd:nonPositiveInteger	Número inteiro negativo (incluindo 0)
	xsd:nonNegativeInteger	Número inteiro positivo (incluindo 0)
	xsd:negativeInteger	Número inteiro negativo
	xsd:positiveInteger	Número inteiro positivo
	xsd:long	Números inteiros (64bits)
	xsd:int	Números inteiros (32bits)
	xsd:short	Números inteiros (16bits)

	xsd:byte xsd:unsignedLong xsd:unsignedInt xsd:unsignedShort xsd:unsignedByte	Números inteiros (8bits) Números long positivos (incluindo 0) Números int positivos (incluindo 0) Números short positivos (incluindo 0) Números byte positivos (incluindo 0)
Data/Hora	xsd:dateTime xsd:date xsd:time xsd:gDay xsd:gMonth xsd:gYear xsd:gYearMonth xsd:gMonthDay xsd:duration	YYYY-MM-DDtHH:MM:SS.000 YYYY-MM-DD HH:MM:SS.000 Número do dia (1-31) Número do mês (1-12) Número do ano Números do ano e do mês Números do mês e do dia Período de tempo
String	xsd:string xsd:normalizedString xsd:token	Caracteres Unicode Caracteres sem CRLF nem Tabs Sem espaços
Binários	xsd:hexBinary xsd:base64Binary	Dígitos em HEX (hexadecimal) Binários em base64
Lógicos	xsd:boolean	1 | 0 | true | false

- **SOAP**

Serve para fazer a comunicação entre o cliente e o servidor através do HTTP. Através dele especificamos o endereço da máquina para qual se estabelece a comunicação.

SOAP é um protocolo simples e leve baseado em XML que proporciona troca de informações encima do protocolo HTTP, em suma, é um protocolo para acessar Web Services.

A arquitetura tem sido desenhada para ser independente de qualquer modelo particular de programa e de outras implementações específicas.

Os dois maiores objetivos do SOAP são a simplicidade e extensibilidade e esse protocolo obedece a esses requisitos pois trafega encima do http e o http é suportado por todos os servidores e browsers do mercado, com diferentes tecnologias e linguagens de programação.

- **Estrutura de um arquivo SOAP**

Analisemos uma requisição para o servidor:

```
<SOAP-ENV:envelope>
<SOAP-ENV:header>
<!--Especifica outros dados da mensagem(é opcional)-->
</SOAP-ENV:header>
<SOAP-ENV:body>
<!--O elemento BODY contém a mensagem em si-->
</SOAP-ENV:body>
</SOAP-ENV:envelope>
```

Estrutura da resposta

```
<SOAP-ENV:envelope>
<SOAP-ENV:body>
<!--A resposta do servidor-->
</SOAP-ENV:body>
</SOAP-ENV:envelope>
```

Elementos da requisição e resposta SOAP

Elemento Envelope

Responsável por definir o conteúdo da mensagem;

Elemento Header (opcional)

Dados do cabeçalho;

Elemento Body

Contém as informações de chamada de resposta ao servidor;

Elemento Fault

Tem as informações dos erros ocorridos no envio da mensagem. Esse elemento, obviamente, só aparece nas mensagens de resposta do servidor.

O namespace padrão (o namespace define as regras de codificação que o documento deve seguir) para o elemento Envelope é:

```
<soap:Envelope
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/soap:encodingstyle=http://schemas
.xmlsoap.org/soap/encoding/>

<!--Dados da mensagem aqui-->

</soap:Envelope>
```

A medida que se avança na estrutura do SOAP, é possível ver que sua codificação é bastante simples.

Atributos SOAP

A estrutura é semelhante a estrutura de atributos da XML, como qualquer outra estrutura SOAP.

actor:

O atributo actor define a URI (equivalente à URL do http) à qual o HEADER se refere. Lembrando que o elemento HEADER é opcional dentro do SOAP. Veja a sintaxe deste atributo:

```
<soap:Header>
<!--Aqui definimos o namespace como sendo "r" para personalizar nosso documento SOAP,
você pode criar o seu próprio namespace-->
<r:mercado xmlns:r="http://www.mercadao.com.br/valores/"
soap:actor="http://www.mercadao.com.br/descricao" />
<r:lingua>port</r:lingua>
<r:dinheiro>REAL</r:dinheiro>
</r:mercado>
</soap:Header>
```

encodingStyle:

Esse atributo serve para definir um estilo de codificação do documento. Veja:

```
<soap:Envelope
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
soap:encodingstyle=http://schemas.xmlsoap.org/soap/encoding/>
Mensagem aqui
</soap:Envelope>
```

mustUnderstand:

Define qual elemento do HEADER deve aparecer para o receptor da mensagem:

```
<soap:Header>
<r:mercado xmlns:r="http://www.mercadao.com.br/valores/"/>
<r:lingua soap:mustUnderstand="0">port</r:lingua>
<r:dinheiro soap:mustUnderstand="1">REAL</r:dinheiro>
</r:mercado>
</soap:Header>
```

O valor "0" deste atributo significa que o elemento não deve aparecer para o receptor, e o valor "1" significa que esse elemento deve ser visto pelo receptor da mensagem.

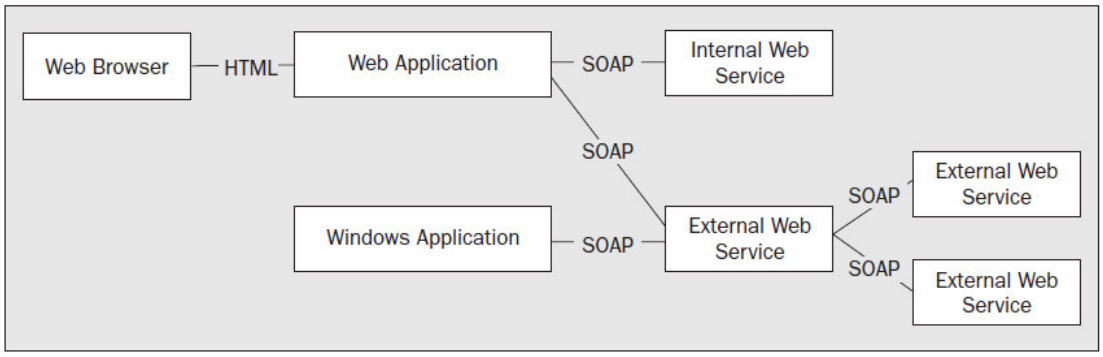
Um Web Service é uma lógica de aplicativo acessível a programas através de protocolos web de um modo independente de plataforma:

Lógica de aplicativo – um Web Service expõe a lógica de aplicativo ou de código. Este código pode realizar diversas tarefas.

Web services são acessíveis por programas ou por aplicativos

O conceito de Web Services é baseado em um conjunto de protocolos web, como HTTP, XML, SOAP, WSDL e UDDI.

Web Services podem ser implementados em qualquer plataforma. O Web Service é acessado por uma aplicação, de qualquer natureza, como mostrado na figura a seguir:



WSDL

WSDL é um formato XML para descrever serviços de rede como um conjunto de operações como endpoint nas mensagens contendo cada documento orientado ou informação orientada. As operações e mensagens são descritas de forma abstrata e então o salto para um protocolo de rede concreto e formato de mensagens para definir um ponto final.

WSDL é extensível para permitir descrição do endpoint e suas mensagens não levem em consideração de quais formatos de mensagens ou protocolos de rede são usados para comunicação.

Como protocolos de comunicação e formatos de mensagens são padronizadas na comunidade web, isso se torna crescente a possibilidade e importante ser capaz de descrever as comunicações em alguma forma de estrutura. Esses endereços WSDL necessitam pela definição de uma gramática XML para descrever serviços de rede como coleções de endpoints de comunicações capazes de trocar mensagens. As definições de serviços WSDL provêem documentação para sistemas distribuídos e servem como um recipiente para detalhes de automação envolvidos em aplicações de comunicação.

Um serviço WSDL define documentos como coleções de endpoints de rede, ou portas. No WSDL, a definição abstrata de pontos finais e mensagens são separadas de suas organizações de redes concretas ou formatos de dados de comunicação. Isto permite a reutilização de definições abstratas: mensagens, que são descrições abstratas dos dados sendo trocados, e tipos de portas que são coleções abstratas de operações. O protocolo concreto e especificações de dados concretos para uma tipo de porta em particular constituídas como comunicações reutilizáveis. Uma porta é definida associando um endereço de rede como ligação reutilizável, e uma coleção de portas definindo um serviço. Daqui, um documento WSDL usa os seguintes elementos em sua definição dos serviços de rede:

Estrutura WSDL

Types (tipos) – Um recipiente para definição de tipos de dados usando alguns tipos de sistemas. WSDL usa sintaxe XML para definir tipos de dados.

Message (mensagens) – Um resumo de definições de tipos de dados sendo trafegados, podem conter uma ou mais partes, esses partes podem ser comparadas à parâmetros de uma função.

portType (Tipo de porta) – Um resumo da configuração das operações suportadas por um ou mais endpoints.

Binding (Ligação): Define o formato da mensagem e detalhes de protocolos para cada porta.

Estrutura do WSDL:

```
<definitions>
   <types>
      definition of types........
   </types>
   <message>
      definition of a message....
   </message>
   <portType>
      definition of a port.......
   </portType>
   <binding>
      definition of a binding....
   </binding>
</definitions>
```

WSDL ports

WSDL Ports é o elemento mais importante no WSDL, ele define um WS. Suas operações desempenhadas e mensagens que estão envolvidas.

Operation Types (Tipos de operação):

- One-Way – A operação pode receber uma mensagem mas não irá retornar uma resposta.
- Request-responde – A operação pode receber uma requisição e retornar uma resposta.
- Solicit-response – A operação pode enviar uma requisição e esperar por uma resposta

Bindings SOAP

O elemento Binding tem dois atributos, name e type, o atributo name define o nome da ligação, e o type indica o ponto para a ligação da porta, nesse caso o "glossary terms port".

O elemento soap:binding tem dois atributos, style e transport, style

pode ser o **"rpc"** ou **"documento"**, o transport define o protocolo SOAP a ser usado, no caso o htt

Exercicio - Elaboração e Consumo do WebService

1. Criar um projeto **Dynamic Web Project** chamado **04_CartaoWS**.
2. Incluir a interface a seguir:

```
package br.com.fiap.ws;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService
public interface CartaoWebService {
   @WebMethod
```

```
    String validarCartao(@WebParam(name="numero") String numCartao,
                         @WebParam(name="valor") double valor);
}
```

3. Criar a classe que implementará a interface anterior:

```
package br.com.fiap.ws;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService(serviceName="CartaoWebService/cartaoendpoint")
public class CartaoWebServiceImpl implements CartaoWebService {

   @Override
   @WebMethod
   public String validarCartao(@WebParam(name = "numero") String numCartao,
        @WebParam(name = "valor") double valor) {

      if(numCartao.length() != 16){
         return "Cartão inválido";
      }
      else {
         return "Débito de " + valor + " efetuado com sucesso";
      }
   }
}
```

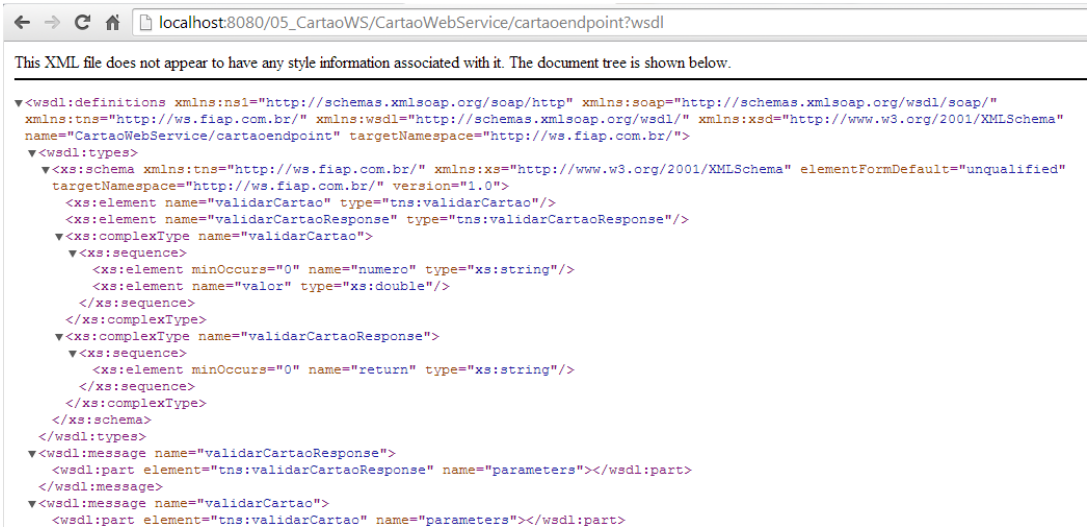
4. Iniciar o servidor, incluindo este projeto na lista de execução. Ao executá-lo, verificar a seguinte linha no log do JBoss:

```
address=http://localhost:8080/04_CartaoWS/CartaoWebService/cartaoendpoint
```

5. Abra o browser e execute a URL do item anterior, acrescentando ?wsdl ao final:

```
http://localhost:8080/04_CartaoWS/CartaoWebService/cartaoendpoint?wsdl
```

Esta URL deve produzir o resultado:



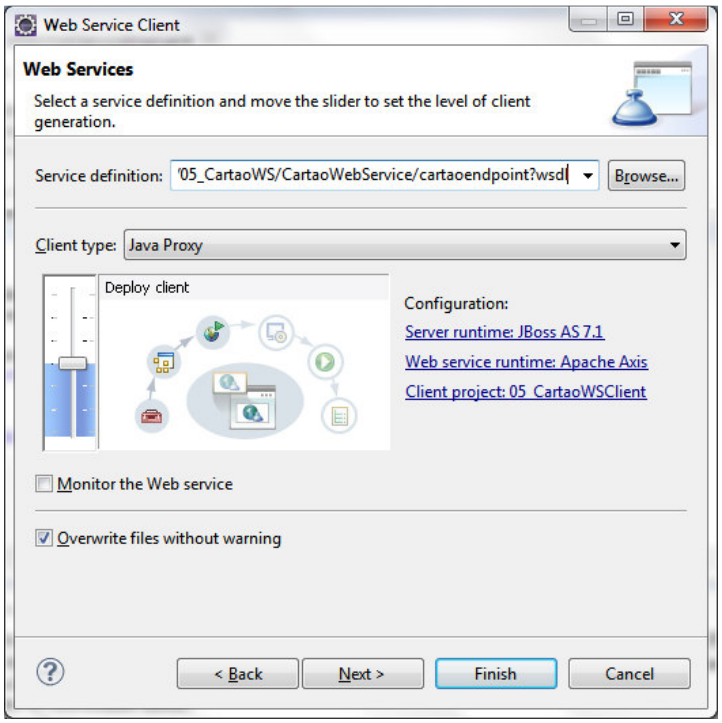
Este resultado é o conteúdo do WSDL gerado pelo WebService. Em outras palavras, é o próprio WebService em execução.

6. Para consumir o WebService, vamos criar uma aplicação cliente. Criar um projeto Java (Dynamic Web Project) chamado **04_CartaoWSClient**.

7. Na pasta **src**, clique com o direito do mouse e selecione **New ->Other...**

8. Escolha a opção Webservice -> WebService Client

9. Na tela que aparecer, copiar o endereço do WSDL que você testou.



10. Você verá que o Eclipse incluiu algumas classes utilitárias para conveniência do WebService; são os proxies. Estas classes são geradas em função do WSDL para permitir o acesso aos métodos disponibilizados pelo serviço.

11. Criar uma página JSP contendo o formulário abaixo:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <form action="cartao" method="post">
  Cartão: <input type="text" name="cartao" size="20"/><br/>
  Valor:<input type="text" name="valor" size="10"><br/>
  <input type="submit" value="Enviar"/>

  </form>
</body>
</html>
```

12. Incluir um servlet no projeto, com a seguinte classe:

```
package br.com.fiap.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.fiap.ws.CartaoWebService;
import br.com.fiap.ws.CartaoWebServiceProxy;

@WebServlet("/cartao")
public class ServletCartao extends HttpServlet {
   private static final long serialVersionUID = 1L;

    public ServletCartao() {
        super();
        // TODO Auto-generated constructor stub
    }

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
     PrintWriter out = response.getWriter();
     response.setContentType("text/html");

     try {
        String cartao = request.getParameter("cartao");
        double valor = Double.parseDouble(request.getParameter("valor"));
        CartaoWebService card = new CartaoWebServiceProxy();
        out.print(card.validarCartao(cartao, valor));
     } catch(Exception e){
        out.print(e.getMessage());
     }
   }
}
```

13. o projeto Java que você criou, incluir uma classe com o método main(). Neste método, incluir o código:

```
package br.com.fiap.ws.client;

import br.com.fiap.ws.CartaoWebService;
import br.com.fiap.ws.CartaoWebServiceProxy;

public class ClienteWS {
```

```
        public static void main(String[] args) {
           CartaoWebService ws = new CartaoWebServiceProxy();
           try {
              String cartao = "1234123412341234";
              double valor = 120;

              System.out.println(ws.validarCartao(cartao, valor));

           } catch (Exception e) {
              e.printStackTrace();
           }
        }
     }
```

Exercicio – EJB como WebService

1. Criar um projeto EJB chamado **05_EJBService**.
2. Criar um pacote chamado **br.com.fiap.ejbws**.
3. No pacote, criar uma interface chamada **Conversor** com o código abaixo:

```
package br.com.fiap.ejbws;

import javax.ejb.Remote;
import javax.jws.WebService;

@Remote
@WebService(name = "ConversorPortType",
            targetNamespace = "http://jaxws.exemplos.fiap.com.br")
public interface Conversor {

   double DolarParaReal(double dolar);
   double RealParaDolar(double real);
}
```

4. Adicionar a seguinte classe abaixo. Avaliar as anotações e a implementação:

```
package br.com.fiap.ejbws;

import javax.ejb.Stateless;
import javax.jws.WebService;

@Stateless
@WebService(serviceName = "Conversor",
        portName = "ConversorPort",
        endpointInterface = "br.com.fiap.ejbws.Conversor",
        targetNamespace = "http://jaxws.exemplos.fiap.com.br")
public class ConversorBean implements Conversor{
```

```
   private double valorDolar = 2.25;
   @Override
   public double DolarParaReal(double dolar) {
      return dolar * valorDolar;
   }

   @Override
   public double RealParaDolar(double real) {
      return real / valorDolar;
   }
}
```

5. Remover os projetos do JBoss e adicionar seu novo projeto. Iniciar o servidor, e verificar no browser o WSDL.

6. Criar uma aplicação cliente. Definir um novo projeto chamado **05_EJBServiceClient**, tipo Dynamic Web Project.

7. Adicionar as classes através da opção **New -> Web Service -> Web Service Client**.

8. No seu novo projeto, adicionar um Servlet, como codificado abaixo:

```
package br.com.fiap.ejbws.client;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.rpc.ServiceException;

import br.com.fiap.exemplos.jaxws.Conversor;
import br.com.fiap.exemplos.jaxws.ConversorLocator;
import br.com.fiap.exemplos.jaxws.ConversorPortType;

@WebServlet("/conversor")
public class ServletConversor extends HttpServlet {
   private static final long serialVersionUID = 1L;

    public ServletConversor() {
        super();
    }

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
     PrintWriter out = response.getWriter();
     response.setContentType("text/html");
     double valor = 100;
     double real4dolar, dolar4real;

     try {
        Conversor service = new ConversorLocator();
```

```
        ConversorPortType port = service.getConversorPort();
        real4dolar = port.realParaDolar(valor);
        dolar4real = port.dolarParaReal(valor);
        out.print(valor + " reais = " + real4dolar + " dolares");
        out.print("<br/>");
        out.print(valor + " dolares = " + dolar4real + " reais");

      } catch (ServiceException e) {

        e.printStackTrace();
      }



   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
   }
}
```

9. Testar a aplicação.

Bom trabalho!