

# INTRO TO JAVA WS

Rajeev Gupta MTech CS  
Rgupta.mtech@gmail.com

# service oriented arch

RMI  
web service  
rest  
.net remoting

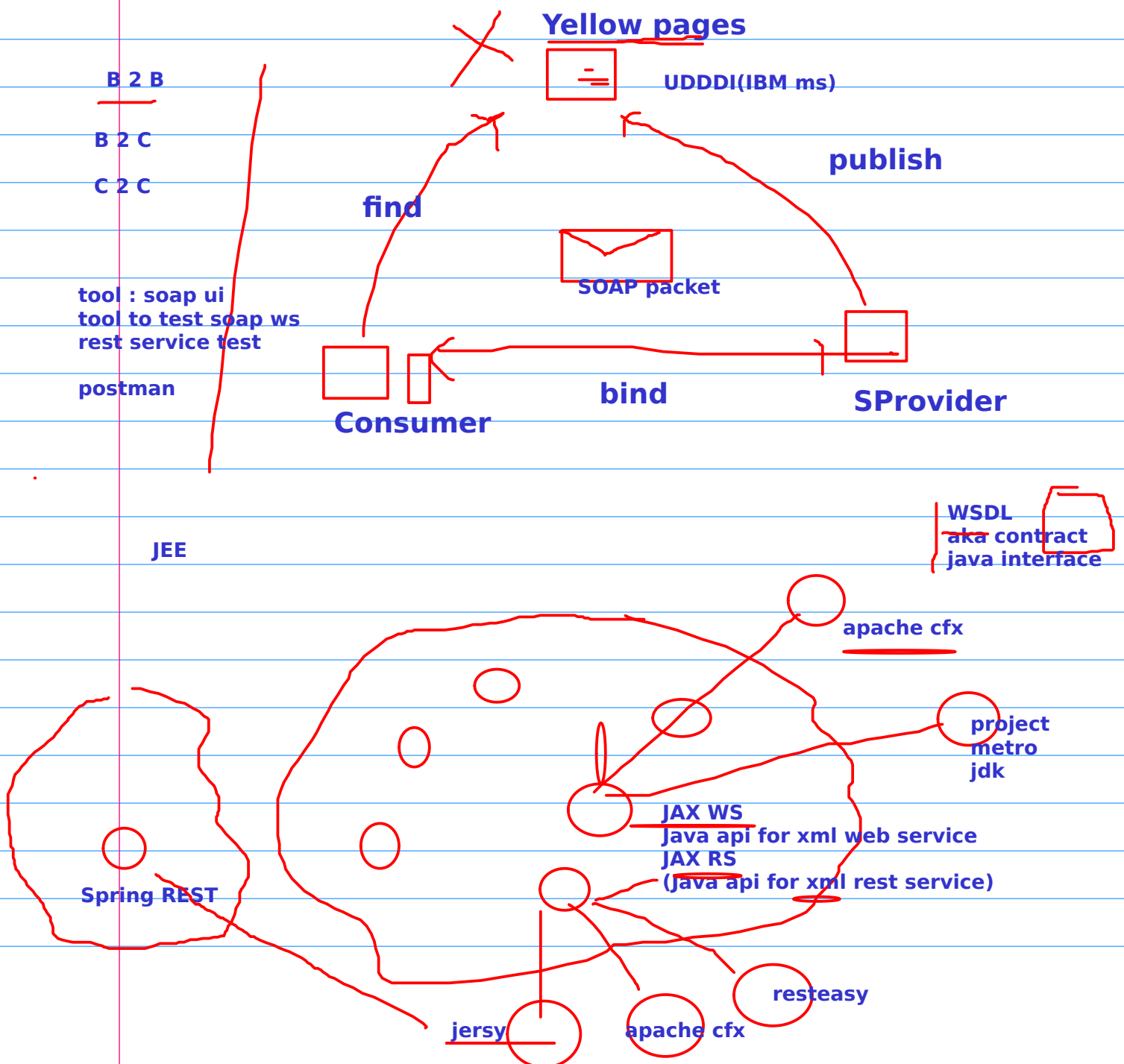
## SOAP based web service

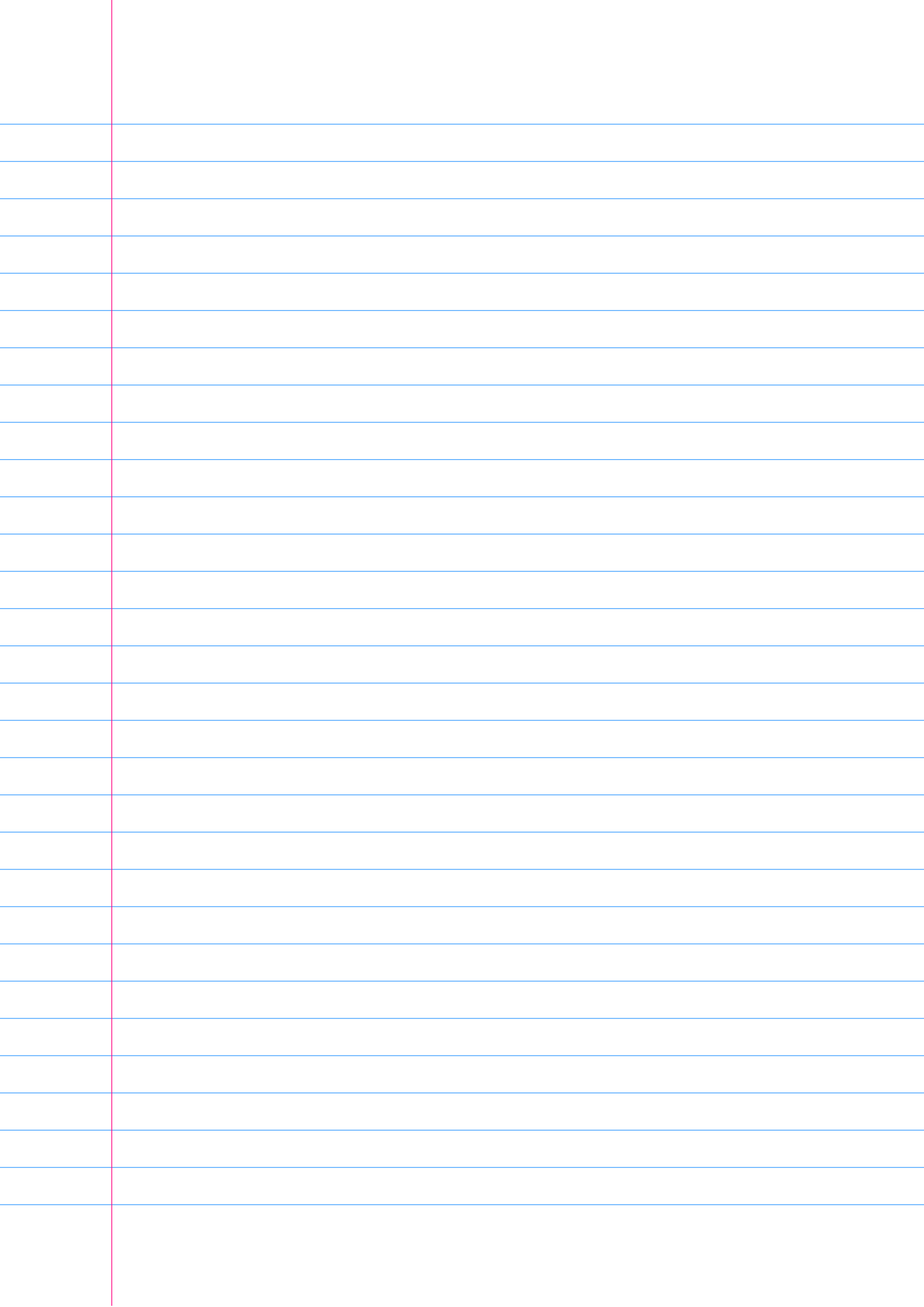
### 3 xml based protocol:

SOAP: Simple object access protocol

WSDL: web service desc language

UDDI: Universal Description, Discovery, and Integration





# Agenda



- **WS vs. Web applications**
- **Introduction to WS**
- **Introduction to SOAP, WSDL, UDDI, SEI**
- **Introduction to JAX-WS 2.x**
- **Hello World WS**

# Agenda

---

- **WS vs. Web applications**
- **Introduction to WS**
- **Introduction to SOAP, WSDL, UDDI, SEI**
- **Introduction to JAX-WS 2.x**
- **Hello World WS**

# WS vs. Web applications

## □ Two type of Web applications:-

### 1. Presentation-oriented: H to M

- A presentation-oriented web application generates interactive web pages containing various types of mark-up language (HTML, XML etc.) and dynamic content in response to requests.

### 2. Service-oriented: M to M

- A service-oriented web application implements the endpoint of a web service.
- **Presentation-oriented applications are often clients of service-oriented web applications.**

#### Web Application

- User-to-program interaction
- Static integration of components
- Monolithic service

#### Web Service

- Program-to-program interaction
- Possibility of dynamic integration of components (in the future)
- Possibility of service aggregation (in the future)

# Agenda

---

- **WS vs. Web applications**
- **Introduction to WS**
- **Introduction to SOAP, WSDL, UDDI, SEI**
- **Introduction to JAX-WS 2.x**
- **Hello World WS**

# Introduction to WS

- What is WS?
  - Web services are Web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data with calling clients
  - Distributed Applications
- Why WS?
- Application Area
  - Data Provider, B2B, EAI
- Type of WS?
  - SOAP based, RESTful
- Feature of WS?
  - have No UI
  - Interacts with applications (M to M)
  - Works with any browser client

## Web Services:

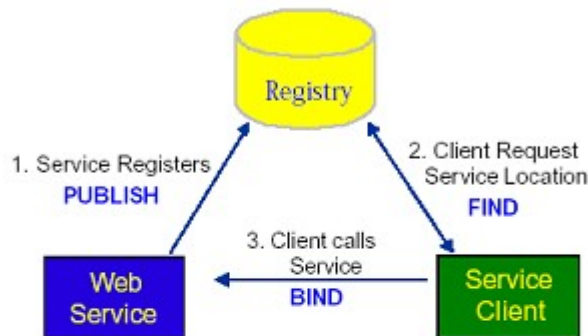
- ⌘ Are platform neutral
- ⌘ Are accessible in a standard way
- ⌘ Are accessible in an **interoperable** way
- ⌘ Use simple and ubiquitous plumbing
- ⌘ Are relatively cheap
- ⌘ Simplify **enterprise integration**





# Simplified WS Architecture

## (Simplified) Web Service Architecture



According to W3C website, <http://www.w3.org/TR/ws-desc-reqs>:

*A Web Service is a software application identified by a URI whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and [that] supports direct interactions with other software applications using XML based messages via Internet-based protocols.*

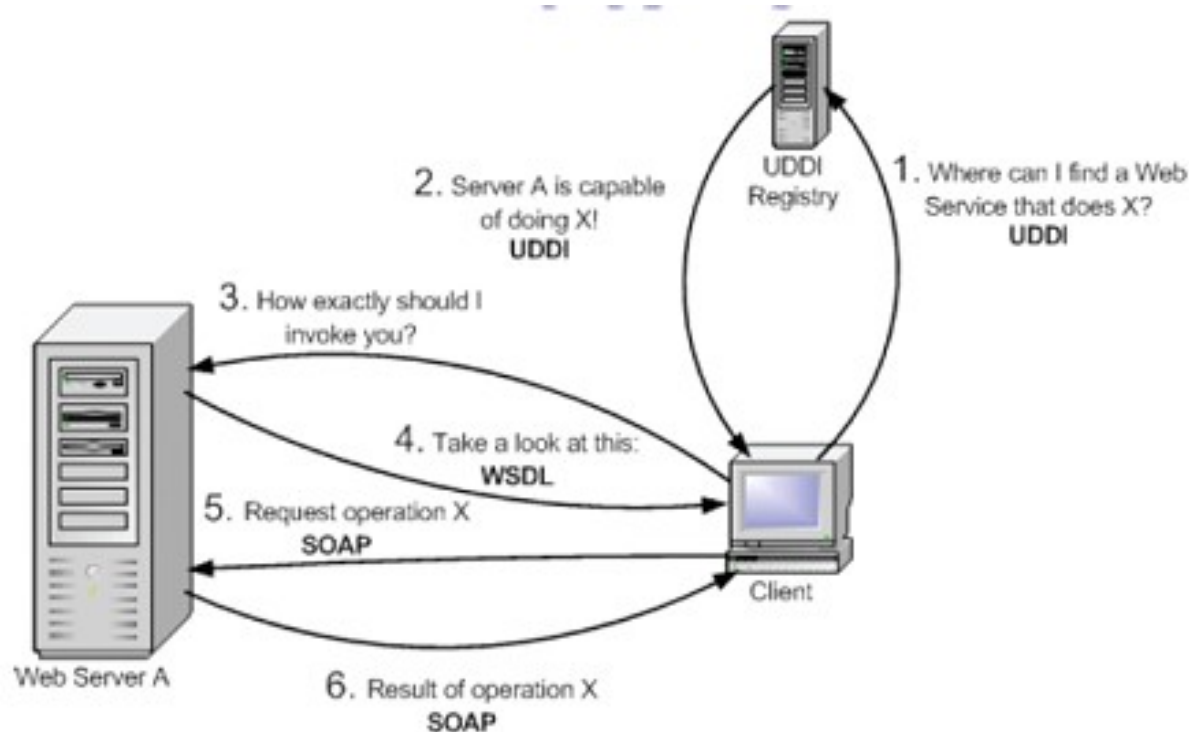
# Agenda

---

- **WS vs. Web applications**
- **Introduction to WS**
- **Introduction to SOAP, WSDL, UDDI, SEI**
- **Introduction to JAX-WS 2.x**
- **Hello World WS**

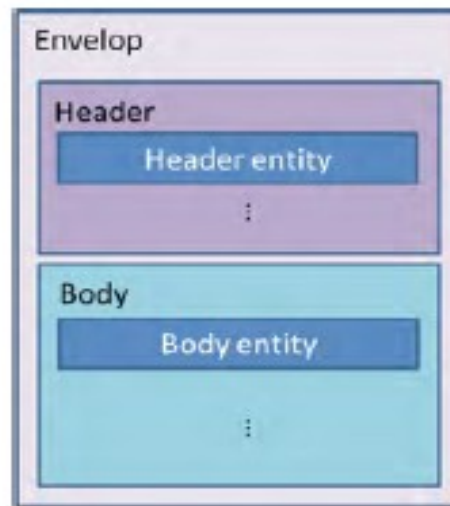
# Introduction to SOAP, WSDL, UDDI

- **Three key components of SOAP based WS**
  - **UDDI** (Universal Description, Discovery and Integration)
  - **WSDL** (Web Services Description Language)
  - **SOAP** (Simple Object Access Protocol)



# SOAP

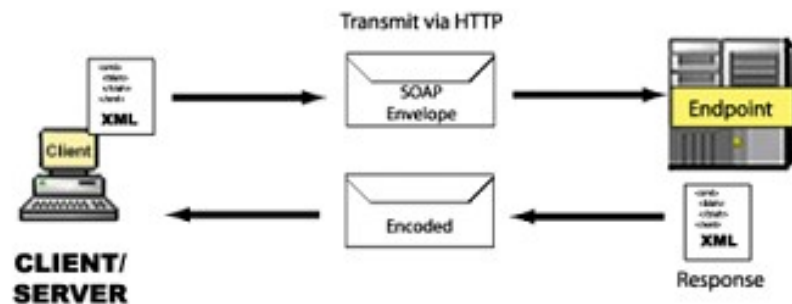
*...a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.*



**Figure 2-1.** SOAP Message Structure

# SOAP

- **How to pass java String to C++ application ; exchange of data?**
  - SOAP works same as marshalling/unmarshaling in RMI
  - SOAP format aka xml format
  - What C++ app does with java serialized data?
  - XML protocol that all to transmit data between client and service provider



# SOAP request response

*Listing 2-1. A SOAP Message Request*

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:emp="http://employees.ws.bemach.com">
  <soapenv:Header />
  <soapenv:Body>
    <emp:getEmployee>
      <emplNo>100011</emplNo>
    </emp:getEmployee>
  </soapenv:Body>
</soapenv:Envelope>
```

*Listing 2-2. A SOAP Message Response*

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getEmployeeResponse xmlns:ns2="http://employees.ws.bemach.com/"
      xmlns:ns3="http://bemach.com">
      <return>
        <emplNo>100011</emplNo>
        <firstName>Shmuel</firstName>
        <lastName>Birge</lastName>
        <birthDate>1956-07-20T00:00:00-04:00</birthDate>
        <gender>M</gender>
        <hireDate>1989-11-23T00:00:00-05:00</hireDate>
      </return>
    </ns2:getEmployeeResponse>
  </S:Body>
</S:Envelope>
```

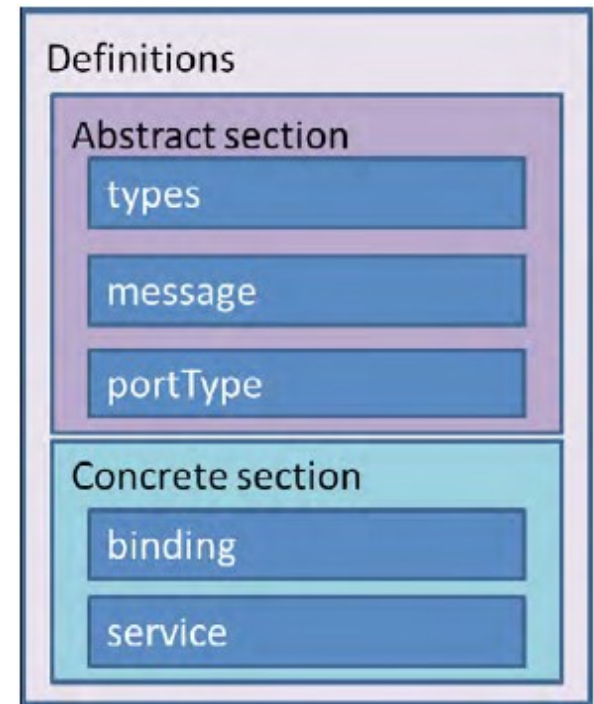
# SOAP Fault

*Listing 2-3. A SOAP Fault*

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>S:Server</faultcode>
      <faultstring>No such employee!</faultstring>
      <detail>
        <ns2:SOAPException xmlns:ns2="http://employees.ws.bemach.com/"
          xmlns:ns3="http://bemach.com">
          <message>No such employee!</message>
        </ns2:SOAPException>
      </detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

# WSDL

- Lets say we have a ws , how we could share details of our customer?
- **How to provide an interface (contract) to the consumer and that language independent too! (Interoperability)**
- **WSDL describing the following:-**
- What a service does
- the methods that the service provides
- How a service is accessed
- details of the data formats and
- protocols necessary to access a service operations
- Where a service is located?
- details of the protocol specific network address, such as a URL

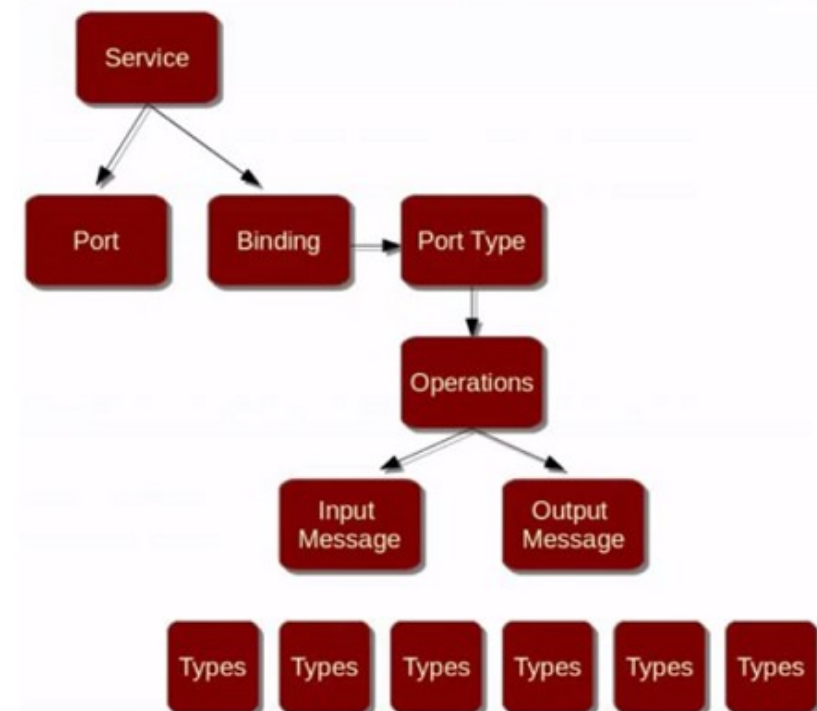




# WSDL consists of two parts

- **abstract interface**
- Types?
  - Describes the XML types & elements that are used in the messages to/from the web services
- Messages?
  - Describes the input, output and fault messages that comes into and goes out of web service operations
- PortType?
  - Names the web service and describes the operations offered by the web service
- **concrete implementation**

concrete implementation part binds the abstract interface with a concrete network address
- Bindings?
  - Describes the transport protocol like HTTP to carry the SOAP envelope for communicating with web service
- Service?
  - Describes the endpoint where the service is really available for consumption



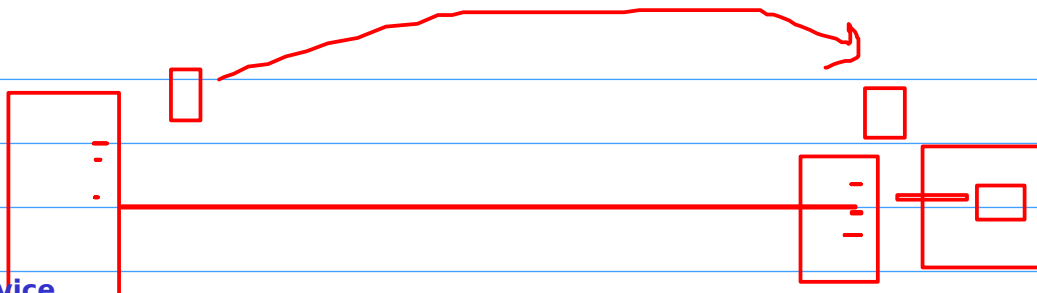
@WebService

Calculator{

}

wsdl

SEI  
jaxb code  
wsimport



# Understanding WSDL

```
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://hello.ws.bemach.com/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://hello.ws.bemach.com/"
  name="HelloWorldService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://hello.ws.bemach.com/"
        schemaLocation="http://localhost:9999/java-ws/hello?xsd=1" />
    </xsd:schema>
  </types>
  <message name="say">
    <part name="parameters" element="tns:say" />
  </message>
  <message name="sayResponse">
    <part name="parameters" element="tns:sayResponse" />
  </message>
  <portType name="HelloWorld">
    <operation name="say">
      <input message="tns:say" />
      <output message="tns:sayResponse" />
    </operation>
  </portType>
  <binding name="HelloWorldPortBinding" type="tns:HelloWorld">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <operation name="say">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <service name="HelloWorldService">
    <port name="HelloWorldPort" binding="tns:HelloWorldPortBinding">
      <soap:address location="http://localhost:9999/java-ws/hello" />
    </port>
  </service>
</definitions>
```

types

message

portType

binding

service

All data types are defined inside this element types. Usually, the element types points to an external URL that contains an XML schema which may contain other schemas. WSDL specification does not prohibit the use of other type definition systems; however, it prefers the XML schema.

A message element is an abstract data type describing the input and output of an operation. It consists of parts that are linked to types. Each operation (@WebMethod) may contain three message types: input, output, and fault. Each of these message types is unique throughout the entire WSDL document. Each has a unique name. Inside each message, the parts define the actual types that are properly defined inside the <types> element specifically (RPC style) or via a schema (Document style).

The <portType> element consists of an abstract set of operations. These operations may be supported by one or more endpoints. Operations are used to exchange messages between a service consumer and the provider.

Binding maps <portType> to a implementation specified in the <service> element. From listing 3-1, <portType> HelloWorld is bound to <port> HelloWorldPort inside the <service> element. The implementation of HelloWorld is SOAP over HTTP (<soap:binding>). SOAP specifies data formats and HTTP is a specific protocol to be used for the service offering.

A service is a collection of network-specific addresses (<port>) where the service may be rendered. <port> and <portType> are linked via a <binding> element. The connection is critical for runtime dynamic binding between the service consumers and providers.

# Service Endpoint Interface SEI

- Who does the conversion ; form Java object to SOAP objects?
  - Conversion is done by SEI (Service end point interface)
  - SEI Translate whole ws call to SOAP ; so that other side can understand it
  - SEI is specific to application to client side; we need different SEI to handle C++ client and different one for php client
  - tools generate it; wsimport
  - It translate java call to ws call !



# Agenda

---

- **WS vs. Web applications**
- **Introduction to WS**
- **Introduction to SOAP, WSDL, UDDI, SEI**
- **Introduction to JAX-WS 2.x**
- **Hello World WS**

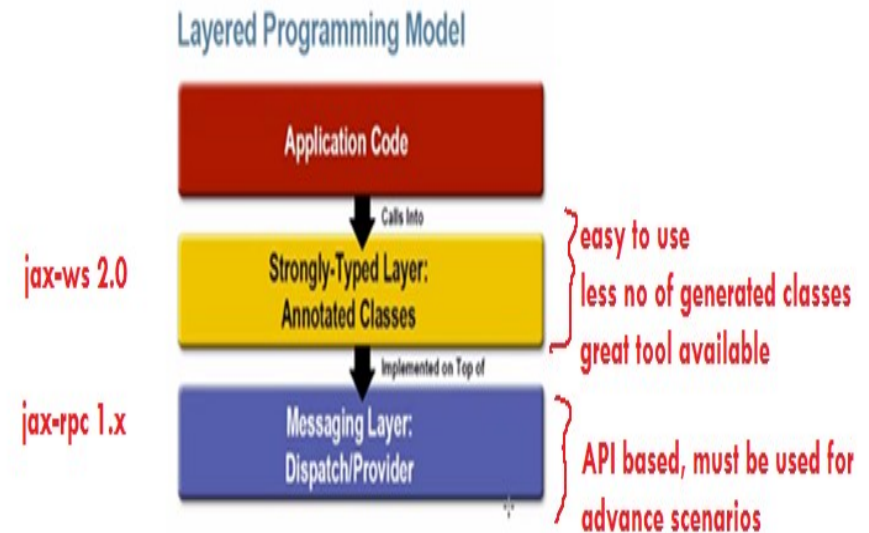
# JAX-WS 2.0

## □ What is JAX-WS 2.0?

- Simpler way to create & deploy SOAP based WS compared to JAX-WS 1.0 ( JAX-RPC)
- API stack for web services.
- New API's: JAX-WS, SAAJ, Web Service metadata
- New packages: javax.xml.ws, javax.xml.soap, javax.jws

## □ Why JAX-WS 2.0?

- POJO based
- No DD
- Layered programming model
- Both Java SE and Java EE support
- Build in data binding using JAXB 2.0
- Protocol and Transport independent

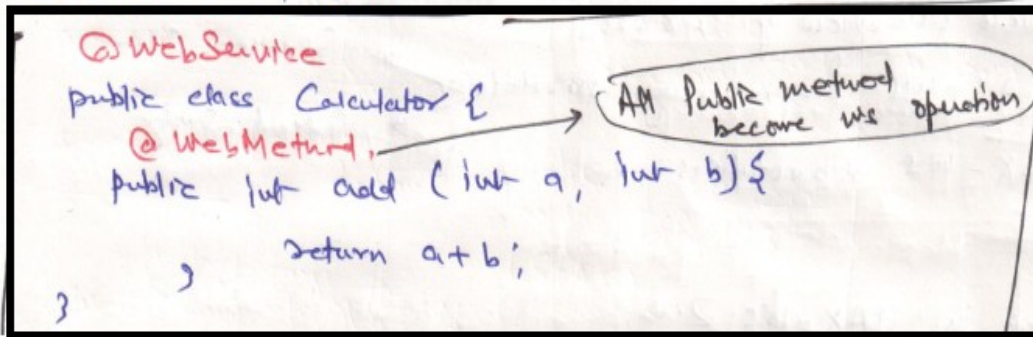


# Creating JAX WS

- Two approaches
  1. **Contract first / top down / WSDL first approach**
    - We start with WSDL
    - wsimport tool: WSDL to java code
  2. **Contract last / bottom up/ code first approach**
    - annotated POJO => Build and deploy
    - wsgen tool to generate wsdl from java code

# Contract last approach

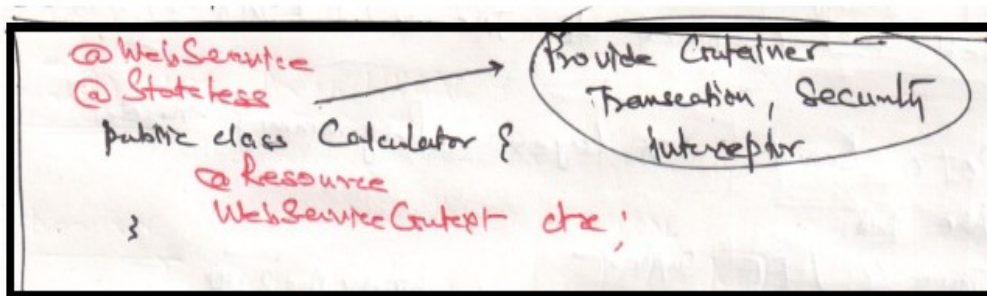
## Servlet based Endpoint



Handwritten code for a Servlet based endpoint. It shows a Java class `Calculator` with a `public add(int a, int b)` method. The code is annotated with `@WebService` and `@WebMethod`. A note in a bubble points to the `add` method, stating: "All Public method become ws operation".

```
@WebService
public class Calculator {
    @WebMethod
    public int add (int a, int b) {
        return a+b;
    }
}
```

## EJB based Endpoint



Handwritten code for an EJB based endpoint. It shows a Java class `Calculator` with a `public add(int a, int b)` method. The code is annotated with `@WebService`, `@Stateless`, and `@Resource`. A note in a bubble points to the `@Resource` annotation, stating: "Provide Container Transaction, Security Interceptor".

```
@WebService
@Stateless
public class Calculator {
    @Resource
    WebServiceContext ctx;
}
```

## Steps Contract first Approach:

1. Write POJO implementing the service
2. Add `@WebService` annotation on that
3. Optionally inject `WebServiceContext`
4. Deploy application
5. Point your client at WSDL

## What is `WebServiceContext`?

used for server side DI, provides:

1. Security Info ( `getUserPrincipal` )
2. Message context ( Bag of properties contain data that is not part of xml payload )





# Agenda

---

- **WS vs. Web applications**
- **Introduction to WS**
- **Introduction to SOAP, WSDL, UDDI, SEI**
- **Introduction to JAX-WS 2.x**
- **Hello World WS**

# CalculatorWS example

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

```
@WebService(serviceName = "CalculatorWS")
```

```
public class CalculatorWS {
```

Declare that method  
add is a WebMethod

Specify parameter  
names

```
    @WebMethod
```

```
    public int add (@WebParam (name= "value1") int value1,
```

```
    @WebParam( name="value2" ) int value2){
```

```
        return value1 + value2;
```

```
    }
```

```
}
```

# Requirements of a JAX-WS Endpoint

- The implementing class must be annotated with the `@WebService` or `@WebServiceProvider` annotation
- The business methods of the implementing class must be public.
- The business methods must not be declared static or final.
- Business methods that are exposed to web service clients must be annotated with `@WebMethod`.
- Business methods that are exposed to web service clients must have JAXB-compatible parameters and return types.
  - See the list of JAXB default data type bindings at
  - <http://docs.oracle.com/javaee/5/tutorial/doc/bnazq.html#bnazs>

# Coding the SEI implementation Class

- **@WebService** annotation at the beginning of each new web service class you create
    - Optional element **name**
      - specifies the name of the proxy class that will be generated for the client
    - Optional element **serviceName**
      - specifies the name of the class to obtain a proxy object
  - **@WebMethod** annotation at the beginning of each method that is exposed as a WSDL operation
    - Methods that are tagged with the **@WebMethod** annotation can be called remotely
    - Methods that are not tagged with **@WebMethod** are not accessible to clients that consume the web service
- ✂
- ✂ **@WebParam** annotation is used here to control the name of a parameter in the WSDL
- Without this annotation the parameter name = arg0

# Web service client

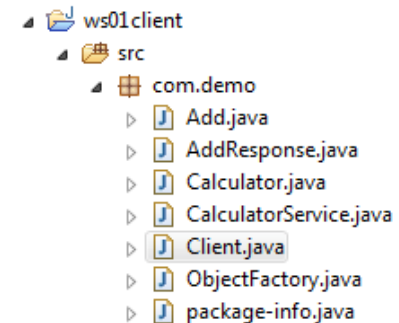
## □ From src of client project

- `wsimport -keep http://localhost:8080/hello-ws/CalculatorService?`

```
wedl
C:\00_expAug14\ws\ws01client\src>wsimport -keep http://localhost:8080/ws01/Calcu
latorService?wsdl
parsing WSDL...

Generating code...

Compiling code...
```



## □ What is wsimport ?

- The `wsimport` command-line tool processes an existing WSDL file and generates the required portable support classes (SEI) for developing JAX-WS web service applications
- Essentially, it is going to automatically generate all of the class files  
`Calculator proxy=new CalculatorService().getCalculatorPort();` ur web  
`System.out.println(proxy.add(22, 22));`

