Universidade do Minho
Departamento de Informática

Engenharia de Aplicações

Web Services

António Nestor Ribeiro
(anr@di.uminho.pt)

Web Services: serviços básicos

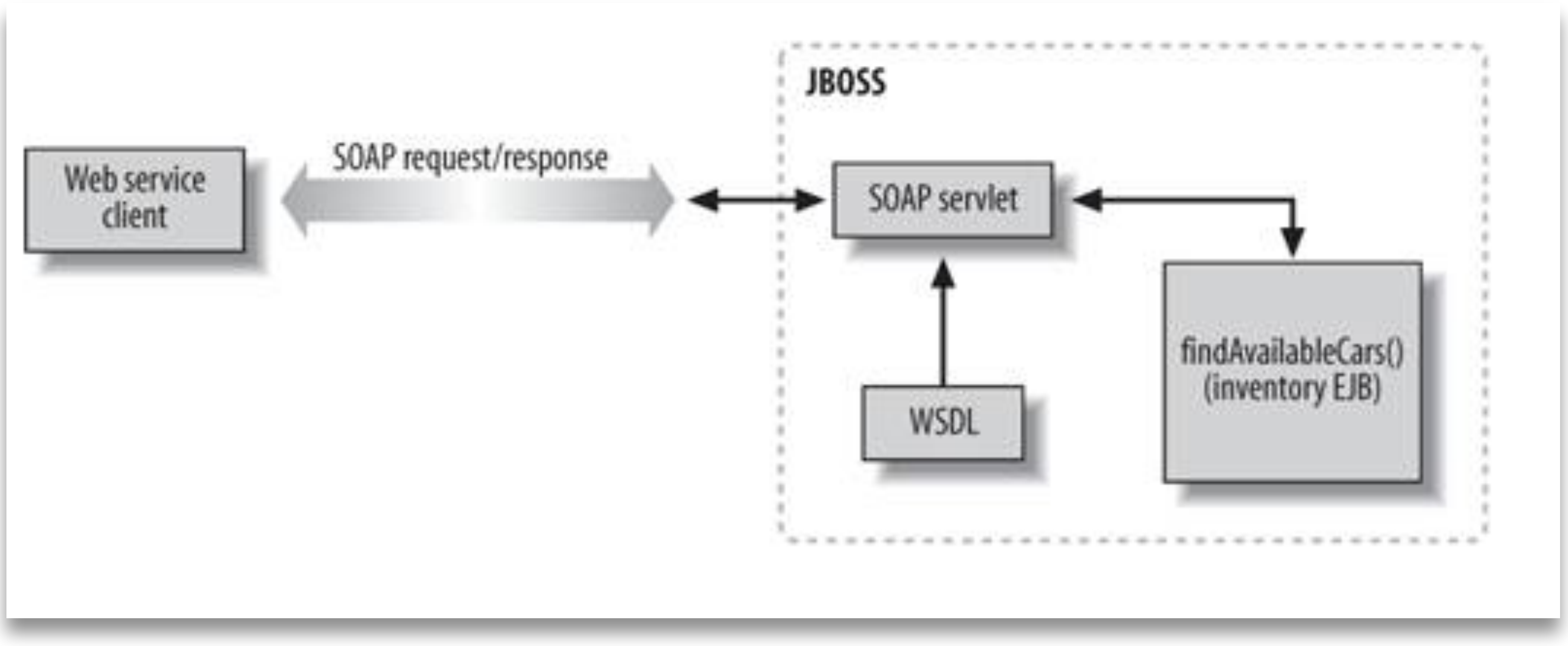
- Standard *de facto* para inter-operabilidade entre sistemas
  - independente da tecnologia existente em cada uma das extremidades
  - independente do hardware
  - independente do sistema
  - funciona sobre protocolos conhecidos (HTTP)
- Baseado em XML, WSDL e SOAP
  - adoptado por: Microsoft, IBM, BEA, JBOSS, Oracle, HP, etc.
- No mundo do JEE:
  - Java API for XML Web Services (JAX-WS)
  - Java API for XML-based RPC (JAX-RPC)
  - SOAP with Attachments API for Java (SAAJ)
  - Java API for XML Registries (JAXR)

Web Services: âmbito

- Um Web Service é uma interface para uma aplicação remota:
  - descrita de acordo com WSDL
  - acessível via SOAP
- Baseia-se num esquema XML onde se descreve a informação necessária:
  - o quê
  - onde
  - conteúdo
  - serviços
  - parâmetros
  - tipos de dados dos parâmetros

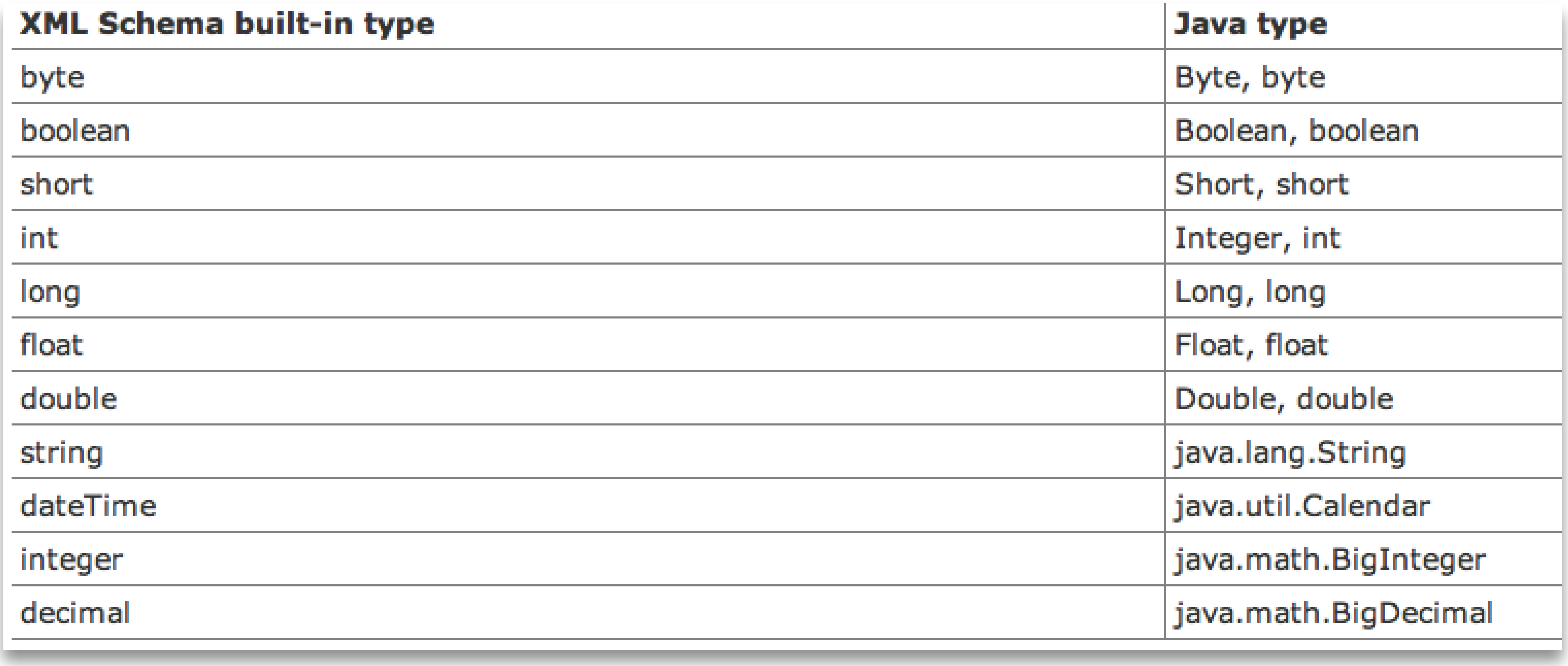
WebServices: arquitectura

- Modelo de invocação



Web Services: tipos de dados

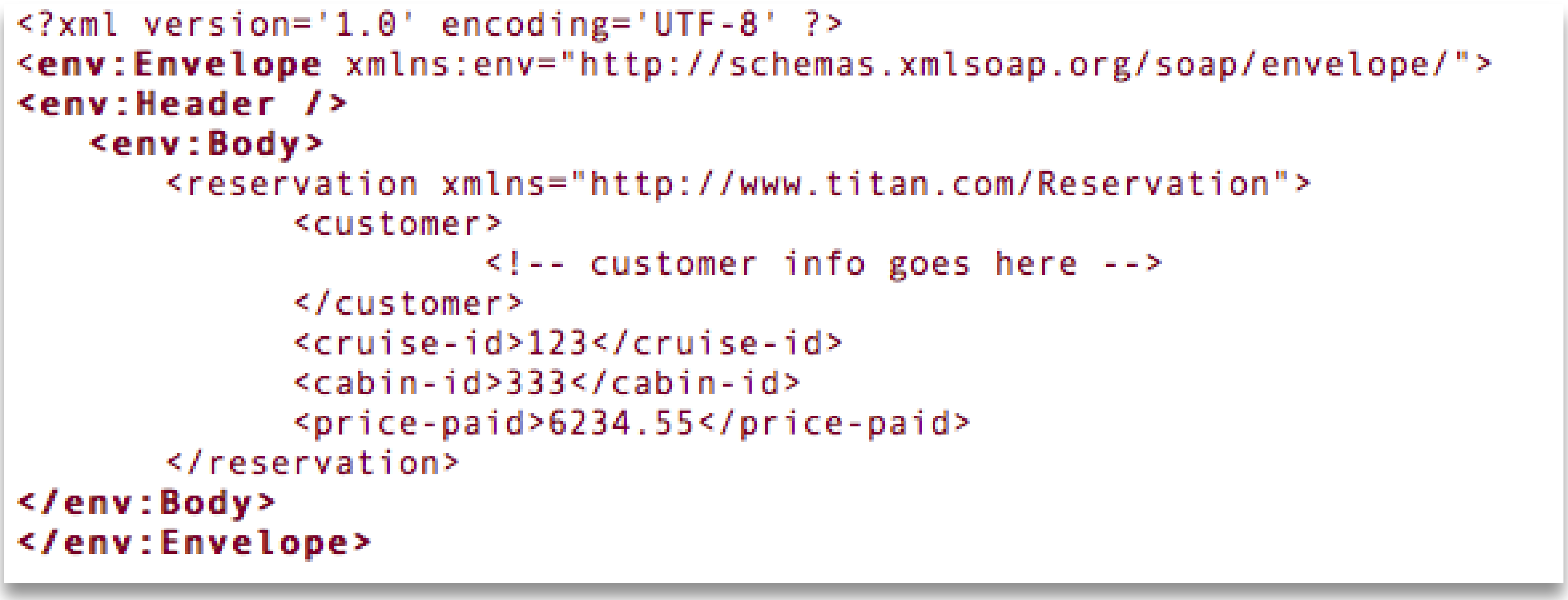
- Os tipos de dados devem ser independentes da plataforma tecnológica
- Para Java a correspondência é:



XML Schema built-in type	Java type
byte	Byte, byte
boolean	Boolean, boolean
short	Short, short
int	Integer, int
long	Long, long
float	Float, float
double	Double, double
string	java.lang.String
dateTime	java.util.Calendar
integer	java.math.BigInteger
decimal	java.math.BigDecimal

Web Services: SOAP

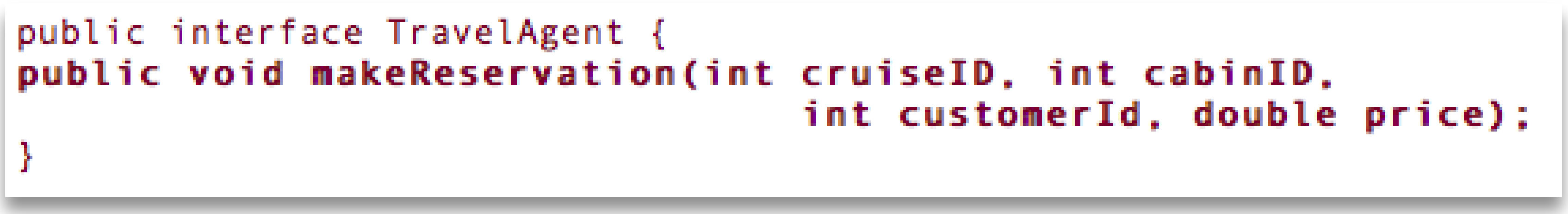
- SOAP é um protocolo distribuído para troca de informação.
- Cada mensagem SOAP é enviada como sendo um documento XML



```xml
<?xml version='1.0' encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Header />
   <env:Body>
       <reservation xmlns="http://www.titan.com/Reservation">
             <customer>
                      <!-- customer info goes here -->
             </customer>
             <cruise-id>123</cruise-id>
             <cabin-id>333</cabin-id>
             <price-paid>6234.55</price-paid>
       </reservation>
</env:Body>
</env:Envelope>
```

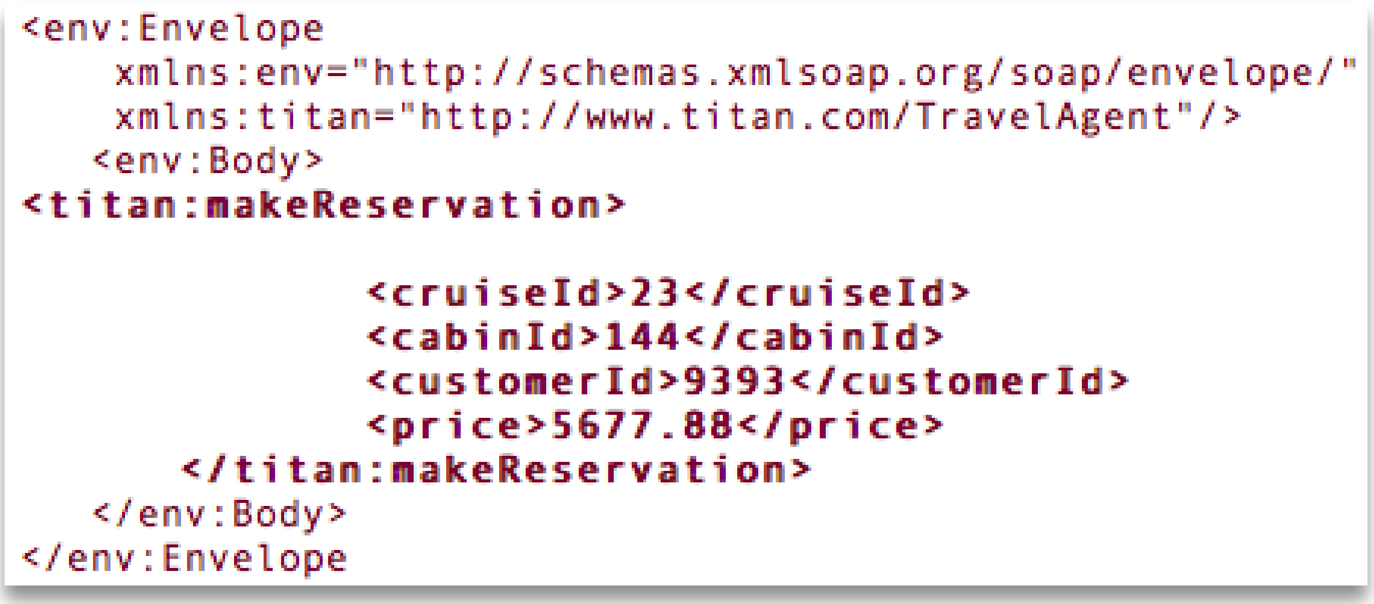
Web Services: SOAP

- Para o serviço:



```
public interface TravelAgent {
public void makeReservation(int cruiseID, int cabinID,
                                        int customerId, double price);
}
```

- A mensagem SOAP é:



```
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:titan="http://www.titan.com/TravelAgent"/>
   <env:Body>
<titan:makeReservation>

              <cruiseId>23</cruiseId>
              <cabinId>144</cabinId>
              <customerId>9393</customerId>
              <price>5677.88</price>
      </titan:makeReservation>
   </env:Body>
</env:Envelope
```

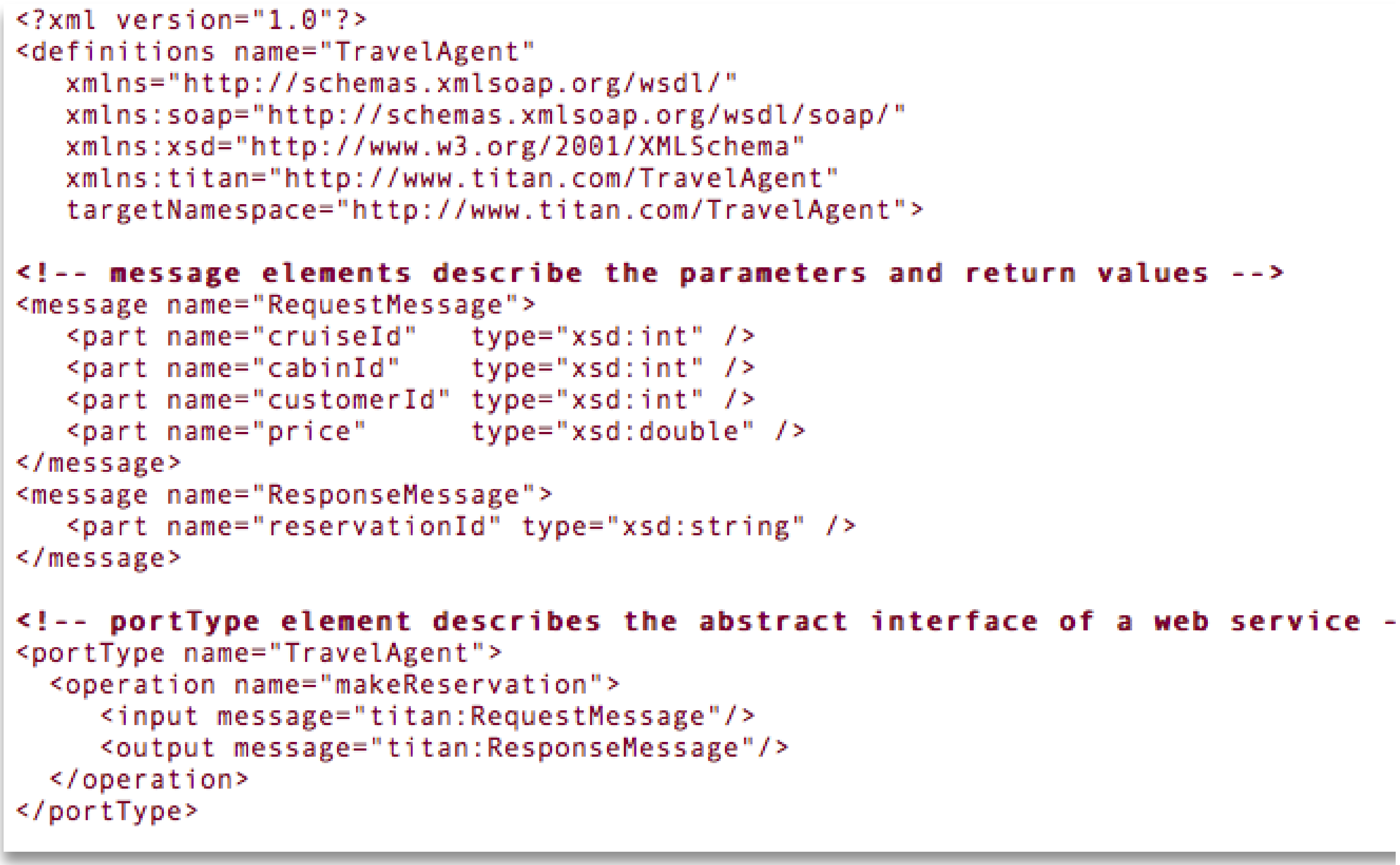
Web Services: WSDL

- A linguagem WSDL é um documento XML utilizado para descrever um serviço Web.
- A WSDL é agnóstica em relação aos protocolos o que significa que pode ser utilizada para descrever web services que não utilizem SOAP e HTTP

- O objectivo do WSDL é expor a descrição do serviço, por forma a que uma aplicação cliente o possa invocar.

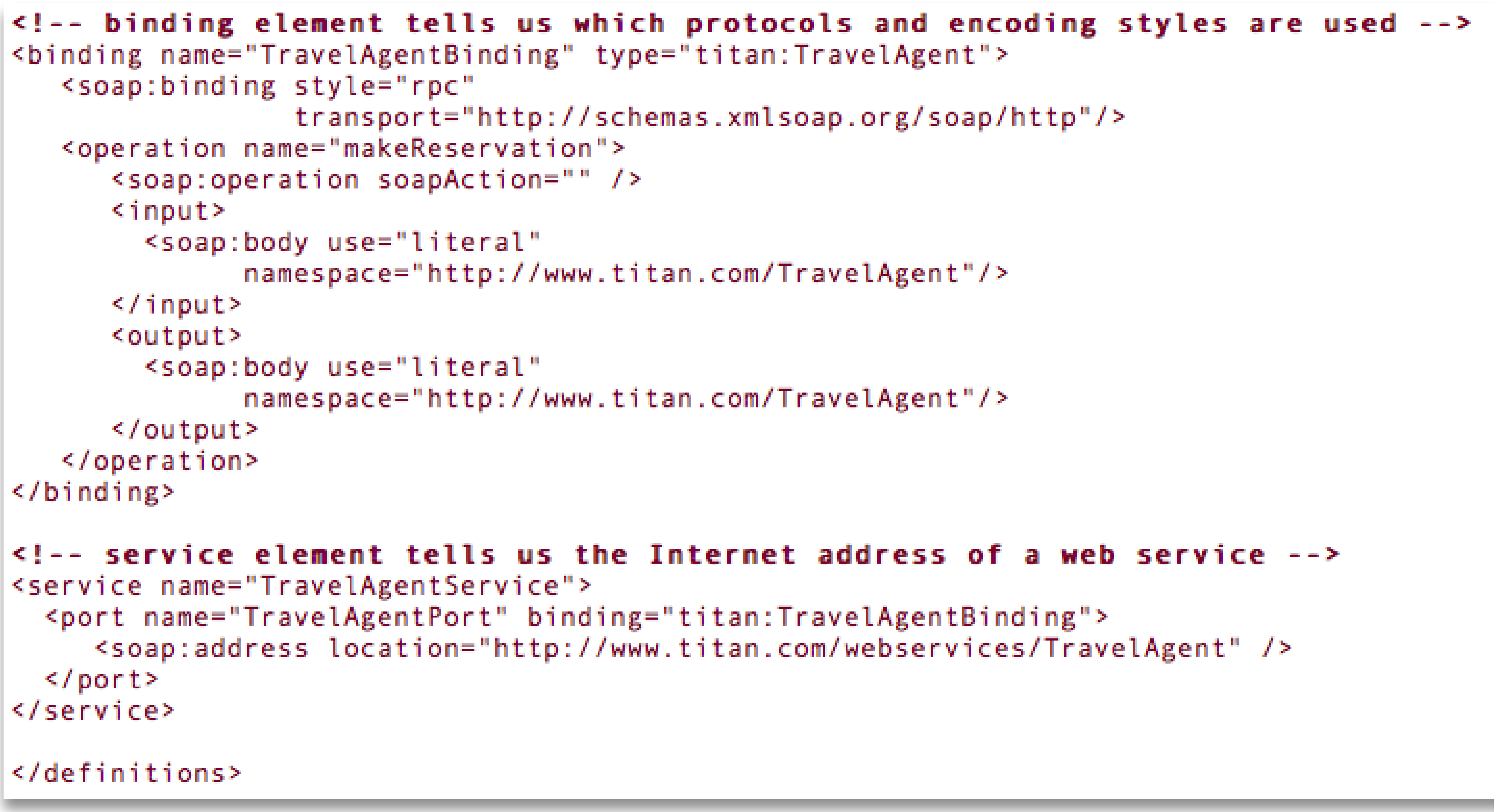
- Para o interface remoto que disponibiliza o método `makeReservation`, teremos a seguinte descrição WSDL

WSDL: Parte I



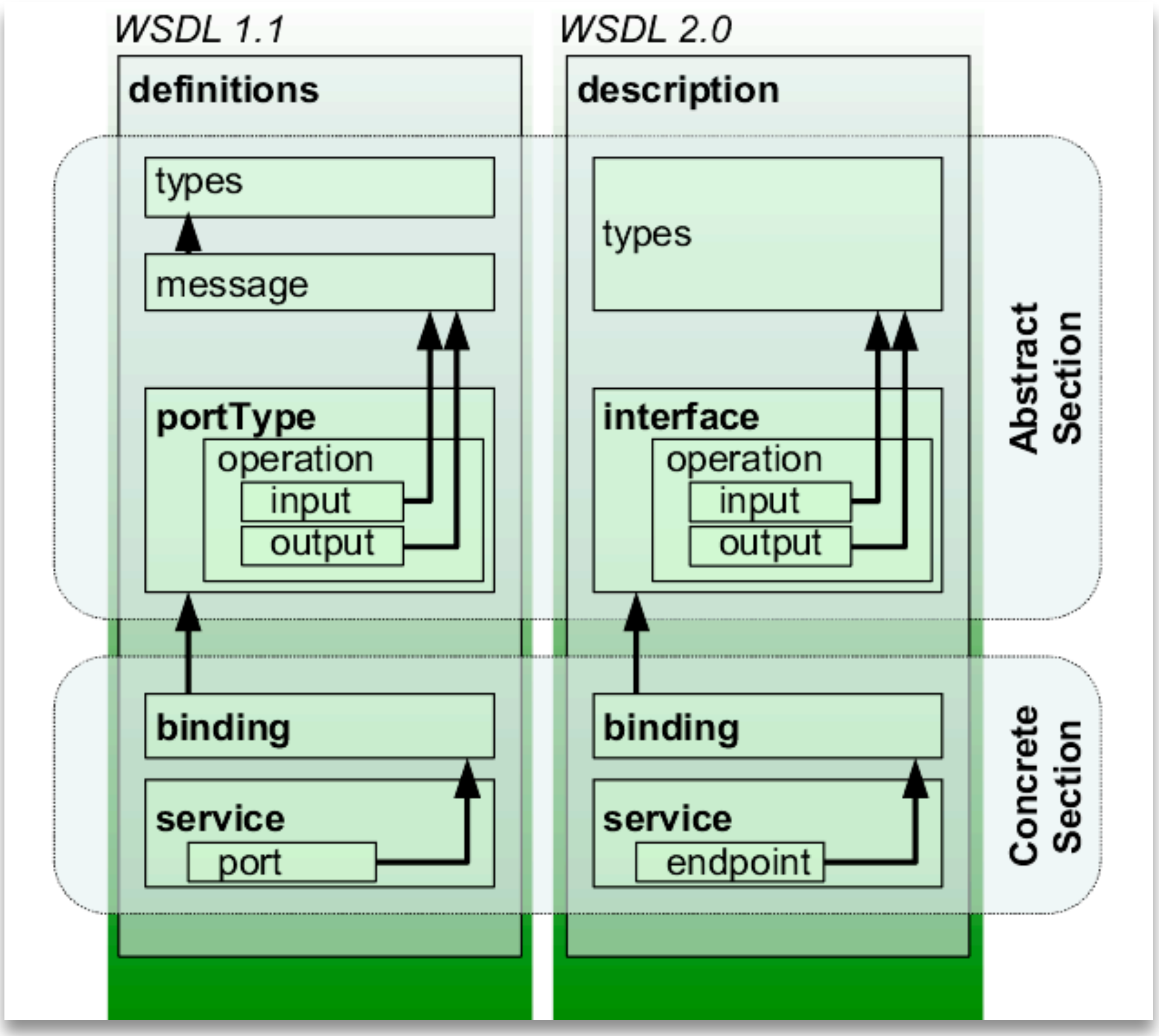
```
<?xml version="1.0"?>
<definitions name="TravelAgent"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:titan="http://www.titan.com/TravelAgent"
   targetNamespace="http://www.titan.com/TravelAgent">

<!-- message elements describe the parameters and return values -->
<message name="RequestMessage">
   <part name="cruiseId"   type="xsd:int" />
   <part name="cabinId"    type="xsd:int" />
   <part name="customerId" type="xsd:int" />
   <part name="price"      type="xsd:double" />
</message>
<message name="ResponseMessage">
   <part name="reservationId" type="xsd:string" />
</message>

<!-- portType element describes the abstract interface of a web service -
<portType name="TravelAgent">
  <operation name="makeReservation">
     <input message="titan:RequestMessage"/>
     <output message="titan:ResponseMessage"/>
  </operation>
</portType>
```

WSDL: Parte II



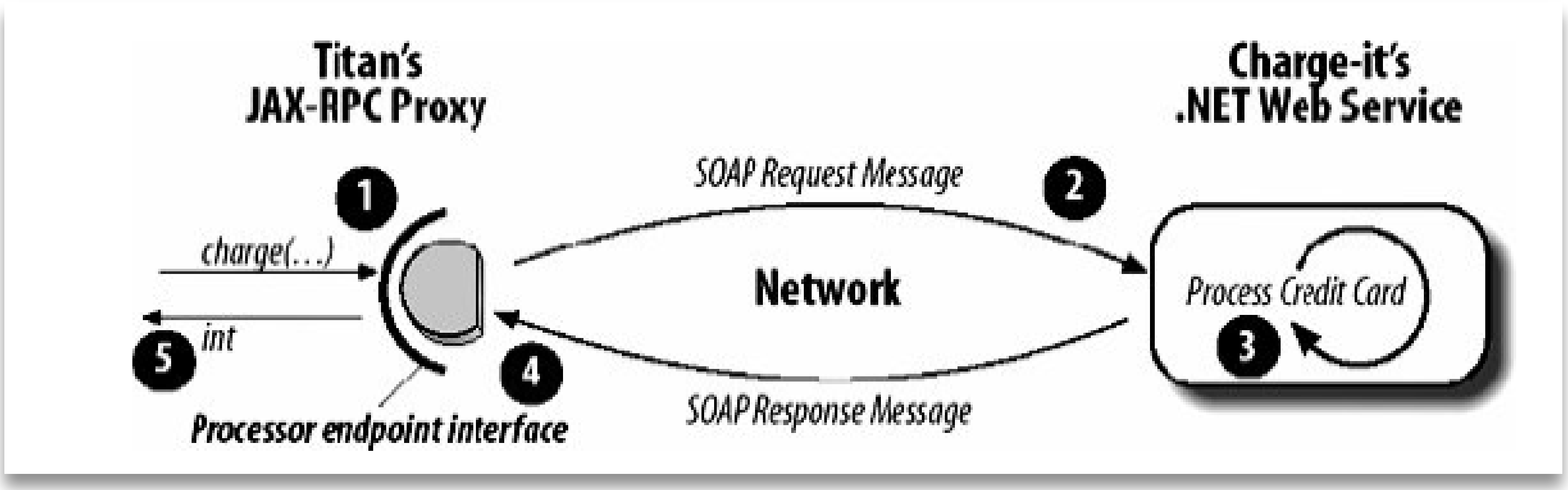
```
<!-- binding element tells us which protocols and encoding styles are used -->
<binding name="TravelAgentBinding" type="titan:TravelAgent">
   <soap:binding style="rpc"
                 transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="makeReservation">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"
              namespace="http://www.titan.com/TravelAgent"/>
      </input>
      <output>
        <soap:body use="literal"
              namespace="http://www.titan.com/TravelAgent"/>
      </output>
   </operation>
</binding>

<!-- service element tells us the Internet address of a web service -->
<service name="TravelAgentService">
  <port name="TravelAgentPort" binding="titan:TravelAgentBinding">
     <soap:address location="http://www.titan.com/webservices/TravelAgent" />
  </port>
</service>

</definitions>
```

WSDL 1.1 vs WSDL 2.0

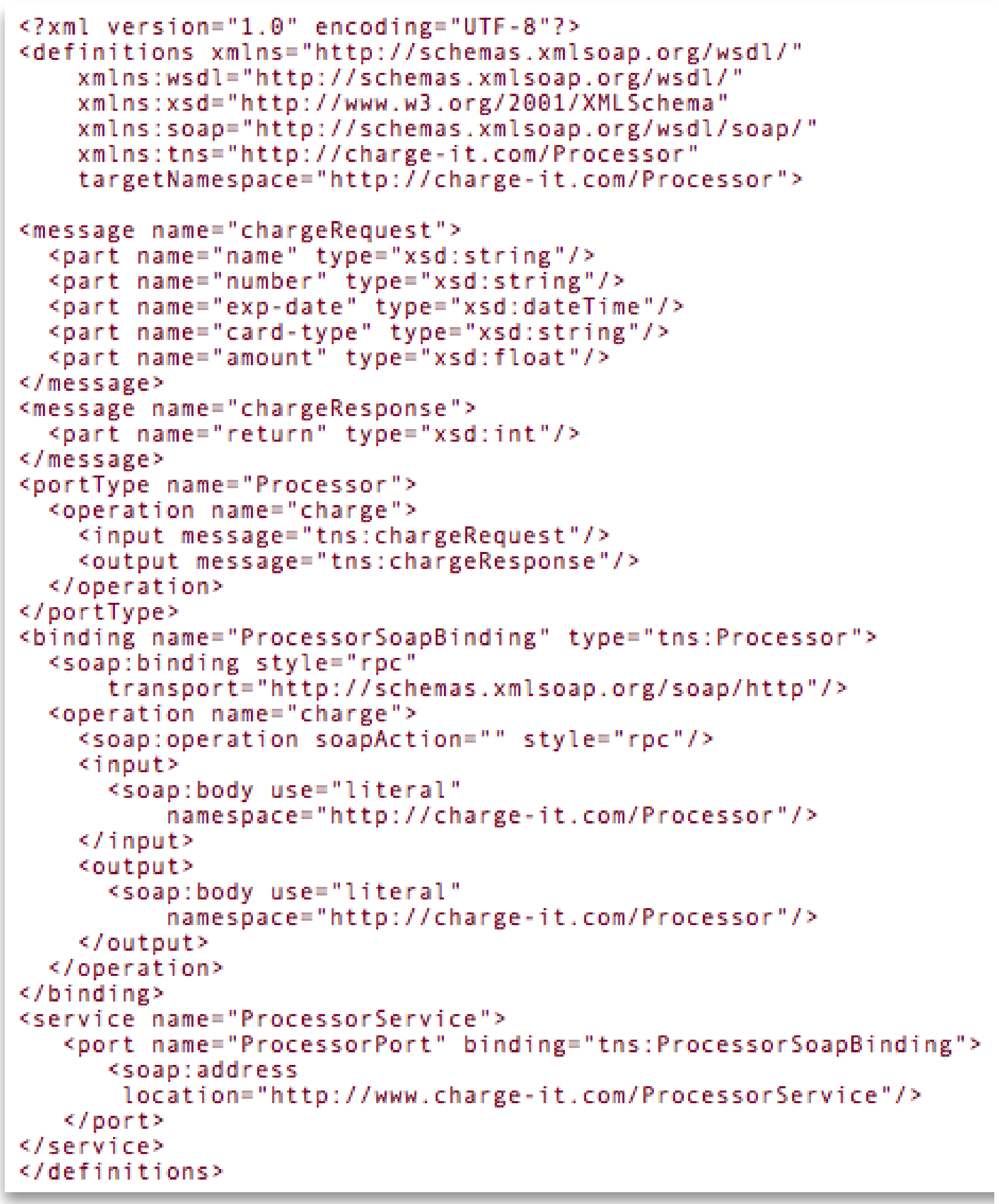


EJB3.0 e Web Services

- Neste exemplo com a utilização de JAX-RPC
  - Modo de funcionamento

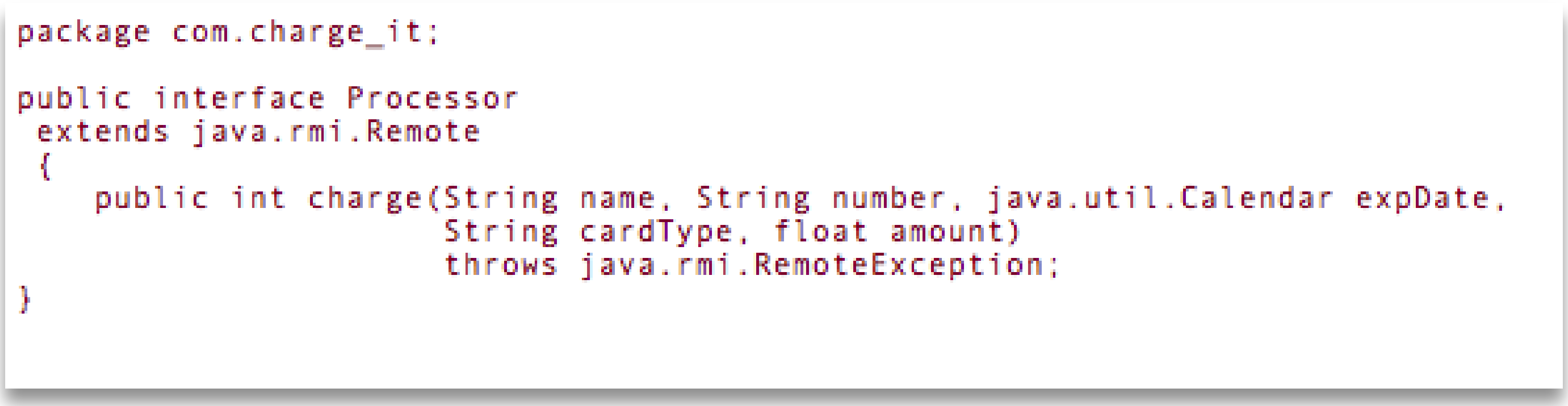


- Criação de um Web Service para o pagamento de uma viagem.
  - Ex: pagamento a um interface financeiro (método **charge**)



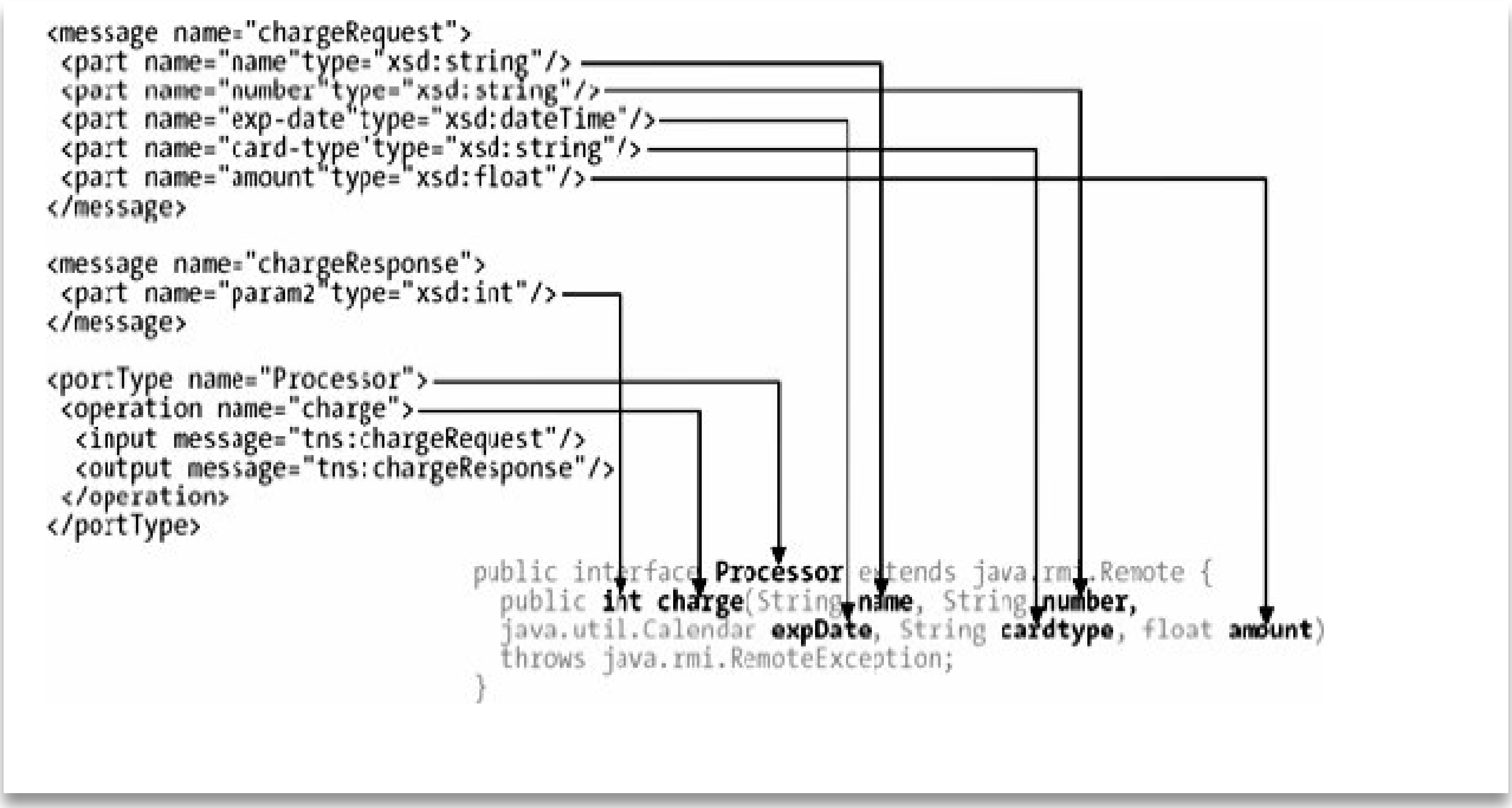
```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://charge-it.com/Processor"
    targetNamespace="http://charge-it.com/Processor">

<message name="chargeRequest">
  <part name="name" type="xsd:string"/>
  <part name="number" type="xsd:string"/>
  <part name="exp-date" type="xsd:dateTime"/>
  <part name="card-type" type="xsd:string"/>
  <part name="amount" type="xsd:float"/>
</message>
<message name="chargeResponse">
  <part name="return" type="xsd:int"/>
</message>
<portType name="Processor">
  <operation name="charge">
    <input message="tns:chargeRequest"/>
    <output message="tns:chargeResponse"/>
  </operation>
</portType>
<binding name="ProcessorSoapBinding" type="tns:Processor">
  <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="charge">
    <soap:operation soapAction="" style="rpc"/>
    <input>
      <soap:body use="literal"
          namespace="http://charge-it.com/Processor"/>
    </input>
    <output>
      <soap:body use="literal"
          namespace="http://charge-it.com/Processor"/>
    </output>
  </operation>
</binding>
<service name="ProcessorService">
   <port name="ProcessorPort" binding="tns:ProcessorSoapBinding">
      <soap:address
       location="http://www.charge-it.com/ProcessorService"/>
   </port>
</service>
</definitions>
```

- O que a aplicação deve fazer é criar um método que responda ao serviço



```
package com.charge_it;

public interface Processor
 extends java.rmi.Remote
 {
    public int charge(String name, String number, java.util.Calendar expDate,
                      String cardType, float amount)
                      throws java.rmi.RemoteException;
}
```

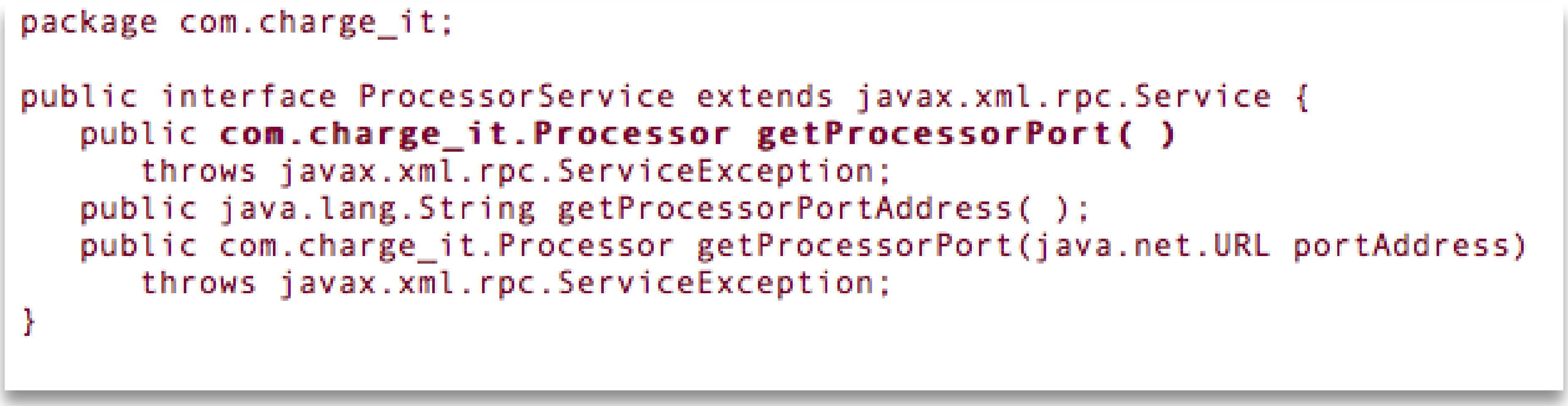
- É necessário estabelecer o mapeamento entre os tipos de dados da mensagem WSDL e os parâmetros da operação.



```
<message name="chargeRequest">
 <part name="name"type="xsd:string"/>
 <part name="number"type="xsd:string"/>
 <part name="exp-date"type="xsd:dateTime"/>
 <part name="card-type'type="xsd:string"/>
 <part name="amount"type="xsd:float"/>
</message>

<message name="chargeResponse">
 <part name="param2"type="xsd:int"/>
</message>

<portType name="Processor">
 <operation name="charge">
  <input message="tns:chargeRequest"/>
  <output message="tns:chargeResponse"/>
 </operation>
</portType>
```

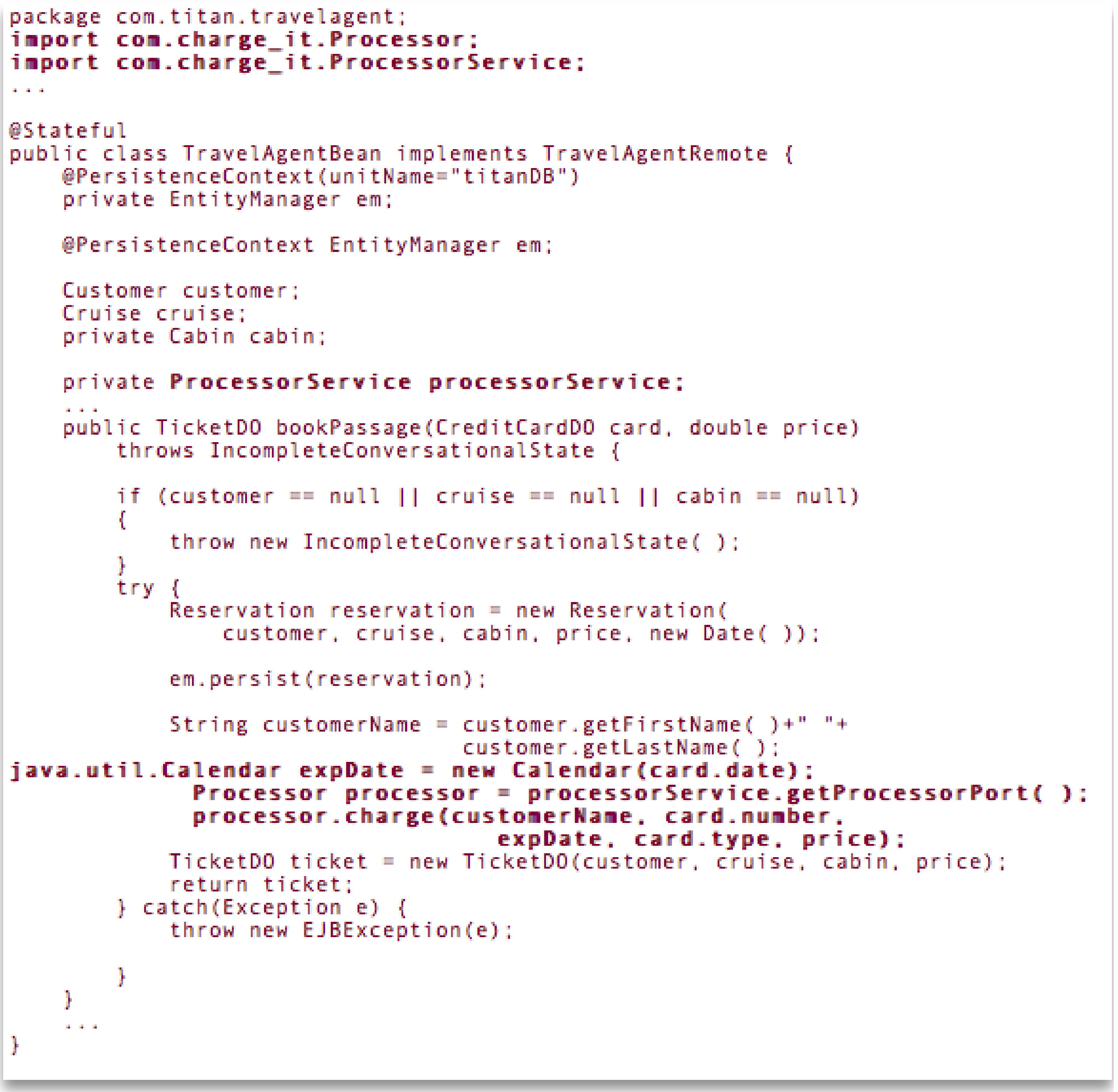
```
public interface Processor extends java.rmi.Remote {
  public int charge(String name, String number,
  java.util.Calendar expDate, String cardtype, float amount)
  throws java.rmi.RemoteException;
}
```

- O endereço http://www.charge-it.com/ProcessorService especifica o ponto onde o serviço troca mensagens SOAP
- O compilador cria o ponto de contacto da troca de mensagens



```
package com.charge_it;

public interface ProcessorService extends javax.xml.rpc.Service {
   public com.charge_it.Processor getProcessorPort( )
      throws javax.xml.rpc.ServiceException;
   public java.lang.String getProcessorPortAddress( );
   public com.charge_it.Processor getProcessorPort(java.net.URL portAddress)
      throws javax.xml.rpc.ServiceException;
}
```

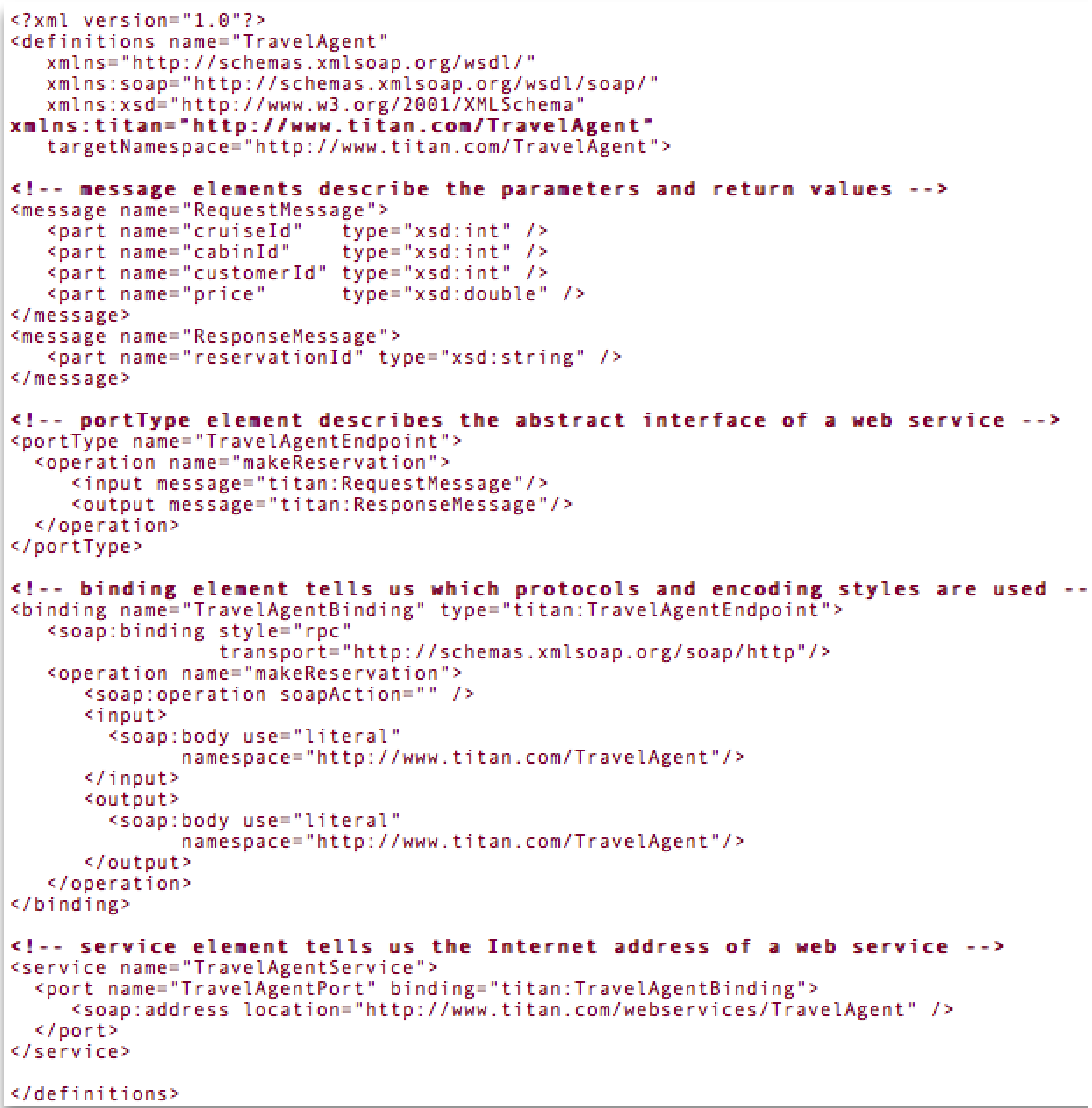
- Invocar o serviço numa aplicação



```
package com.titan.travelagent;
import com.charge_it.Processor;
import com.charge_it.ProcessorService;
...

@Stateful
public class TravelAgentBean implements TravelAgentRemote {
    @PersistenceContext(unitName="titanDB")
    private EntityManager em;

    @PersistenceContext EntityManager em;

    Customer customer;
    Cruise cruise;
    private Cabin cabin;

    private ProcessorService processorService;
    ...
    public TicketDO bookPassage(CreditCardDO card, double price)
        throws IncompleteConversationalState {

        if (customer == null || cruise == null || cabin == null)
        {
            throw new IncompleteConversationalState( );
        }
        try {
            Reservation reservation = new Reservation(
                customer, cruise, cabin, price, new Date( ));

            em.persist(reservation);

            String customerName = customer.getFirstName( )+" "+
                                  customer.getLastName( );
java.util.Calendar expDate = new Calendar(card.date);
             Processor processor = processorService.getProcessorPort( );
             processor.charge(customerName, card.number,
                                    expDate, card.type, price);
            TicketDO ticket = new TicketDO(customer, cruise, cabin, price);
            return ticket;
        } catch(Exception e) {
            throw new EJBException(e);

        }
    }
    ...
}
```

- Definir um Web Service

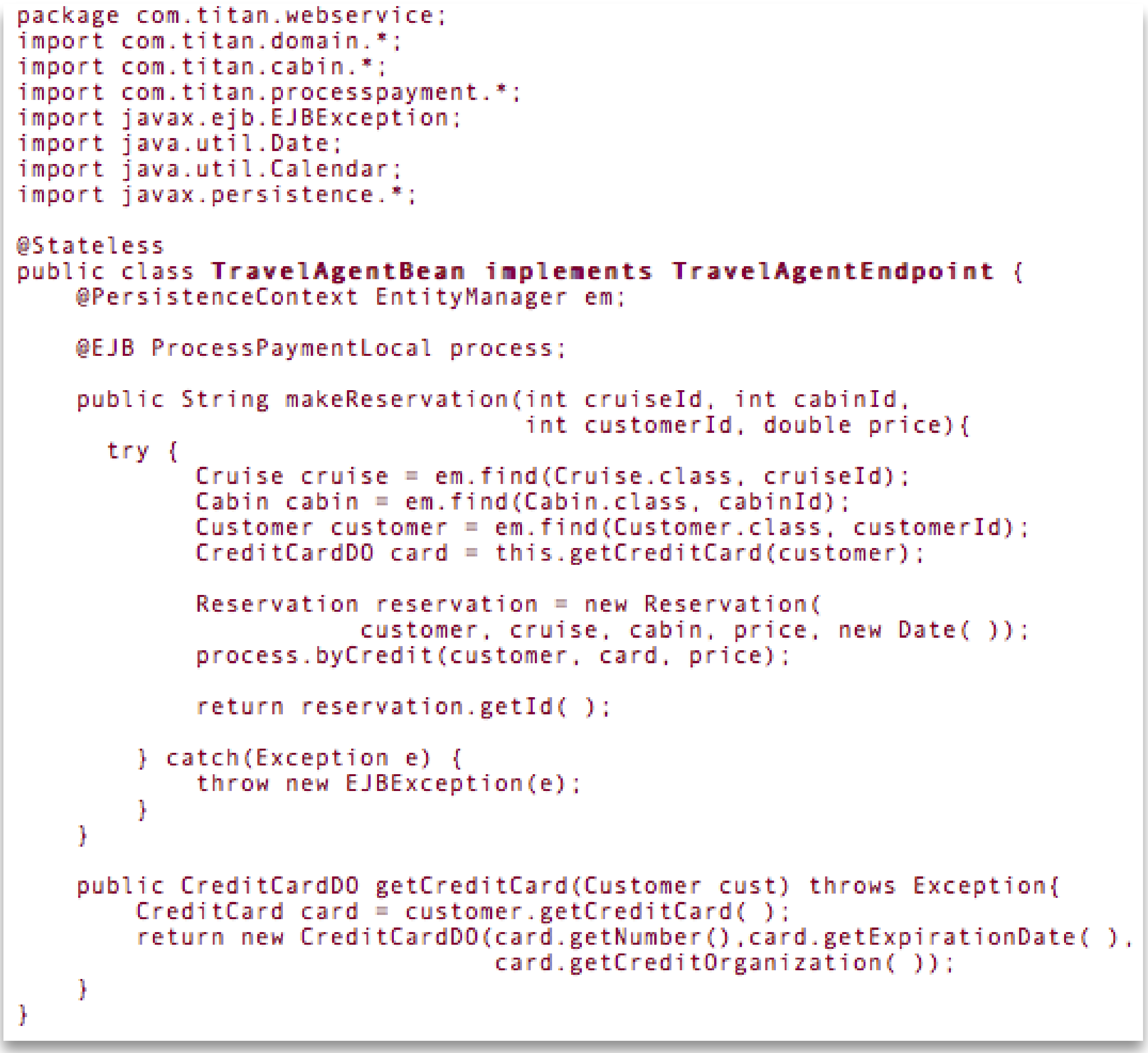
- Serviço para efectuar uma reserva



```
<?xml version="1.0"?>
<definitions name="TravelAgent"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:titan="http://www.titan.com/TravelAgent"
   targetNamespace="http://www.titan.com/TravelAgent">

<!-- message elements describe the parameters and return values -->
<message name="RequestMessage">
   <part name="cruiseId"   type="xsd:int" />
   <part name="cabinId"    type="xsd:int" />
   <part name="customerId" type="xsd:int" />
   <part name="price"      type="xsd:double" />
</message>
<message name="ResponseMessage">
   <part name="reservationId" type="xsd:string" />
</message>

<!-- portType element describes the abstract interface of a web service -->
<portType name="TravelAgentEndpoint">
  <operation name="makeReservation">
     <input message="titan:RequestMessage"/>
     <output message="titan:ResponseMessage"/>
  </operation>
</portType>

<!-- binding element tells us which protocols and encoding styles are used --
<binding name="TravelAgentBinding" type="titan:TravelAgentEndpoint">
   <soap:binding style="rpc"
                 transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="makeReservation">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"
              namespace="http://www.titan.com/TravelAgent"/>
      </input>
      <output>
        <soap:body use="literal"
              namespace="http://www.titan.com/TravelAgent"/>
      </output>
   </operation>
</binding>

<!-- service element tells us the Internet address of a web service -->
<service name="TravelAgentService">
  <port name="TravelAgentPort" binding="titan:TravelAgentBinding">
     <soap:address location="http://www.titan.com/webservices/TravelAgent" />
  </port>
</service>

</definitions>
```

- O Bean que fornece o serviço

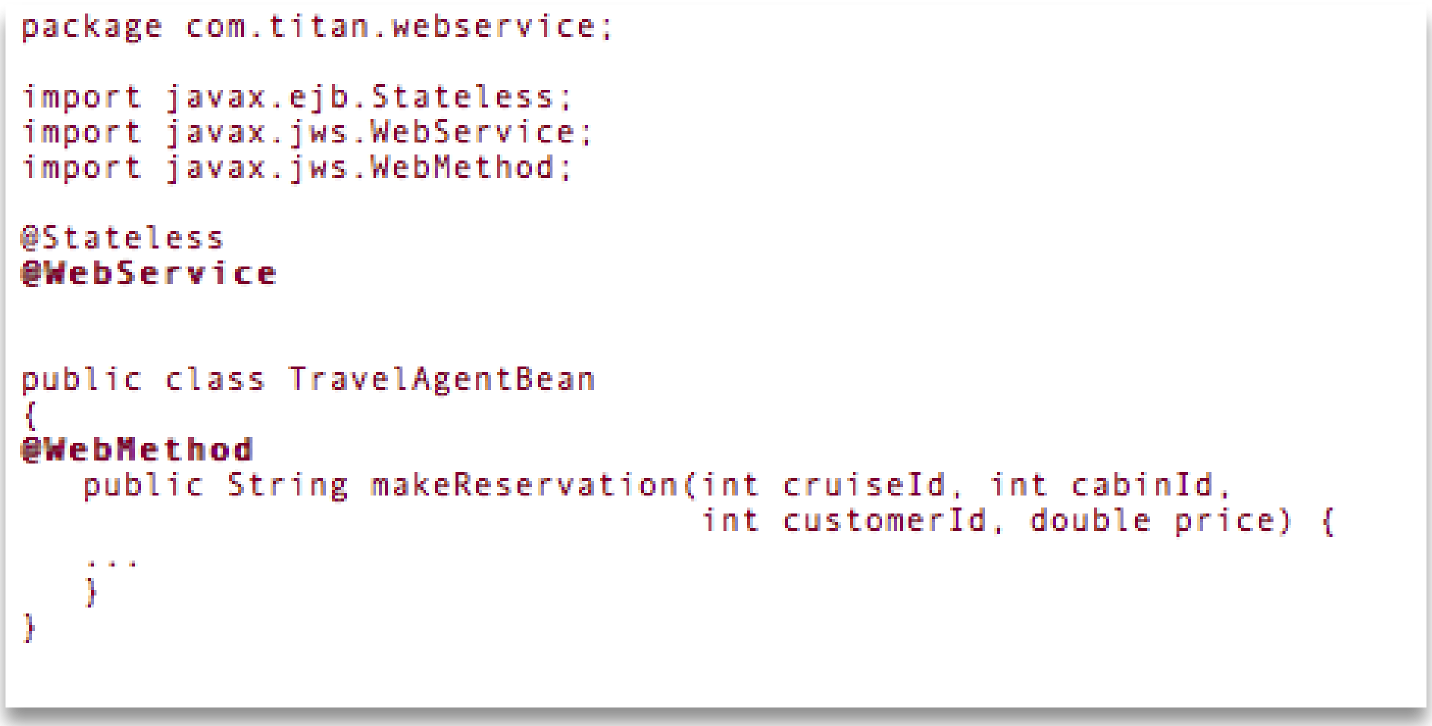
- É stateless!



```
package com.titan.webservice;
import com.titan.domain.*;
import com.titan.cabin.*;
import com.titan.processpayment.*;
import javax.ejb.EJBException;
import java.util.Date;
import java.util.Calendar;
import javax.persistence.*;

@Stateless
public class TravelAgentBean implements TravelAgentEndpoint {
    @PersistenceContext EntityManager em;

    @EJB ProcessPaymentLocal process;

    public String makeReservation(int cruiseId, int cabinId,
                                  int customerId, double price){
      try {
            Cruise cruise = em.find(Cruise.class, cruiseId);
            Cabin cabin = em.find(Cabin.class, cabinId);
            Customer customer = em.find(Customer.class, customerId);
            CreditCardDO card = this.getCreditCard(customer);

            Reservation reservation = new Reservation(
                       customer, cruise, cabin, price, new Date( ));
            process.byCredit(customer, card, price);

            return reservation.getId( );

        } catch(Exception e) {
            throw new EJBException(e);
        }
    }

    public CreditCardDO getCreditCard(Customer cust) throws Exception{
        CreditCard card = customer.getCreditCard( );
        return new CreditCardDO(card.getNumber(),card.getExpirationDate( ),
                                card.getCreditOrganization( ));
    }
}
```

EJB3.0: JAX-WS

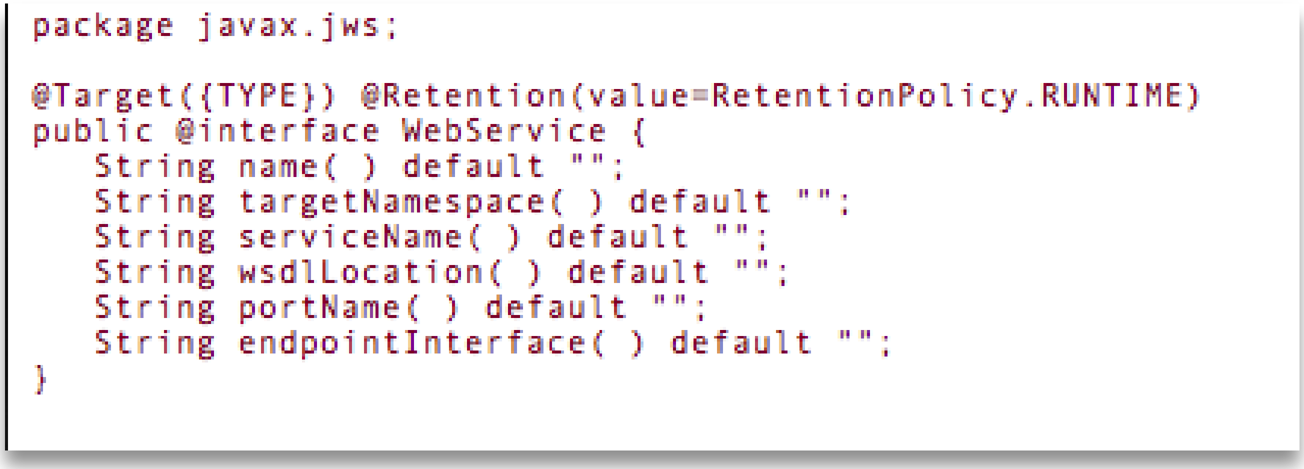
- Utilização de anotações para a definição dos elementos de um Web Service



```java
package com.titan.webservice;

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

@Stateless
@WebService


public class TravelAgentBean
{
@WebMethod
   public String makeReservation(int cruiseId, int cabinId,
                                 int customerId, double price) {
   ...
   }
}
```

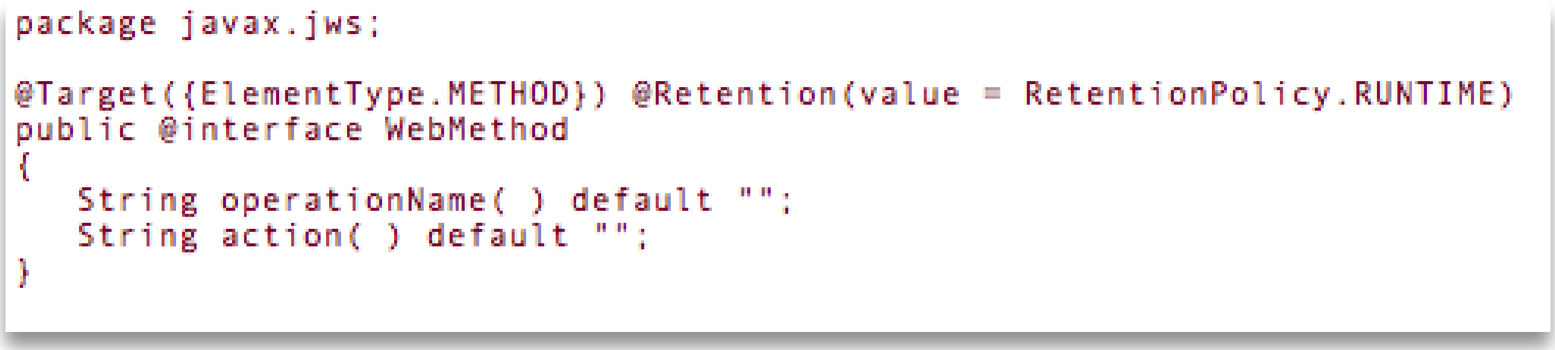
EJB3.0: JAX-WS

- A anotação @WebService é definida como:



```
package javax.jws;

@Target({TYPE}) @Retention(value=RetentionPolicy.RUNTIME)
public @interface WebService {
   String name( ) default "";
   String targetNamespace( ) default "";
   String serviceName( ) default "";
   String wsdlLocation( ) default "";
   String portName( ) default "";
   String endpointInterface( ) default "";
}
```

- A anotação @WebMethod é definida como:

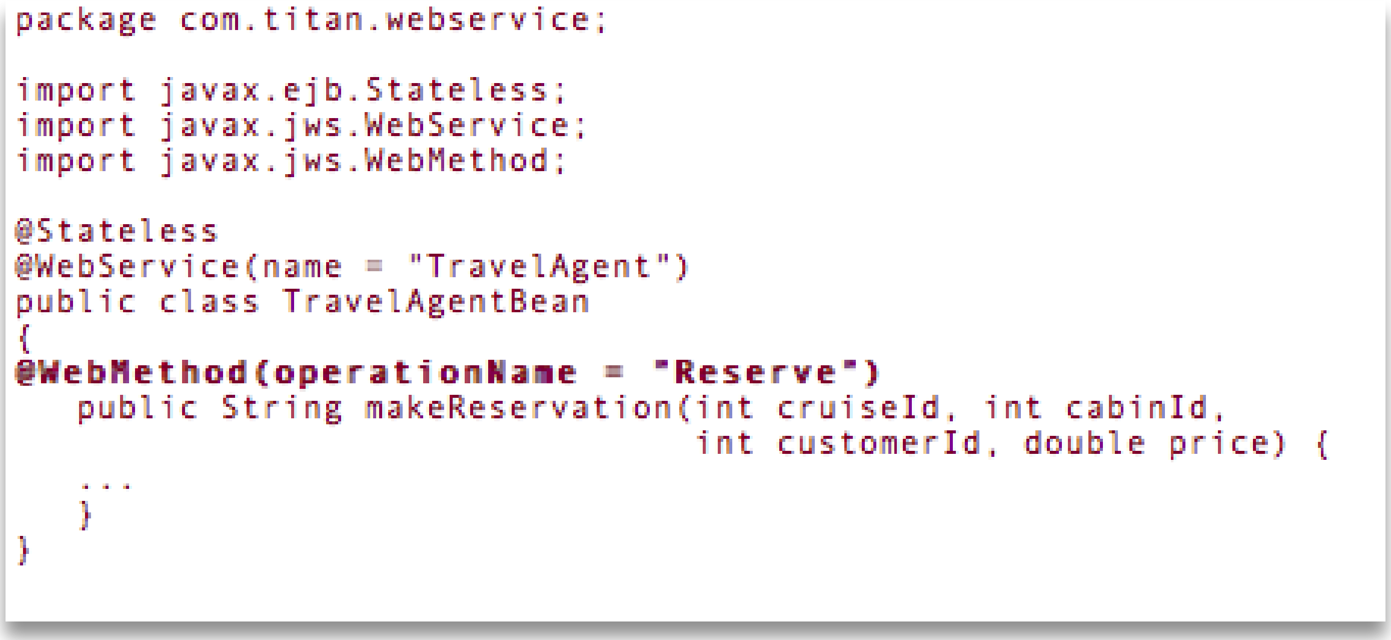


```
package javax.jws;

@Target({ElementType.METHOD}) @Retention(value = RetentionPolicy.RUNTIME)
public @interface WebMethod
{
   String operationName( ) default "";
   String action( ) default "";
}
```

- coloca-se nos métodos a ser expostos. Por *default* são todos!

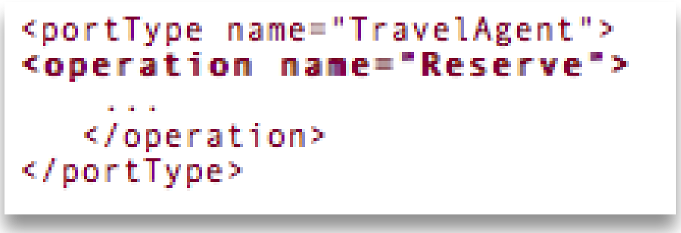
EJB3.0: JAX-WS

- Exemplo:



```java
package com.titan.webservice;

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

@Stateless
@WebService(name = "TravelAgent")
public class TravelAgentBean
{
@WebMethod(operationName = "Reserve")
   public String makeReservation(int cruiseId, int cabinId,
                                 int customerId, double price) {
   ...
   }
}
```

- que corresponde ao WSDL



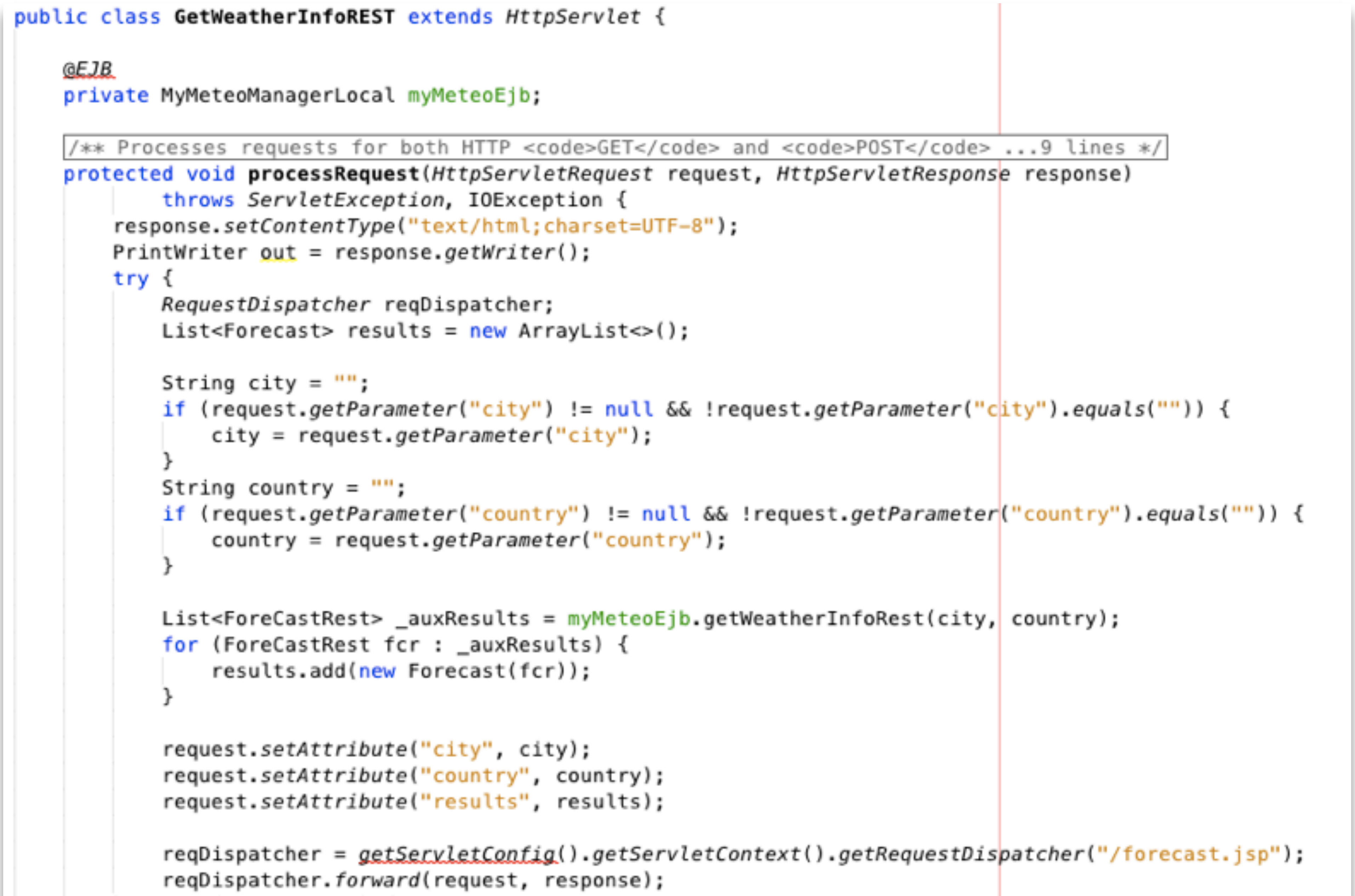
```xml
<portType name="TravelAgent">
<operation name="Reserve">
    ...
   </operation>
</portType>
```

Exemplo Clientes WSDL e REST

- Dois beans, um para WSDL e outro para Rest, para acesso ao serviço
- A servlet invoca o bean respectivo
- O resultado é enviado para um JSP

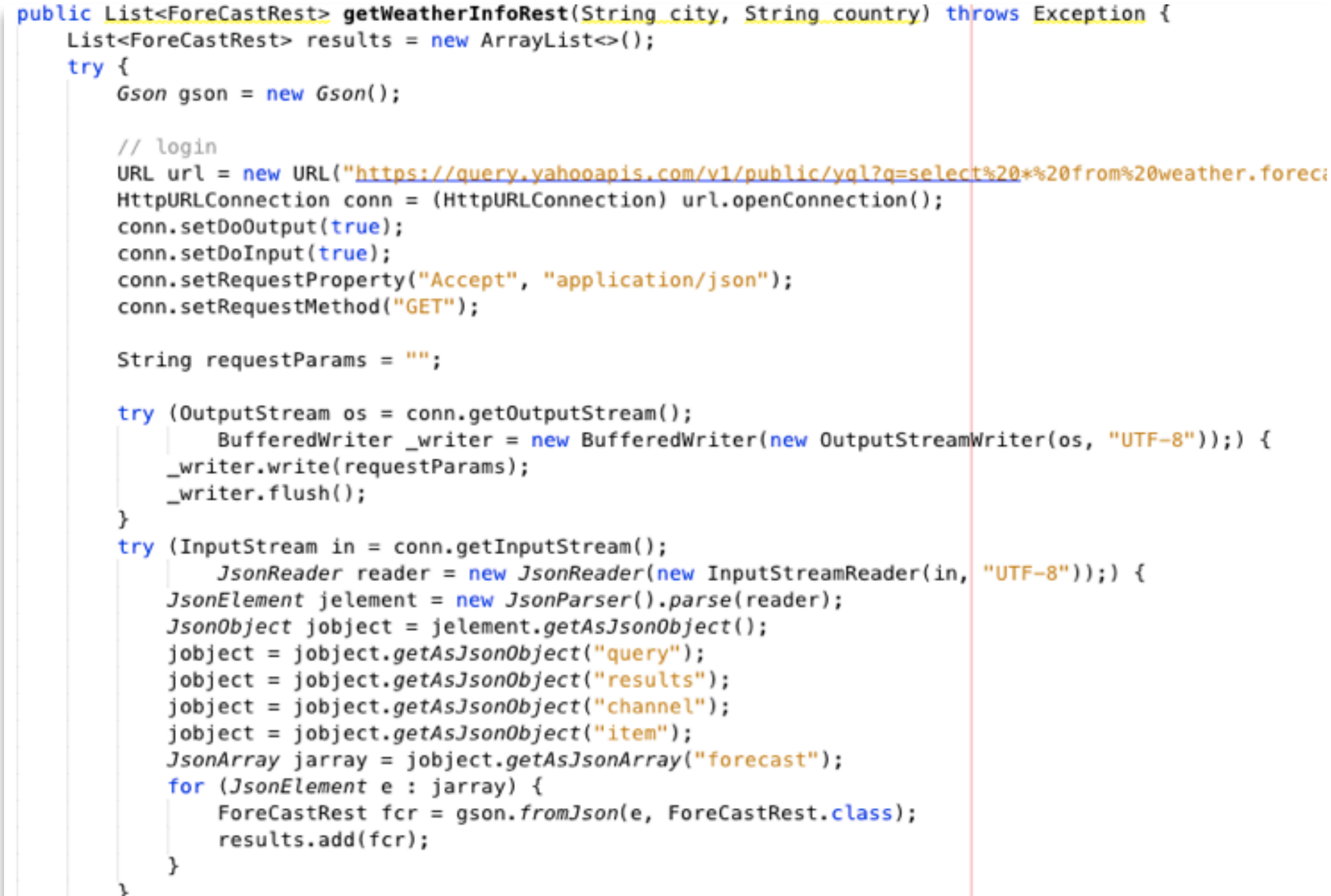
- Recursos
  - Documentação do serviço REST: https://developer.yahoo.com/weather/
  - WSDL do serviço SOAP: http://www.webservicex.com/globalweather.asmx?WSDL

Invocar um webservice Rest da Yahoo (meteo)



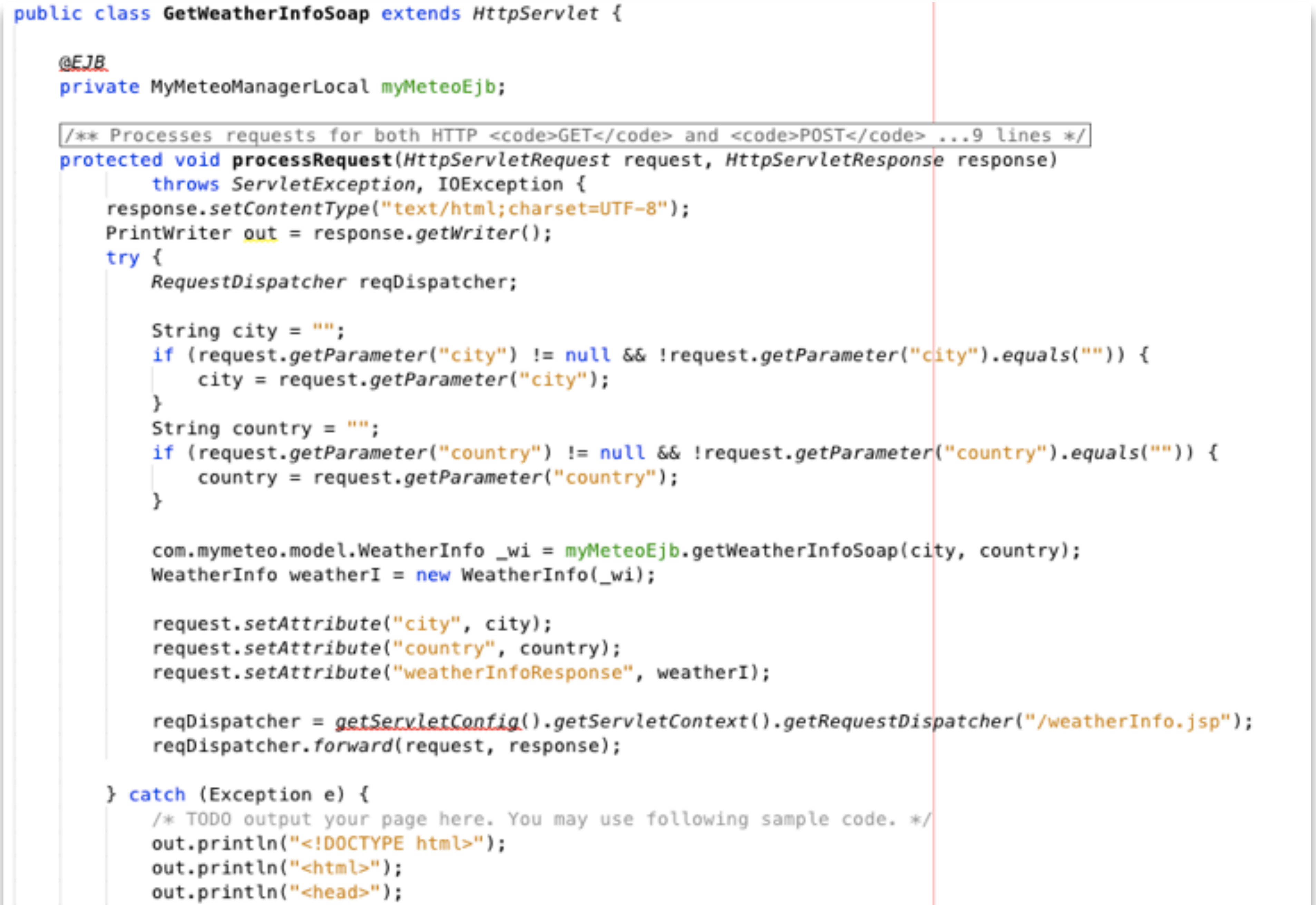
```java
public class GetWeatherInfoREST extends HttpServlet {

    @EJB
    private MyMeteoManagerLocal myMeteoEjb;

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            RequestDispatcher reqDispatcher;
            List<Forecast> results = new ArrayList<>();

            String city = "";
            if (request.getParameter("city") != null && !request.getParameter("city").equals("")) {
                city = request.getParameter("city");
            }
            String country = "";
            if (request.getParameter("country") != null && !request.getParameter("country").equals("")) {
                country = request.getParameter("country");
            }

            List<ForeCastRest> _auxResults = myMeteoEjb.getWeatherInfoRest(city, country);
            for (ForeCastRest fcr : _auxResults) {
                results.add(new Forecast(fcr));
            }

            request.setAttribute("city", city);
            request.setAttribute("country", country);
            request.setAttribute("results", results);

            reqDispatcher = getServletConfig().getServletContext().getRequestDispatcher("/forecast.jsp");
            reqDispatcher.forward(request, response);
```

Stateless Bean que faz o acesso à API REST



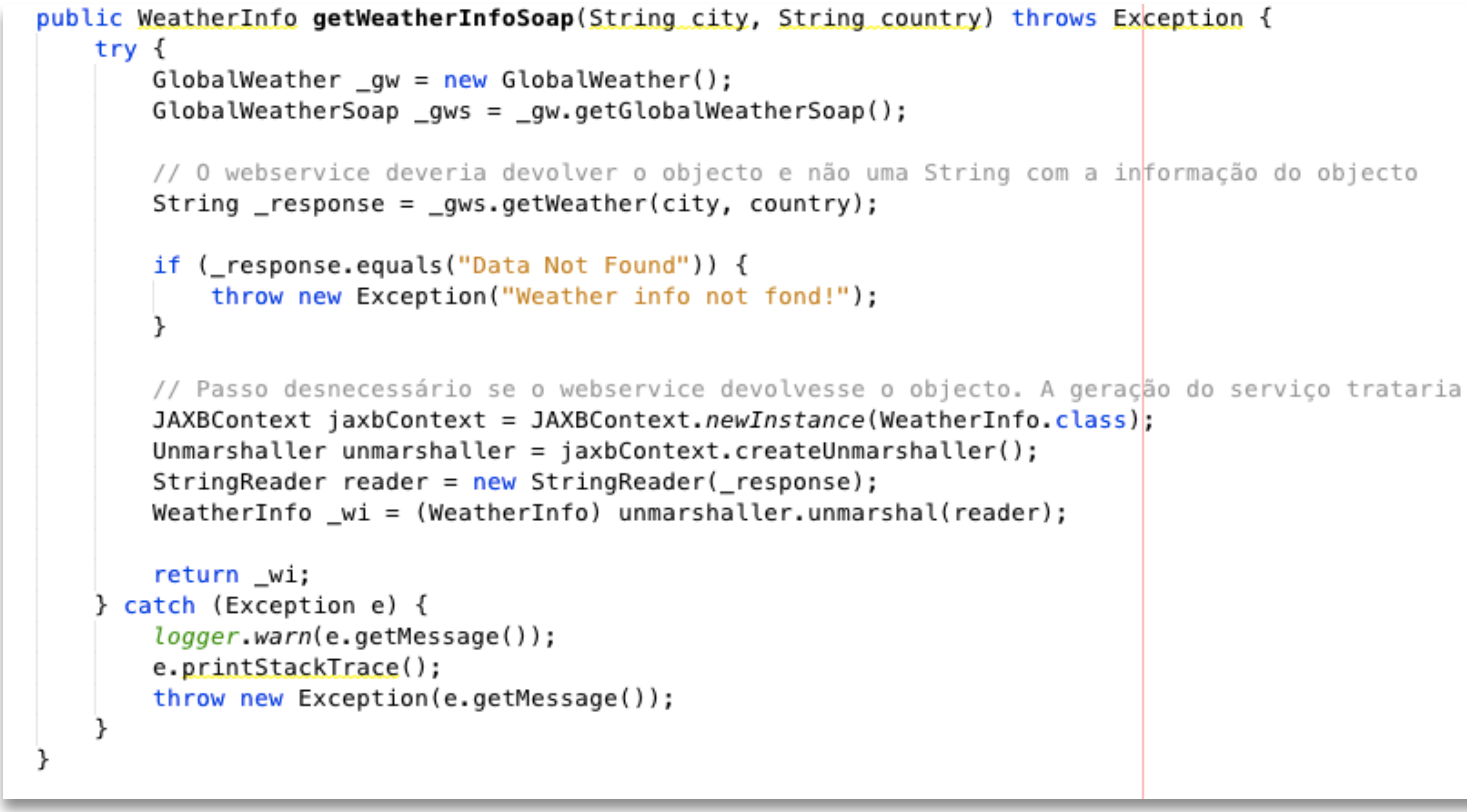
```java
public List<ForeCastRest> getWeatherInfoRest(String city, String country) throws Exception {
    List<ForeCastRest> results = new ArrayList<>();
    try {
        Gson gson = new Gson();

        // login
        URL url = new URL("https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20weather.foreca
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setDoOutput(true);
        conn.setDoInput(true);
        conn.setRequestProperty("Accept", "application/json");
        conn.setRequestMethod("GET");

        String requestParams = "";

        try (OutputStream os = conn.getOutputStream();
                BufferedWriter _writer = new BufferedWriter(new OutputStreamWriter(os, "UTF-8"));) {
            _writer.write(requestParams);
            _writer.flush();
        }
        try (InputStream in = conn.getInputStream();
                JsonReader reader = new JsonReader(new InputStreamReader(in, "UTF-8"));) {
            JsonElement jelement = new JsonParser().parse(reader);
            JsonObject jobject = jelement.getAsJsonObject();
            jobject = jobject.getAsJsonObject("query");
            jobject = jobject.getAsJsonObject("results");
            jobject = jobject.getAsJsonObject("channel");
            jobject = jobject.getAsJsonObject("item");
            JsonArray jarray = jobject.getAsJsonArray("forecast");
            for (JsonElement e : jarray) {
                ForeCastRest fcr = gson.fromJson(e, ForeCastRest.class);
                results.add(fcr);
            }
        }
```

Invocar um serviço WSDL



```java
public class GetWeatherInfoSoap extends HttpServlet {

    @EJB
    private MyMeteoManagerLocal myMeteoEjb;

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            RequestDispatcher reqDispatcher;

            String city = "";
            if (request.getParameter("city") != null && !request.getParameter("city").equals("")) {
                city = request.getParameter("city");
            }
            String country = "";
            if (request.getParameter("country") != null && !request.getParameter("country").equals("")) {
                country = request.getParameter("country");
            }

            com.mymeteo.model.WeatherInfo _wi = myMeteoEjb.getWeatherInfoSoap(city, country);
            WeatherInfo weatherI = new WeatherInfo(_wi);

            request.setAttribute("city", city);
            request.setAttribute("country", country);
            request.setAttribute("weatherInfoResponse", weatherI);

            reqDispatcher = getServletConfig().getServletContext().getRequestDispatcher("/weatherInfo.jsp");
            reqDispatcher.forward(request, response);

        } catch (Exception e) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
```

Fazer o acesso à API WSDL (SOAP)



```
public WeatherInfo getWeatherInfoSoap(String city, String country) throws Exception {
    try {
        GlobalWeather _gw = new GlobalWeather();
        GlobalWeatherSoap _gws = _gw.getGlobalWeatherSoap();

        // O webservice deveria devolver o objecto e não uma String com a informação do objecto
        String _response = _gws.getWeather(city, country);

        if (_response.equals("Data Not Found")) {
            throw new Exception("Weather info not fond!");
        }

        // Passo desnecessário se o webservice devolvesse o objecto. A geração do serviço trataria
        JAXBContext jaxbContext = JAXBContext.newInstance(WeatherInfo.class);
        Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
        StringReader reader = new StringReader(_response);
        WeatherInfo _wi = (WeatherInfo) unmarshaller.unmarshal(reader);

        return _wi;
    } catch (Exception e) {
        logger.warn(e.getMessage());
        e.printStackTrace();
        throw new Exception(e.getMessage());
    }
}
```