

Rozproszone systemy internetowe

Web serwisy

JAX-WS (SOAP web services)

Sources and help:

[Web services \(javaTpoint\)](#)

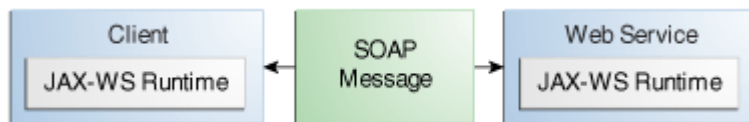
[Java Web Services - Book online](#)

[JAX-WS Tutorial by mkyong](#)

[Java EE Tutorial -Web services](#)

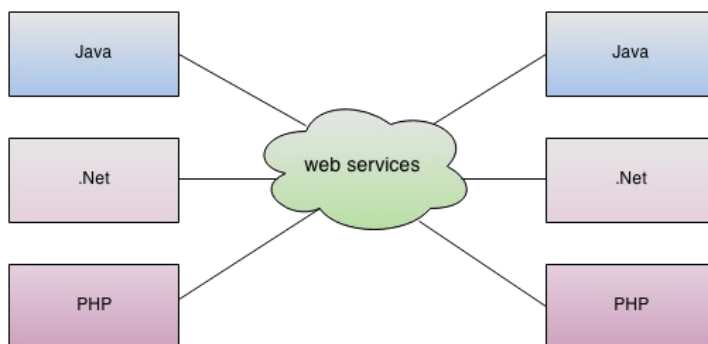
Java API for XML Web Services (JAX-WS), is a set of APIs for creating web services in **XML format (SOAP)**. JAX-WS provides many annotation to simplify the development and deployment for both web service clients and web service providers (endpoints).

Java API for XML Web Services is a technology for building **web services and clients** that **communicate** using **XML**.



In **JAX-WS**, a web service operation invocation is represented by an **XML-based protocol**, such as **SOAP**. The SOAP specification defines the envelope structure, encoding rules, and conventions for **representing web service invocations and responses**. These calls and responses are transmitted as **SOAP messages** (XML files) over **HTTP**.

Although SOAP messages are complex, the JAX-WS API hides this complexity from the application developer. On the **server side**, the developer **specifies the web service operations** by defining **methods** in an **interface written in the Java** programming language. The developer also codes one or more classes that implement those methods. **Client programs** are also easy to code. A client **creates a proxy** (a local object representing the service) and then **simply invokes methods** on the proxy. With JAX-WS, the developer **does not generate or parse SOAP messages**. It is the **JAX-WS runtime system** that **converts the API calls and responses** to and from **SOAP messages**.



Java, .net or PHP applications can communicate with other applications through web service over the network. For example, java application can interact with Java, .Net and PHP applications. So web service is a **language independent way of communication**.

Ćwiczenie 1. Tworzenie web serwisu

1.1 Tworzenie Web Service Endpoint Interface (*HelloWorld.java*)

```

package org.jg.rsi;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
import javax.jws.soap.SOAPBinding.Use;

//Service Endpoint Interface

@WebService

@SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL) //optional

public interface HelloWorld {

    @WebMethod

    String getHelloWorldAsString(String name);

}

```

1.2 Tworzenie Web Service Endpoint implementacji

```

package org.jg.rsi;

//Service Implementation

import javax.jws.WebService;

@WebService(endpointInterface = "org.jg.rsi.HelloWorld")

public class HelloWorldImpl implements HelloWorld {

    @Override

    public String getHelloWorldAsString(String name) {

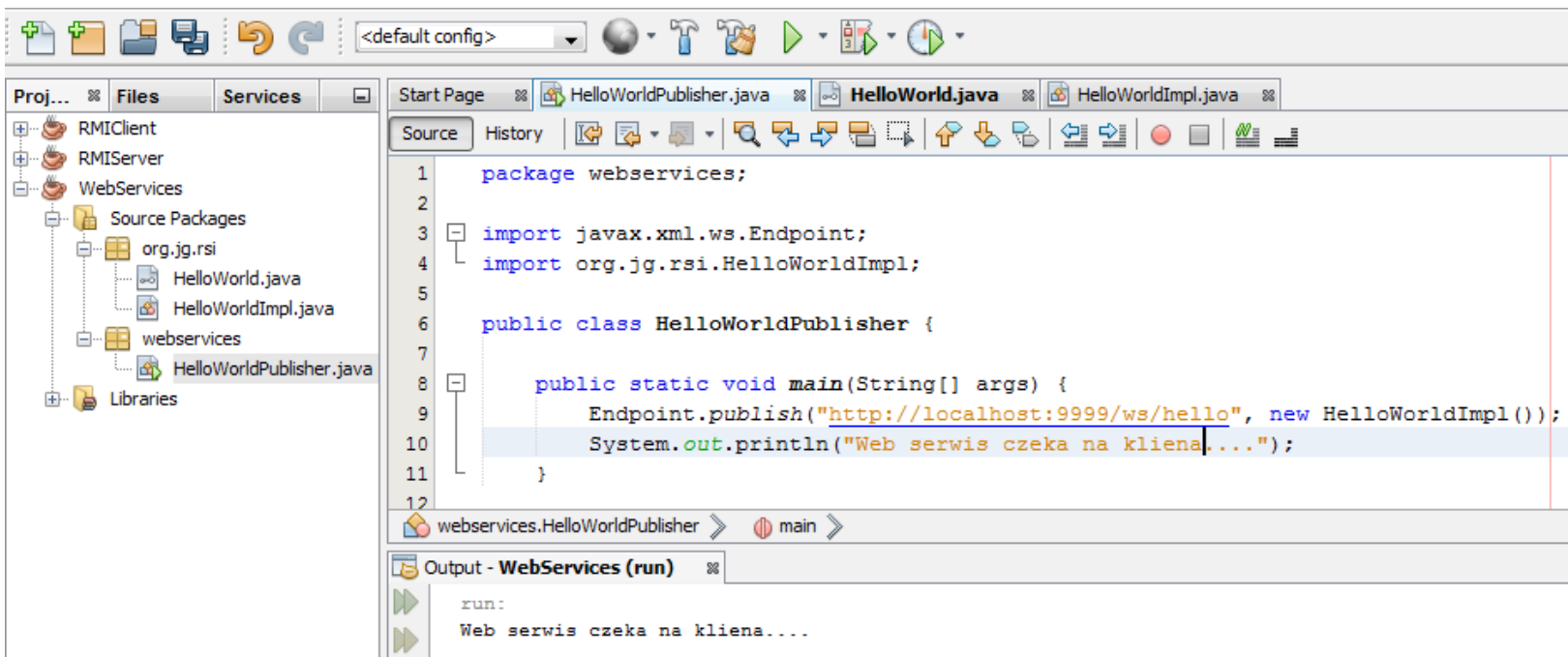
        return "Witaj świecie JAX-WS: " + name;

    }

}

```

1.3 Tworzenie Endpoint publikatora.



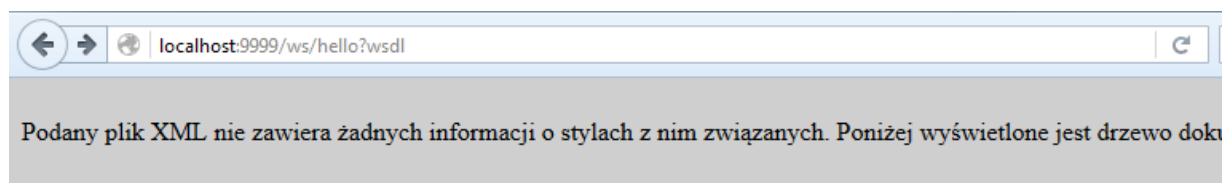
Publikowanie serwisu (bez użycia serwera Glassfish) do celów testowych (szybsze, prościej)

```
Endpoint.publish("http://localhost:9999/ws/hello", new HelloWorldImpl());
```

Ćwiczenie 2. Podgląd WSDL serwisu

W naszym przypadku serwis powinien być dostępny pod adresem
<http://localhost:9999/ws/hello>

Aby sprawdzić czy działa i pobrać WSDL należy użyć w przeglądarce
<http://localhost:9999/ws/hello?wsdl> (do nazwy serwisu dodano: ?wsdl)



```

- <!--
  Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1
-->
- <!--
  Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1
-->
- <definitions targetNamespace="http://rsi.jg.org/" name="HelloWorldImplService">
- <types>
- <xsd:schema>
- <xsd:import namespace="http://rsi.jg.org/" schemaLocation="http://localhost:9999/ws/hello?xsd=1"/>
- </xsd:schema>
- </types>
- <message name="getHelloWorldAsString">
- <part name="parameters" element="tns:getHelloWorldAsString"/>
- </message>
- <message name="getHelloWorldAsStringResponse">
- <part name="parameters" element="tns:getHelloWorldAsStringResponse"/>
- </message>
- <portType name="HelloWorld">

```

ZADANIA:

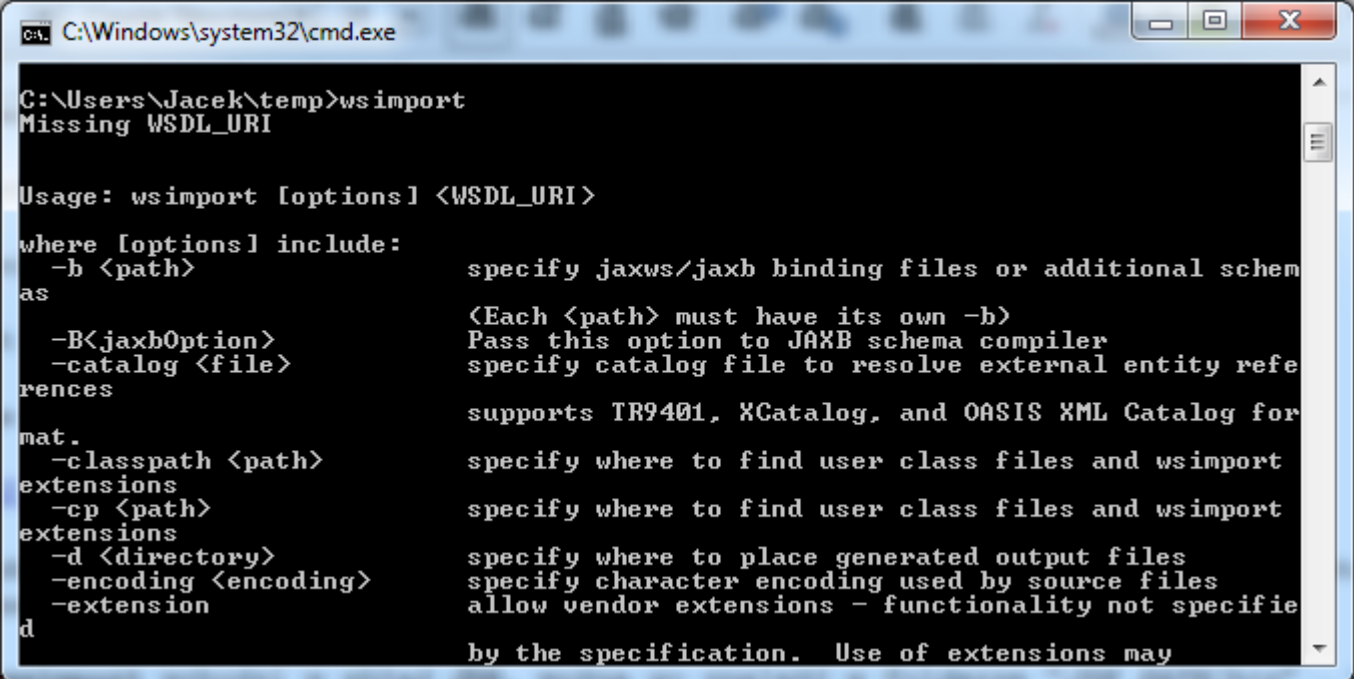
- Przeanalizować zawartość pliku WSDL
- Znaleźć znaczniki:
- <service name=...
- <port name=...
- Gdzie jest zdefiniowany XSD (XML Schema Definition)

Ćwiczenie 3. Tworzenie aplikacji klienta do web serwisu

Użyj narzędzia "**wsimport**" do parsowania publicznego pliku **wsdl**, i wygenerowanie potrzebnych dla klienta **plików** (stub) do dostępu do opublikowanego serwisu.

Narzędzie **wsimport** wchodzi w skład **JDK**, można go znaleźć w folderze "**JDK_PATH/bin**" .

Utwórz katalog np. **temp** a następnie będąc w nim uruchom **wsimport**



```
C:\Windows\system32\cmd.exe

C:\Users\Jacek\temp>wsimport
Missing WSDL_URI

Usage: wsimport [options] <WSDL_URI>

where [options] include:
  -b <path>                specify jaxws/jaxb binding files or additional schem
as                          (Each <path> must have its own -b)
  -B<jaxbOption>           Pass this option to JAXB schema compiler
  -catalog <file>          specify catalog file to resolve external entity refe
rences                      supports TR9401, XCatalog, and OASIS XML Catalog for
mat.
  -classpath <path>        specify where to find user class files and wsimport
extensions
  -cp <path>               specify where to find user class files and wsimport
extensions
  -d <directory>           specify where to place generated output files
  -encoding <encoding>     specify character encoding used by source files
  -extension               allow vendor extensions - functionality not specifie
d                            by the specification. Use of extensions may
```

```
C:\Windows\system32\cmd.exe

C:\Users\Jacek\temp>wsimport -keep http://localhost:9999/ws/hello?wsdl
parsing WSDL...

Generating code...

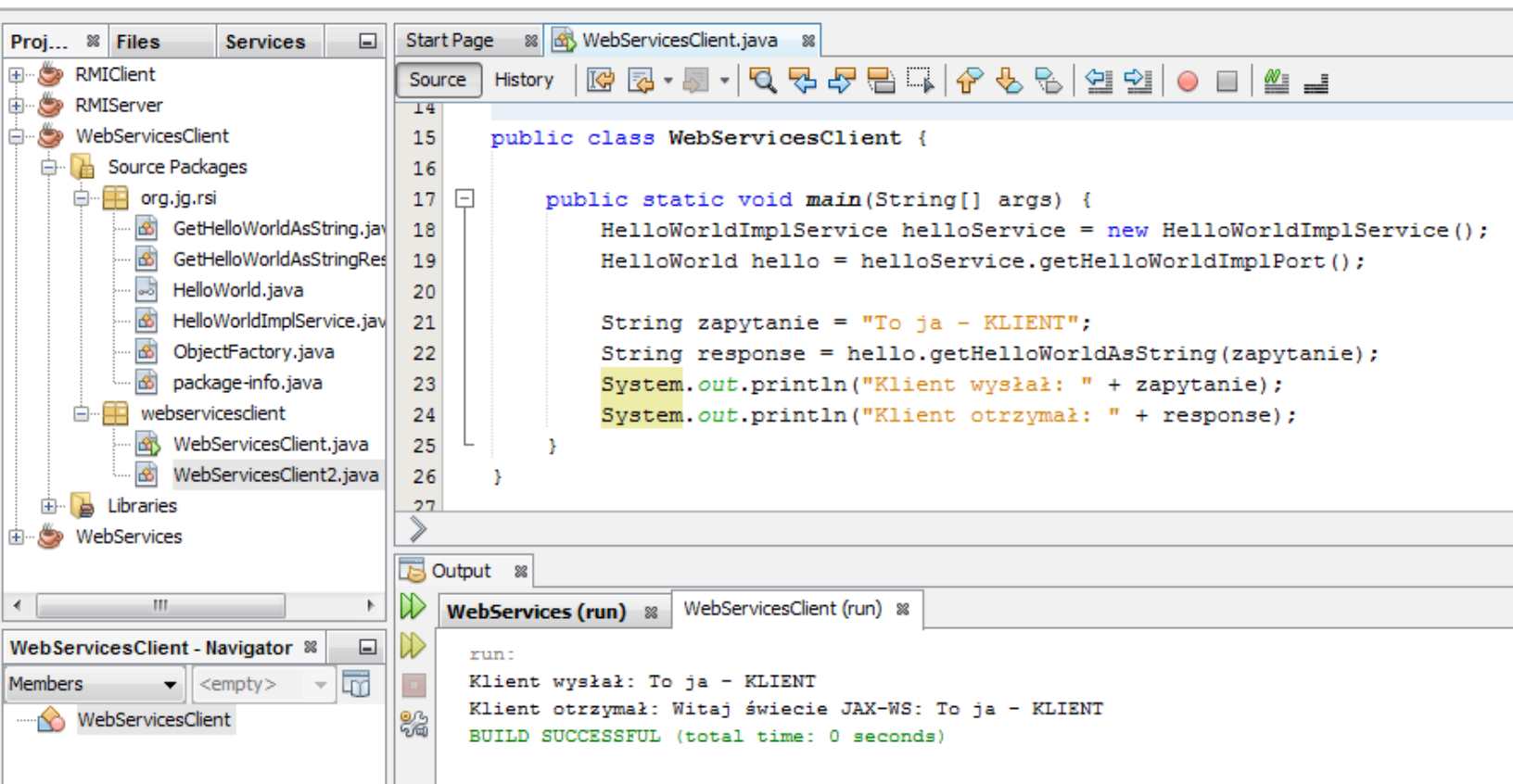
Compiling code...

C:\Users\Jacek\temp>_
```

Skopiuj wygenerowane *.java files skopiuj do swego projektu.

Dla zainteresowanych - przetestuj różne opcje programu **wsimport** (-s, -verbose, -d).

Uruchom klienta.



ZADANIA:

Sprawdź co zawierają pliki:

- HelloWorldImplService.java
- HelloWorld.java

Znajdź gdzie w pliku WSDL są wskazane nazwy:

- HelloWorldImplService

- HelloWorldImplPort
- HelloWorld

Ćwiczenie 4. Tworzenie aplikacji klienta ver.2 do web serwisu

The screenshot shows an IDE with the following components:

- Project Explorer:** Shows a project named 'WebServicesClient' with a package 'org.jg.rsi' containing several Java files, including 'WebServicesClient2.java'.
- Source Editor:** Displays the code for 'WebServicesClient2.java':


```

12 public class WebServicesClient2 {
13
14     public static void main(String[] args) throws Exception {
15         URL url = new URL("http://localhost:9999/ws/hello?wsdl");
16         QName qname = new QName("http://rsi.jg.org/", "HelloWorldImplService");
17
18         Service service = Service.create(url, qname);
19         HelloWorld hello = service.getPort(HelloWorld.class);
20
21         String zapytanie = "To ja - KLIENT 2";
22         String response = hello.getHelloWorldAsString(zapytanie);
23         System.out.println("Klient wysłał: " + zapytanie);
24         System.out.println("Klient otrzymał: " + response);
25     }
26 }
      
```
- Output Console:** Shows the execution results:


```

run:
Klient wysłał: To ja - KLIENT 2
Klient otrzymał: Witaj świecie JAX-WS: To ja - KLIENT 2
BUILD SUCCESSFUL (total time: 0 seconds)
      
```

Ćwiczenie 5. Monitorowanie pracy serwisu

tcpmon - A Utility to Monitor A TCP Connection. TCPMon is a utility that allows the user to monitor the messages passed along in TCP based conversation. It is based on a swing UI and works on almost all platforms that Java supports. The aim of this simple tutorial is to explain how TCPMon works and also to explain some of its features.



1. Client ----> SOAP envelope ----> TcpMonitor:8080
2. TcpMonitor:8080 --> SOAP envelope ---> Server:9999
3. Server:9999 ----> SOAP envelope ---> TcpMonitor:8080
4. TcpMonitor:8080 ----> SOAP envelope ---> Client

Użytkowanie tcpmon [tutorial](#)

tcpmon.jar

The screenshot shows the TCP Monitor application window. At the top, there are tabs for 'Admin' and 'Port 8080'. Below the tabs, there are buttons for 'Stop Monitor' and 'Close Tab'. The 'Local Port' is set to 8080, 'Server Name' is 127.0.0.1, and 'Server Port' is 9999. A table displays the request details:

State	Time	Request Host	Target Host	Request
Finished	Wed Feb 10 12:43:03...	127.0.0.1	127.0.0.1	GET /ws/he...

Below the table, there are buttons for 'Delete Row', 'Delete All Rows', and 'Submit to Server'. The main area displays the raw HTTP and SOAP messages:

```

Accept: text/html, image/gif, image/jpeg, *, q=.2, */*; q=.2
Connection: keep-alive

POST /ws/hello HTTP/1.1
Accept: text/xml, multipart/related
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://rsi.jg.org/HelloWorld/getHelloWorldAsStringRequest"
User-Agent: JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e
Host: localhost:8080
Connection: keep-alive

HTTP/1.1 200 OK
Date: Wed, 10 Feb 2016 11:43:03 GMT
Transfer-encoding: chunked
Content-type: text/xml; charset=utf-8

8ad
<?xml version="1.0" encoding="UTF-8"?><!-- Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e. --><!-- Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e. --><definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:
  
```

Przy użyciu klienta 2 zmień numer portu w kodzie tak aby zapytanie przechodziło przez TCP Monitor

ZADANIA:

Na podstawie otrzymanych danych z TCP Monitor

1. Przeanalizuj zapytanie od klienta

- nagłówek HTTP
- Zawartość komunikatu SOAP

2. Przeanalizuj odpowiedź do serwera

- nagłówek HTTP
- Zawartość komunikatu SOAP

PS. How to change webservice url endpoint?

```
EchoService service = new EchoService();

Echo port = service.getEchoPort();

/* Set NEW Endpoint Location */

String endpointURL = "http://NEW_ENDPOINT_URL";

BindingProvider bp = (BindingProvider)port;

String address = (String) bp.getRequestContext().get(BindingProvider.ENDPOINT_ADDRESS_PROPERTY);

address = address.replaceFirst("8080", "4040")

bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, address);

System.out.println("Server said: " + echo.echo(args[0]));
```