ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

# «САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет программной инженерии и компьютерной техники

Кафедра вычислительной техники

**Отчёт по лабораторной работе №7**

**ПО КУРСУ**

**«Технологии веб-сервисов»**

**Регистрация и поиск сервиса в реестре jUDDI**

# Выполнил: студент группы P41142

Белоголов М.А.

Проверил: канд. техн. Наук,доцент Дергачев А. М.

Санкт-Петербург

# 2020

**Задание:**

Требуется разработать приложение, осуществляющее регистрацию сервиса в реестре jUDDI, а также поиск сервиса в реестре и обращение к нему. Рекомендуется реализовать консольное приложение, которое обрабатывает 2 команды. Итог работы первой команды – регистрация сервиса в реестре; вторая команда должна осуществлять поиск сервиса, а также обращение к нему.

**Выполнение работы:**

В файле *App.java*, код которого представлен в листинге 7.1 происходит создание *client* из конфигурационного файла *uddi.xml*, который представлен в листинге 7.4. Затем создается *clerk* и экземпляр класса *Juddi*. После регистрируется веб-сервис с заданным именем и *wsdl*, выполняется поиск по полученному при регистрации ключу сервиса, выполняется поиск по имени сервиса и вызов сервиса по точке доступа, полученной в результате поиска. Файл *ServiceInformation.java*, исходный код которого представлен в листинге 7.2, – POJO, представляет из себя сущность сервиса, содержащего ключ, имя, описание, и точку доступа. Файл *Juddi.java*, исходный код которого представлен в листинге 7.3, содержит следующие методы:

• метод *publish* регистрирует сервис с указанным именем, описанием и wsdl-описанием, возвращает ключ опубликованного сервиса; • метод *findServiceByKey* ищет сервис по ключу и возвращает точку доступа найденного сервиса; • метод *findServiceByName* ищет сервисы по имени и возвращает

*List<ServiceInformation>* – список сервисов;

# • метод *deleteService* удаляет зарегистрированный сервис по ключу; • метод *deleteService* удаляет зарегистрированный сервис по ключу; • метод *callService* отправляет запрос по указанному адресу и сохраняет результаты в файл;

• метод *displayServiceDetails* выводит на экран информацию о сервисе. Также используются методы *ListToString*, *ListToDescString, CatBagToString,**KeyedReferenceToString,**PrintBindingTemplates*, которые выводят информацию об отдельных элементах сервиса.

# Листинг 7.1 – Файл App.java **package** com.maxart;

**import** org.apache.commons.configuration.ConfigurationException; **import** org.apache.juddi.v3.client.config.UDDIClerk; **import** org.apache.juddi.v3.client.config.UDDIClient; **import** org.apache.juddi.v3.client.transport.TransportException; **import** org.xml.sax.SAXException;

**import** javax.xml.parsers.ParserConfigurationException; **import** javax.xml.transform.TransformerException; **import** java.io.IOException; **import** java.net.URL; **import** java.util.List;

**public class** App {

**public static void** main(String args[]) {

String wsdl = **"http://localhost:8081/PictureService?wsdl"**;

String businessKey = **"uddi:localhost:0174084d-cbf1-44f2-af311b3c3ca7087d"**;

String serviceName = **"PictureWebService"**;

String serviceDescription = **"Simple service for getting information about pictures"**;

String serviceKey = **""**;

UDDIClient client = **null**; **try** {

System.***out***.println(**"Create UDDI client using uddi.xml"**); client = **new** UDDIClient(**"META-INF/uddi.xml"**); } **catch** (ConfigurationException e) {

e.printStackTrace();

}

**if** (client != **null**) {

System.***out***.println(**"Get UDDI clerk using clerkName"**); UDDIClerk clerk = client.getClerk(**"maxart"**); **try** {

Juddi juddi = **new** Juddi(client, clerk, businessKey); System.***out***.println(**"Publish service "** + serviceName + **" with access point "** + wsdl);

serviceKey = juddi.publish(serviceName, serviceDescription, wsdl);

System.***out***.println(**"Publishing service key: "** + serviceKey); System.***out***.println();

System.***out***.println(**"Find service by service key = "** + serviceKey);

String accessPoint = juddi.findServiceByKey(serviceKey);

System.***out***.println(**"Access point for results found: "** + accessPoint);

System.***out***.println();

System.***out***.println(**"Find service by service name = "** + serviceName);

List<ServiceInformation> si = juddi.findServiceByName(serviceName);

System.***out***.println(**"Results found"**);

*displayServices*(si); System.***out***.println();

System.***out***.println(**"Call service by wsdl = "** + accessPoint); juddi.callService(**new** URL(accessPoint), **"wsdl.xml"**);

System.***out***.println(**"Results saved into wsdl.xml"**);

System.***out***.println();

juddi.deleteService(serviceKey);

} **catch** (ConfigurationException | TransportException |

ParserConfigurationException | SAXException | TransformerException |

IOException e) {

e.printStackTrace();

}

clerk.discardAuthToken();

}

}

**private static void** displayServices(List<ServiceInformation> si) { **for** (ServiceInformation item : si) {

System.***out***.println(**"Service Id: "** + item.getId());

System.***out***.println(**"Service Name: "** + item.getName());

System.***out***.println(**"Service Description: "** + item.getDescription());

System.***out***.println(**"Service Access point: "** + item.getAccessPoint());

System.***out***.println();

}

}

}

# Листинг 7.2 – Файл ServiceInformation.java **package** com.maxart;

**public class** ServiceInformation {

**private** String **id**; **private** String **name**; **private** String **description**; **private** String **accessPoint**;

**public** ServiceInformation() {

}

**public** String getId() { **return id**;

}

**public void** setId(String id) { **this**.**id** = id;

}

**public** String getName() { **return name**;

}

**public void** setName(String name) { **this**.**name** = name;

}

**public** String getDescription() { **return description**;

}

**public void** setDescription(String description) { **this**.**description** = description;

}

**public** String getAccessPoint() { **return accessPoint**;

}

**public void** setAccessPoint(String accessPoint) { **this**.**accessPoint** = accessPoint;

}

}

# Листинг 7.3 – Файл Juddi.java **package** com.maxart;

**import** org.apache.commons.configuration.ConfigurationException; **import** org.apache.juddi.api\_v3.AccessPointType; **import** org.apache.juddi.v3.client.UDDIConstants; **import** org.apache.juddi.v3.client.config.UDDIClerk; **import** org.apache.juddi.v3.client.config.UDDIClient; **import** org.apache.juddi.v3.client.transport.Transport; **import** org.apache.juddi.v3.client.transport.TransportException; **import** org.uddi.api\_v3.\*;

**import** org.uddi.v3\_service.UDDIInquiryPortType; **import** org.uddi.v3\_service.UDDIPublicationPortType; **import** org.uddi.v3\_service.UDDISecurityPortType; **import** org.w3c.dom.Document; **import** org.xml.sax.SAXException;

**import** javax.xml.parsers.DocumentBuilder; **import** javax.xml.parsers.DocumentBuilderFactory; **import** javax.xml.parsers.ParserConfigurationException; **import** javax.xml.transform.Transformer; **import** javax.xml.transform.TransformerException; **import** javax.xml.transform.TransformerFactory; **import** javax.xml.transform.dom.DOMSource; **import** javax.xml.transform.stream.StreamResult; **import** java.io.File; **import** java.io.IOException; **import** java.net.URL; **import** java.net.URLConnection; **import** java.rmi.RemoteException; **import** java.util.ArrayList; **import** java.util.List;

**public class** Juddi {

**private** UDDIClerk **clerk**; **private** String **businessKey**; **private** UDDIInquiryPortType **inquiry**; Juddi(UDDIClient client, UDDIClerk clerk, String businessKey) **throws** ConfigurationException, TransportException { **this**.**clerk** = clerk;

Transport transport = client.getTransport(**"default"**); **inquiry** = transport.getUDDIInquiryService(); **this**.**businessKey** = businessKey;

}

**public** String publish(String serviceName, String description, String wsdl) {

BusinessService myService = **new** BusinessService(); myService.setBusinessKey(**businessKey**); Name myServName = **new** Name(); myServName.setValue(serviceName); myService.getName().add(myServName);

Description myDescription = **new** Description(); myDescription.setValue(description);

myService.getDescription().add(myDescription);

BindingTemplate myBindingTemplate = **new** BindingTemplate(); AccessPoint accessPoint = **new** AccessPoint();

accessPoint.setUseType(AccessPointType.***WSDL\_DEPLOYMENT***.toString()); accessPoint.setValue(wsdl);

myBindingTemplate.setAccessPoint(accessPoint);

BindingTemplates myBindingTemplates = **new** BindingTemplates();

myBindingTemplate = UDDIClient.*addSOAPtModels*(myBindingTemplate); myBindingTemplates.getBindingTemplate().add(myBindingTemplate);

myService.setBindingTemplates(myBindingTemplates);

BusinessService svc = **clerk**.register(myService); **if** (svc == **null**) {

System.***out***.println(**"Save failed!"**);

System.*exit*(1);

}

**return** svc.getServiceKey();

}

**public** String findServiceByKey(String serviceKey) **throws** RemoteException,

ConfigurationException, TransportException {

BusinessService businessService = **clerk**.getServiceDetail(serviceKey); **if** (businessService == **null**) { **return ""**;

}

**return** displayServiceDetails(businessService);

}

**public** List<ServiceInformation> findServiceByName(String serviceName) **throws** RemoteException, TransportException { FindService findService = **new** FindService(); findService.setAuthInfo(**clerk**.getAuthToken());

FindQualifiers findQualifiers = **new** FindQualifiers();

findQualifiers.getFindQualifier().add(UDDIConstants.***APPROXIMATE\_MATCH***);

findService.setFindQualifiers(findQualifiers); Name searchName = **new** Name();

searchName.setValue(**"%"** + serviceName + **"%"**); findService.getName().add(searchName);

ServiceList serviceList = **inquiry**.findService(findService); GetServiceDetail gsd = **new** GetServiceDetail(); **for** (**int** i = 0; i <

serviceList.getServiceInfos().getServiceInfo().size(); i++) {

gsd.getServiceKey().add(serviceList.getServiceInfos().getServiceInfo().get(i)

.getServiceKey());

}

List<ServiceInformation> services = **new** ArrayList<>(); ServiceDetail serviceDetail = **inquiry**.getServiceDetail(gsd); **if** (serviceDetail != **null**) {

**for** (**int** i = 0; i < serviceDetail.getBusinessService().size(); i++) {

ServiceInformation si = **new** ServiceInformation();

si.setId(serviceDetail.getBusinessService().get(i).getServiceKey());

si.setName(serviceDetail.getBusinessService().get(i).getName().get(0).getValu e());

si.setDescription(serviceDetail.getBusinessService().get(i).getDescription(). get(0).getValue());

si.setAccessPoint(displayServiceDetails(serviceDetail.getBusinessService().ge t(i)));

services.add(si); System.***out***.println();

}

}

**return** services;

}

**public void** deleteService(String serviceKey) { **clerk**.unRegisterService(serviceKey);

}

**public void** callService(URL url, String filename) **throws** IOException,

SAXException, ParserConfigurationException, TransformerException {

URLConnection conn = url.openConnection();

DocumentBuilderFactory factory =

DocumentBuilderFactory.*newInstance*();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.parse(conn.getInputStream());

TransformerFactory tfactory = TransformerFactory.*newInstance*();

Transformer xform = tfactory.newTransformer();

File myOutput = **new** File(filename);

xform.transform(**new** DOMSource(doc), **new** StreamResult(myOutput));

}

**private** String displayServiceDetails(BusinessService businessService) {

System.***out***.println(**"Name: "** +

ListToString(businessService.getName()));

System.***out***.println(**"Desc: "** +

ListToDescString(businessService.getDescription()));

System.***out***.println(**"Key: "** + (businessService.getServiceKey()));

System.***out***.println(**"Cat bag: "** +

CatBagToString(businessService.getCategoryBag())); **if** (!businessService.getSignature().isEmpty()) { System.***out***.println(**"Item is digitally signed"**); } **else** {

System.***out***.println(**"Item is not digitally signed"**);

}

**return** PrintBindingTemplates(businessService.getBindingTemplates());

}

**private** String ListToString(List<Name> name) { StringBuilder sb = **new** StringBuilder(); **for** (Name aName : name) {

sb.append(aName.getValue()).append(**" "**);

}

**return** sb.toString();

}

**private** String ListToDescString(List<Description> name) {

StringBuilder sb = **new** StringBuilder(); **for** (Description aName : name) {

sb.append(aName.getValue()).append(**" "**);

}

**return** sb.toString();

}

**private** String CatBagToString(CategoryBag categoryBag) {

StringBuilder sb = **new** StringBuilder(); **if** (categoryBag == **null**) { **return "no data"**;

}

**for** (**int** i = 0; i < categoryBag.getKeyedReference().size(); i++) {

sb.append(KeyedReferenceToString(categoryBag.getKeyedReference().get(i)));

}

**for** (**int** i = 0; i < categoryBag.getKeyedReferenceGroup().size(); i++)

{

sb.append(**"Key Ref Grp: TModelKey="**); **for** (**int** k = 0; k <

categoryBag.getKeyedReferenceGroup().get(i).getKeyedReference().size(); k++) {

sb.append(KeyedReferenceToString(categoryBag.getKeyedReferenceGroup().get(i).

getKeyedReference().get(k)));

}

}

**return** sb.toString();

}

**private** String KeyedReferenceToString(KeyedReference item) { **return "Key Ref: Name="** + item.getKeyName() + **" Value="** + item.getKeyValue() + **" tModel="** + item.getTModelKey() +

System.*getProperty*(**"line.separator"**);

}

**private** String PrintBindingTemplates(BindingTemplates bindingTemplates) { String accessPoint = **""**; **if** (bindingTemplates == **null**) { **return** accessPoint;

}

**for** (**int** i = 0; i < bindingTemplates.getBindingTemplate().size(); i++) {

System.***out***.println(**"Binding Key: "** +

bindingTemplates.getBindingTemplate().get(i).getBindingKey());

**if** (bindingTemplates.getBindingTemplate().get(i).getAccessPoint()

!= **null**) {

accessPoint =

bindingTemplates.getBindingTemplate().get(i).getAccessPoint().getValue();

System.***out***.println(**"Access Point: "** + accessPoint + **" type "**

+

bindingTemplates.getBindingTemplate().get(i).getAccessPoint().getUseType()); **if**

(bindingTemplates.getBindingTemplate().get(i).getAccessPoint().getUseType()

!= **null**) {

**if**

(bindingTemplates.getBindingTemplate().get(i).getAccessPoint().getUseType().e

qualsIgnoreCase(AccessPointType.***END\_POINT***.toString())) {

System.***out***.println(**"Use this access point value as an invocation endpoint."**);

} **if**

(bindingTemplates.getBindingTemplate().get(i).getAccessPoint().getUseType().e

qualsIgnoreCase(AccessPointType.***BINDING\_TEMPLATE***.toString())) {

System.***out***.println(**"Use this access point value as a reference to another binding template."**);

} **if**

(bindingTemplates.getBindingTemplate().get(i).getAccessPoint().getUseType().e

qualsIgnoreCase(AccessPointType.***WSDL\_DEPLOYMENT***.toString())) {

System.***out***.println(**"Use this access point value as a URL to a WSDL document, which presumably will have a real access point defined."**);

} **if**

(bindingTemplates.getBindingTemplate().get(i).getAccessPoint().getUseType().e

qualsIgnoreCase(AccessPointType.***HOSTING\_REDIRECTOR***.toString())) {

System.***out***.println(**"Use this access point value as an**

**Inquiry URL of another UDDI registry, look up the same binding template there (usage varies)."**);

}

}

}

}

**return** accessPoint;

}

}

# Листинг 7.4 – Файл uddi.xml

*<?***xml version="1.0" encoding="ISO-8859-1"** *?>* <**uddi xmlns="urn:juddi-apache-org:v3\_client" xsi:schemaLocation="classpath:/xsd/uddi-client.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"** >

<**reloadDelay**>5000</**reloadDelay**>

<**client name="example-client"**>

<**nodes**> <**node**>

*<!-- required 'default' node -->*

</**name**>

|  |  |
| --- | --- |
| <**name**> | default |
|  | |

<**properties**>

<**property name="serverName" value="localhost"**/>

<**property name="serverPort" value="8080"**/>

<**property name="keyDomain" value="uddi.joepublisher.com"**/>

*<!-- for UDDI nodes that use HTTP u/p, using the following*

*<property name="basicAuthUsername" value="root" />*

*<property name="basicAuthPassword" value="password" />*

*<property name="basicAuthPasswordIsEncrypted" value="false"*

*/>*

*<property name="basicAuthPasswordCryptoProvider"*

*value="org.apache.juddi.v3.client.crypto.AES128Cryptor (an example)" />-->*

|  |
| --- |
| </**properties**> |
|  |

<**description**>Main jUDDI node</**description**>

*<!-- JAX-WS Transport -->*

|  |  |
| --- | --- |
| <**proxyTransport**> | |
|  |  |

org.apache.juddi.v3.client.transport.JAXWSTransport</**proxyTra nsport**>

<**custodyTransferUrl**>http://${serverName}:${serverPort}/juddiv3/services/custo dy-transfer</**custodyTransferUrl**>

|  |  |
| --- | --- |
| <**inquiryUrl**> | |
|  |  |

http://${serverName}:${serverPort}/juddiv3/services/inquiry</**inqu iryUrl**>

<**inquiryRESTUrl**>http://${serverName}:${serverPort}/juddiv3/services/inquiryRe st</**inquiryRESTUrl**>

<**publishUrl**>http://${serverName}:${serverPort}/juddiv3/services/publish</**publ ishUrl**>

<**securityUrl**>http://${serverName}:${serverPort}/juddiv3/services/security</**se curityUrl**>

<**subscriptionUrl**>http://${serverName}:${serverPort}/juddiv3/services/subscrip tion</**subscriptionUrl**>

<**subscriptionListenerUrl**>http://${serverName}:${serverPort}/juddiv3/services/ subscription-listener</**subscriptionListenerUrl**>

<**juddiApiUrl**>http://${serverName}:${serverPort}/juddiv3/services/juddiapi</**juddiApiUrl**>

</**node**>

</**nodes**>

<**clerks registerOnStartup="false"** >

<**clerk name="maxart" node="default" publisher="uddiadmin" password="da\_password1" isPasswordEncrypted="false"**

**cryptoProvider="org.apache.juddi.v3.client.cryptor.AES128Cryptor"**>

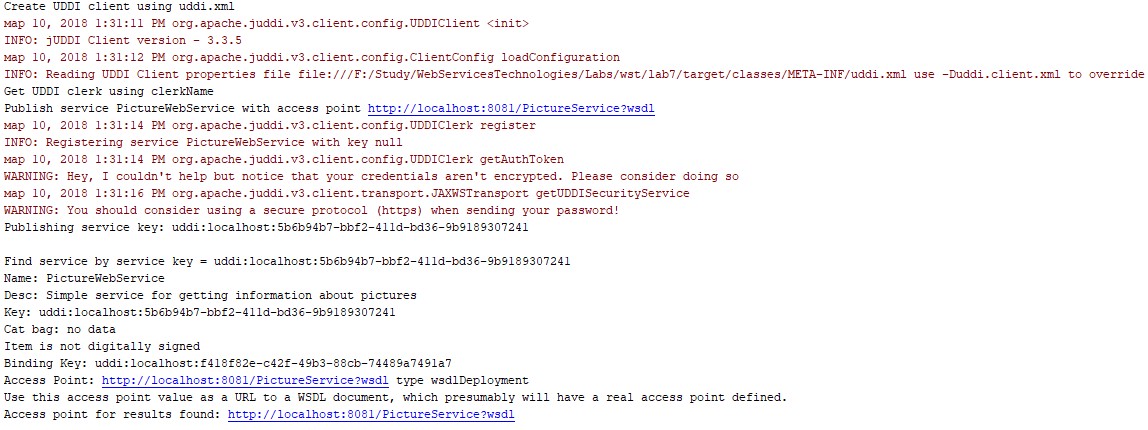
</**clerk**>

</**clerks**>

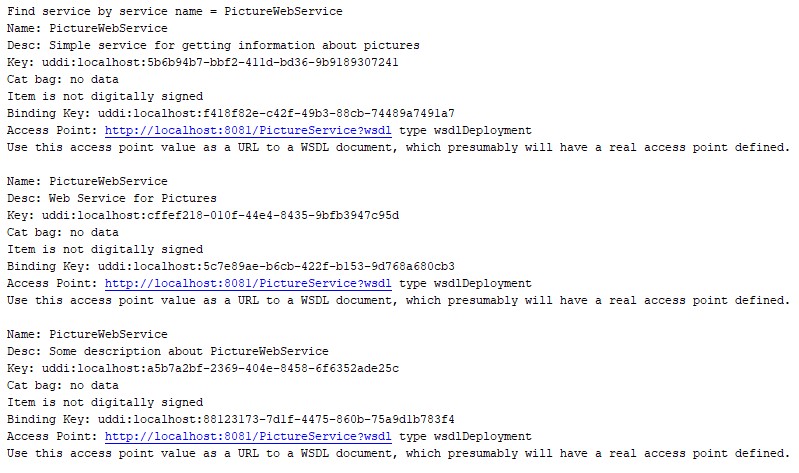
</**client**>

</**uddi**>

Результат выполнения программы приведен на рисунке 7.1 и в листинге 7.5.



# Рисунок 7.1, лист 1 – Результат выполнения клиентского консольного приложения



# Рисунок 7.1, лист 2 – Результат выполнения клиентского консольного приложения

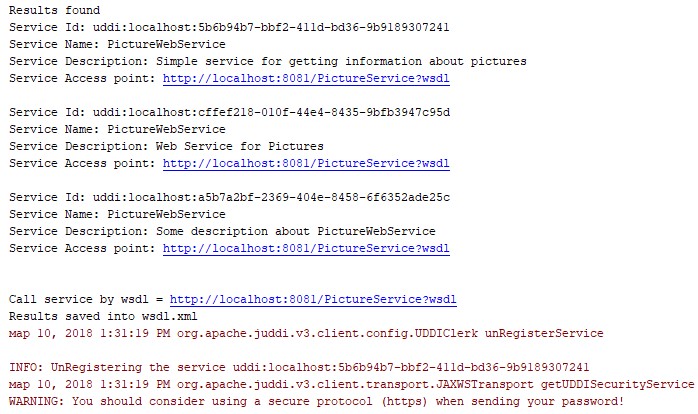


Рисунок 7.1, лист 3 – Результат выполнения клиентского консольного приложения

# Листинг 7.4 – Файл wsdl.xml, полученный в результате выполнения программы

*<?***xml version="1.0" encoding="UTF-8" standalone="no"***?><!-- Published by JAXWS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.5-hudson-*

*$BUILD\_NUMBER-. --><!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net.*

*RI's version is JAX-WS RI 2.1.5-hudson-$BUILD\_NUMBER-. -->* <**definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://service.maxart.com/"**

**xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401wss-wssecurity-utility-1.0.xsd"**

**xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="PictureService"**

**targetNamespace="http://service.maxart.com/"**>

<**types**>

<**xsd:schema**>

<**xsd:import namespace="http://service.maxart.com/"**

**schemaLocation="http://localhost:8081/PictureService?xsd=1"**/>

</**xsd:schema**>

<**xsd:schema**>

<**xsd:import namespace="http://service.maxart.com"**

**schemaLocation="http://localhost:8081/PictureService?xsd=2"**/>

</**xsd:schema**>

</**types**>

<**message name="getAllPictures"**>

<**part element="tns:getAllPictures" name="parameters"**/>

</**message**>

<**message name="getAllPicturesResponse"**>

<**part element="tns:getAllPicturesResponse" name="parameters"**/>

</**message**>

<**message name="findPictures"**>

<**part element="tns:findPictures" name="parameters"**/>

</**message**>

<**message name="findPicturesResponse"**>

<**part element="tns:findPicturesResponse" name="parameters"**/>

</**message**>

<**message name="IllegalQException"**>

<**part element="tns:IllegalQException" name="fault"**/>

</**message**>

<**message name="deletePicture"**>

<**part element="tns:deletePicture" name="parameters"**/>

</**message**>

<**message name="deletePictureResponse"**>

<**part element="tns:deletePictureResponse" name="parameters"**/>

</**message**>

<**message name="InvalidEntityException"**>

<**part element="tns:InvalidEntityException" name="fault"**/>

</**message**>

<**message name="IllegalIdException"**>

<**part element="tns:IllegalIdException" name="fault"**/>

</**message**>

<**message name="updatePicture"**>

<**part element="tns:updatePicture" name="parameters"**/>

</**message**>

<**message name="updatePictureResponse"**>

<**part element="tns:updatePictureResponse" name="parameters"**/>

</**message**>

<**message name="createPicture"**>

<**part element="tns:createPicture" name="parameters"**/>

</**message**>

<**message name="createPictureResponse"**>

<**part element="tns:createPictureResponse" name="parameters"**/>

</**message**>

<**message name="InsertingException"**>

<**part element="tns:InsertingException" name="fault"**/>

</**message**>

<**message name="InvalidCreatingParametersException"**>

<**part element="tns:InvalidCreatingParametersException" name="fault"**/>

</**message**>

<**portType name="PictureWebService"**>

<**operation name="getAllPictures"**>

<**input message="tns:getAllPictures"**/>

<**output message="tns:getAllPicturesResponse"**/>

</**operation**>

<**operation name="findPictures"**>

<**input message="tns:findPictures"**/>

<**output message="tns:findPicturesResponse"**/>

<**fault message="tns:IllegalQException" name="IllegalQException"**/>

</**operation**>

<**operation name="deletePicture"**>

<**input message="tns:deletePicture"**/>

<**output message="tns:deletePictureResponse"**/>

<**fault message="tns:InvalidEntityException" name="InvalidEntityException"**/>

<**fault message="tns:IllegalIdException" name="IllegalIdException"**/>

</**operation**>

<**operation name="updatePicture"**>

<**input message="tns:updatePicture"**/>

<**output message="tns:updatePictureResponse"**/>

<**fault message="tns:IllegalQException" name="IllegalQException"**/>

<**fault message="tns:IllegalIdException" name="IllegalIdException"**/>

<**fault message="tns:InvalidEntityException" name="InvalidEntityException"**/>

</**operation**>

<**operation name="createPicture"**>

<**input message="tns:createPicture"**/>

<**output message="tns:createPictureResponse"**/> <**fault message="tns:InsertingException" name="InsertingException"**/>

<**fault message="tns:InvalidCreatingParametersException" name="InvalidCreatingParametersException"**/>

</**operation**>

</**portType**>

<**binding name="PictureWebServicePortBinding" type="tns:PictureWebService"**>

<**soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"**/>

<**operation name="getAllPictures"**>

<**soap:operation soapAction=""**/>

<**input**>

<**soap:body use="literal"**/>

</**input**>

<**output**>

<**soap:body use="literal"**/>

</**output**>

</**operation**>

<**operation name="findPictures"**>

<**soap:operation soapAction=""**/>

<**input**>

<**soap:body use="literal"**/>

</**input**>

<**output**>

<**soap:body use="literal"**/>

</**output**>

<**fault name="IllegalQException"**>

<**soap:fault name="IllegalQException" use="literal"**/>

</**fault**>

</**operation**>

<**operation name="deletePicture"**>

<**soap:operation soapAction=""**/>

<**input**>

<**soap:body use="literal"**/>

</**input**>

<**output**>

<**soap:body use="literal"**/>

</**output**>

<**fault name="InvalidEntityException"**>

<**soap:fault name="InvalidEntityException" use="literal"**/>

</**fault**>

<**fault name="IllegalIdException"**>

<**soap:fault name="IllegalIdException" use="literal"**/>

</**fault**>

</**operation**>

<**operation name="updatePicture"**>

<**soap:operation soapAction=""**/>

<**input**>

<**soap:body use="literal"**/>

</**input**>

<**output**>

<**soap:body use="literal"**/>

</**output**>

<**fault name="IllegalQException"**>

<**soap:fault name="IllegalQException" use="literal"**/> </**fault**>

<**fault name="IllegalIdException"**>

<**soap:fault name="IllegalIdException" use="literal"**/>

</**fault**>

<**fault name="InvalidEntityException"**>

<**soap:fault name="InvalidEntityException" use="literal"**/>

</**fault**>

</**operation**>

<**operation name="createPicture"**>

<**soap:operation soapAction=""**/>

<**input**>

<**soap:body use="literal"**/>

</**input**>

<**output**>

<**soap:body use="literal"**/>

</**output**>

<**fault name="InsertingException"**>

<**soap:fault name="InsertingException" use="literal"**/>

</**fault**>

<**fault name="InvalidCreatingParametersException"**> <**soap:fault name="InvalidCreatingParametersException" use="literal"**/>

</**fault**>

</**operation**>

</**binding**>

<**service name="PictureService"**>

<**port binding="tns:PictureWebServicePortBinding" name="PictureWebServicePort"**>

<**soap:address location="http://localhost:8081/PictureService"**/>

</**port**> </**service**>

</**definitions**>

**Вывод:** в ходе выполнения работы было разработано консольное приложение, осуществляющее регистрацию сервиса в реестре jUDDI, а также поиск сервиса в реестре по ключу и имени и обращение к нему.