



# Eclipsing SOA: Designing and Building Services with Eclipse STP

---

*Kyle Gabhart*

*Director of SOA Solutions for Web Age Solutions*

# Agenda

---

**Introduction**

SOA Solutions

Standards

Eclipse STP

Review

# Shameless Plug

---

## □ Who is Kyle Gabhart?

- ❖ Technology strategist and enterprise architect with a broad range of relevant experience
- ❖ Currently working with several Fortune 500's on their SOA strategies, including CalPERS and PFG as well as several state and federal agencies in the U.S. and Canada
- ❖ Author of nearly 100 articles, white papers, books, and training programs



- ❖ Open source contributor, consultant, architect and strategist on SOA and Web services since 2001
- ❖ Author of *soamatters.com*
- ❖ SOA Solutions Director for Web Age Solutions, a leading education and mentoring firm

## JavaMUG / Kyle Gabhart history

---

❑ **August 2001** – *SOAP-enabled Java Web Services*

❖ Just before the tech bubble burst

❑ **October 2007** – *SOA Patterns and Anti-Patterns*

❖ Just before we officially slipped into recession

❑ **January 2009** – *Eclipsing SOA*

❖ I sincerely apologize for whatever impending doom my presentation signals

# Agenda

---

Introduction

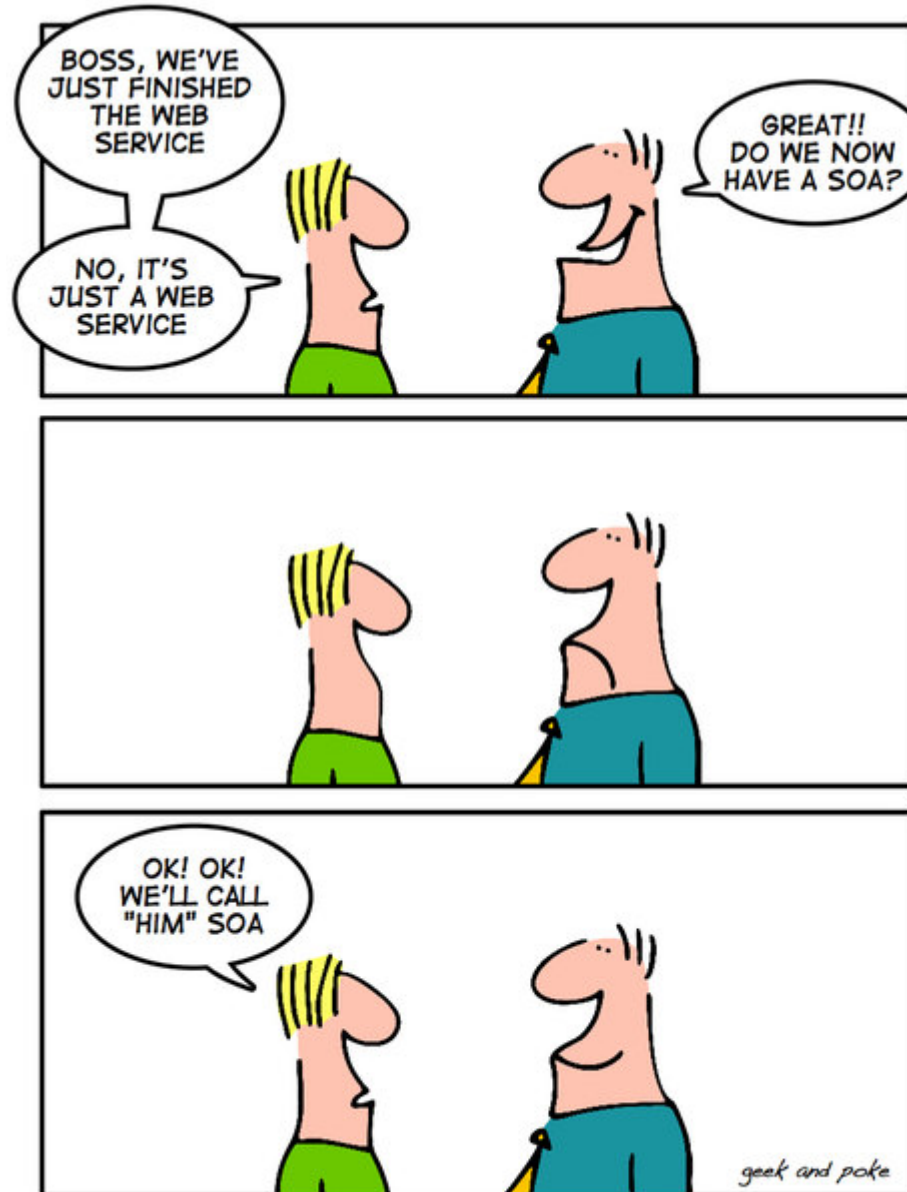
**SOA Solutions**

Standards

Eclipse STP

Review

# How to get a SOA



**HOW TO GET A SOA**

# Understanding by Analogy

---



- ❑ A tightly integrated system that works well together.
- ❑ Cannot easily modify it or add/remove components and keep it working.
- ❑ Hard wired under the covers.

# Service Oriented Architecture

---



- ❑ Component based, loosely integrated.
- ❑ Specific tasks (or services) for specific components.
- ❑ All communicate through a common interface – the amplifier (or integration framework/Internet).
- ❑ Easy to add, remove or replace components.

# How a Requirement Becomes a Service

---

## □ Discovery

1

- ❖ Collect requirements
- ❖ Document the complete solution (business process, process flow, use case, etc.)

## □ Analysis & Design

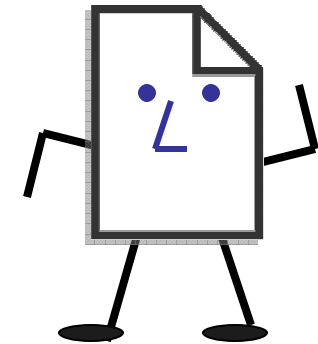
2

- ❖ Break solution into individual steps
- ❖ Identify logical services for each step
- ❖ Select actual services
- ❖ Specify service interfaces

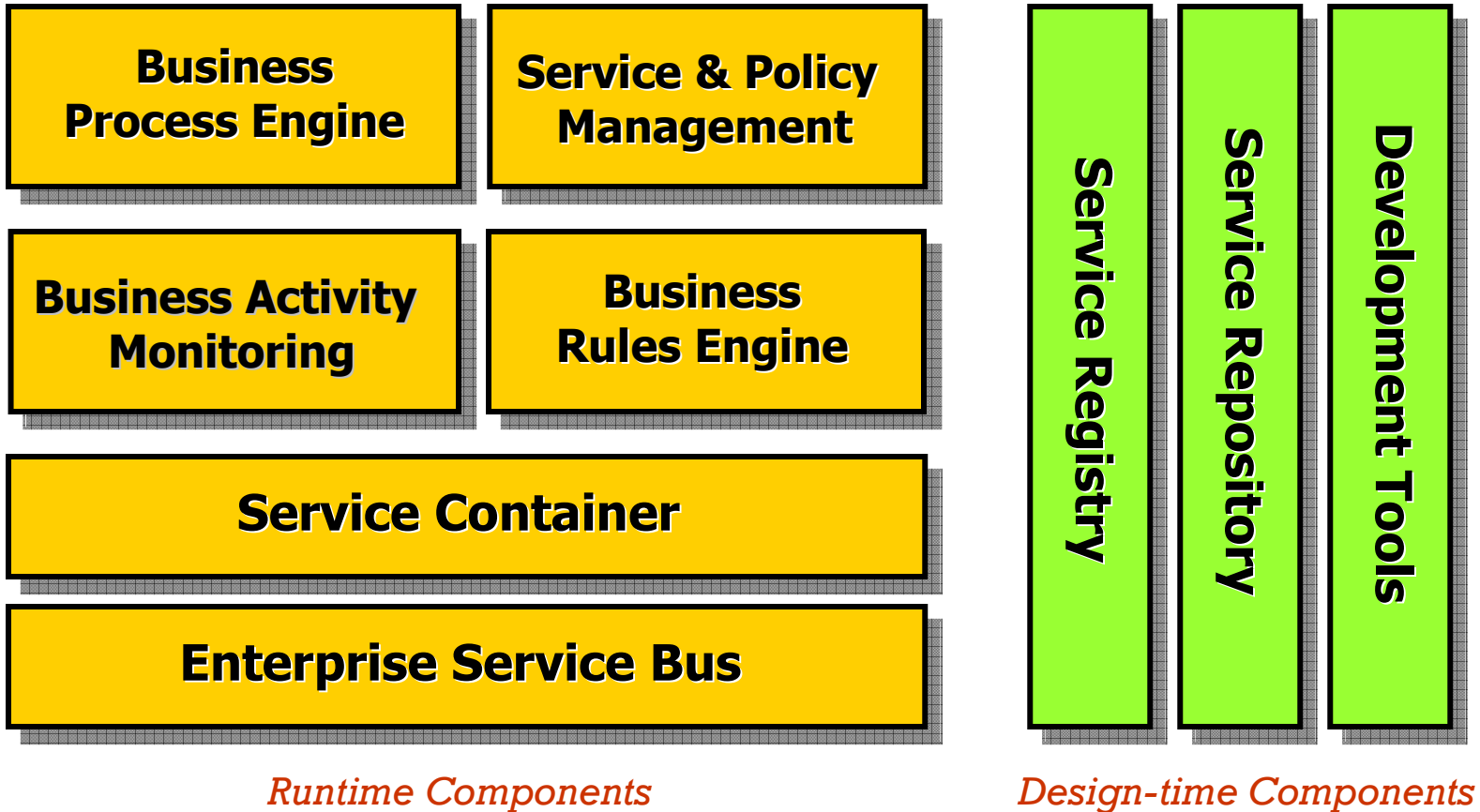
## □ Development

3

- ❖ Implement services



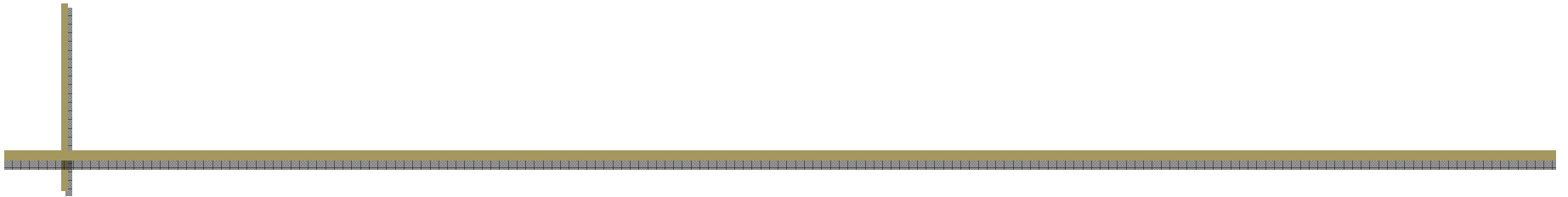
# SOA Infrastructure and Tools



# Agenda

---

- Introduction
- SOA Solutions
- Standards**
- Eclipse STP
- Review



JAX-WS

# JAX Attack

---

- ❑ In 2001, Sun Microsystems released the 'JAX Pack'
  - ❖ JAXP – Java API for XML Processing
  - ❖ JAXB – Java API for XML Binding
  - ❖ JAXM – Java API for XML Messaging
  - ❖ JAX-RPC – Java API for XML-based RPC
  - ❖ JAXR – Java API for XML Registries
  
- ❑ Since that time, additional APIs have been added
  - ❖ JAX-RS – Java API for RESTful Web Services (JSR 311)
  - ❖ JAX-TX – Java API for XML Transactions (JSR 156)
  - ❖ JAX-WS – Java API for XML-based Web Services (JSR 224)
  - ❖ JAX-WSA – Java API for XML Web Services Addressing (JSR 261)



# JAX-WS Origins

---

- ❑ JAX-RPC 1.0
  - ❖ Provided support for SOAP, WSDL, and RPC-style services
- ❑ JAX-RPC 1.1
  - ❖ Better integration with JAXB
  - ❖ Initial support for document-style services
- ❑ JAX-WS 2.0 (*Essentially JAX-RPC 2.0, but renamed*)
  - ❖ The name change reflects the move away from RPC-style and toward document-style services
  - ❖ Reduced the size of generated code by as much as 85%
  - ❖ Java 5 annotation support
  - ❖ Better data mapping, interface mapping, and programming model
- ❑ JAX-WS 2.1
  - ❖ Performance enhancements
  - ❖ New pluggable architecture to support transport modularity

# JAX-WS Architecture and Tools

---

- ❑ Java API for Web services
  - ❖ Supports XML-based Web services
  - ❖ Supports XML-based Web service Clients
  
- ❑ Standard Implementation (SI) of a Web services servlet
  - ❖ Serves as a listener for Web service calls (controller pattern)
  - ❖ Provided by JAXWS-SI.jar
  
- ❑ Declarative programming model
  - ❖ Java 5 annotations such as @WebService and @WebMethod
  - ❖ Data mapping through JAXB (@XmlElement and @XmlAttribute)
  
- ❑ Productivity tools
  - ❖ *Java 5's apt* (annotation processing tool) – generates components for the service and client
  - ❖ *wsimport / wsgen* – generates artifacts based off service descriptor
  - ❖ *Ant / maven* tasks as well as command line tools
  - ❖ And more...



# Providing a Service

---

## ❑ Option 1: WSDL → Java (*Top-down, design-first*)

- ❖ Generate a Service Endpoint Interface (SEI)

**`wsimport [options] stockquote.wsdl`**

- ❖ Implement the SEI (a stub class with annotations, data types, and method signatures)
- ❖ Package the service within a WAR and publish

## ❑ Option 2: Java → WSDL (*Bottom-up, code-first*)

- ❖ Annotate a plain Java class (`@WebService` / `@WebMethod`)
- ❖ Generate portable artifacts

**`apt [options] com.stockquote.Quote.java`**

- ❖ Package the service within a WAR and publish



# Service Source (Option 1) ...

---

## □ Service Endpoint Interface (SEI) source

```
package com.stockquote;
import javax.jws.*;
import javax.jws.soap.*;
@WebService( name="QuotePortType",
    serviceName="QuoteService",
    targetNamespace="http://www.soamatters.com" )
@SOAPBinding( style=SOAPBinding.Style.DOCUMENT,
    use=SOAPBinding.Use.LITERAL,
    parameterStyle=SOAPBinding.ParameterStyle.WRAPPED )
public interface QuotePortType extends
    java.rmi.Remote {
    @WebMethod(operationName="getQuote")
    @WebResult(name="return") public double
    addNumbers( @WebParam(name="arg0") String arg0,
    throws java.rmi.RemoteException; }
```

# Service Source (Option 1)

---

```
package com.stockquote;
import javax.jws.WebService;

@WebService (endpointInterface=
    "com.stockquote.QuotePortType")
public class Quote {
    @WebMethod
    public double getQuote( String
        symbol ) {
        //Retrieve quote from database
        return symbol;
    }
    ...
} //end Quote
```

# Service-enabling Java

---

- ❑ If you choose option #2 (convert an existing class into a Web service), there are some restrictions that must be followed.
- ❑ A valid endpoint implementation class must meet the following requirements:
  - ❖ It *must* have a *javax.jws.WebService* annotation (see JSR 181)
  - ❖ Any of its methods *may* carry a *javax.jws.WebMethod* annotation
  - ❖ All of its methods *may* throw *java.rmi.RemoteException* in addition to any other desired exception types
  - ❖ All method parameters and return types *must* be compatible with the JAXB 2.0 Java-to-XML Schema mapping definition
  - ❖ A method parameter or return value type *must not* implement the *java.rmi.Remote* interface either directly or indirectly (this is a significant change from JAX-RPC)



## Service Source (Option 2)

---

```
package com.stockquote;
import javax.jws.WebService;

@WebService
public class Quote {
    @WebMethod
    public double getQuote( String
symbol ) {
        //Retrieve quote from database
        return symbol;
    }
    ...
} //end Quote
```

# Calling a Service

Hello?



- ❑ Option 1: Port (*Dynamic proxy*)
  - ❖ Generate client artifacts  
***wsimport [options] stockquote.wsdl*** OR  
***wsgen [options] com.stockquote.Quote.class***
  - ❖ Implement the client using the generated proxy classes
  - ❖ Compile, package, and deploy (depends upon client type)
  
- ❑ Option 2: Dispatch (*Roll your own message*)
  - ❖ Create a ***javax.xml.ws.Service*** instance (WSDL abstraction)
  - ❖ Create a dispatch instance using one of two modes  
***javax.xml.ws.Service.Mode.PAYLOAD*** OR  
***javax.xml.ws.Service.Mode.MESSAGE*** object
  - ❖ Configure the request context (URI address, headers, etc.)
  - ❖ Construct message (SOAP, ReST, arbitrary XML, etc.)
  - ❖ Invoke and process response (if any)

# Client Source (Option 1)

---

```
package com.stockquote.client;
import java.rmi.RemoteException;
import com.stockquote.*;

public class StockQuoteClient {
    public static void main( String[] args
    ) {
        try {
            Quote port = new
QuoteService().getQuotePort();
            String result = port.getQuote(
"GOOG" );
            System.out.println( "Google's
            current price is: " + result);
        } catch( Exception ex ) {
            ...
        }
    }
}
```

## Client Source (Option 2) ...

---

### ❑ Create the service instance (WSDL abstraction)

```
Service service = Service.create(  
    new URL("http://xyz.com/HelloWorld?wsdl"),  
    new QName("http://xyz.com/HelloWorld",  
        "HelloService");
```

### ❑ Create a Dispatch instance

```
Dispatch<SOAPMessage> dispatch =  
    service.createDispatch(portName,  
        javax.xml.transform.Source.class,  
        Service.Mode.PAYLOAD);
```

### ❑ Configure the Request context

```
BindingProvider bp = (BindingProvider)  
    dispatch;  
Map<String, Object> rc =  
    bp.getRequestContext();  
rc.put(BindingProvider.SOAPACTION_USE_PROPERTY,  
    Boolean.TRUE);  
rc.put(BindingProvider.SOAPACTION_URI_PROPERTY,  
    "hello");
```

## Client Source (Option 2) ...

---

### ❑ Construct the message

```
SOAPMessage request = factory.createMessage();  
SOAPHeader header = request.getSOAPHeader();  
SOAPBody body = request.getSOAPBody();
```

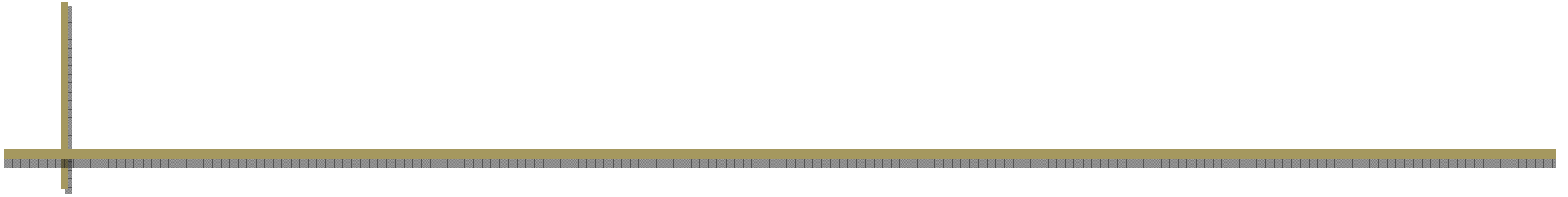
```
QName payloadName = new  
    QName("http://xyz.com/schemas/HelloWorld",  
        "hello", "ns1");  
SOAPBodyElement payload =  
    body.addBodyElement(payloadName);  
SOAPElement message =  
    payload.addChildElement("message");  
message.addTextNode("Hello World!");
```

## Client Source (Option 2)

---

### □ Invoke and process response

```
try {
    reply = dispatch.invoke(request);
    body = reply.getSOAPBody();
    QName responseName = new
    QName("http://xyz.com/schemas/HelloWorld", "helloResponse");
    SOAPBodyElement bodyElement =
    body.getChildElements(responseName
    ).next();
    String message =
    bodyElement.getValue();
} catch( WebServiceException wse )
{
    wse.printStackTrace();
}
```

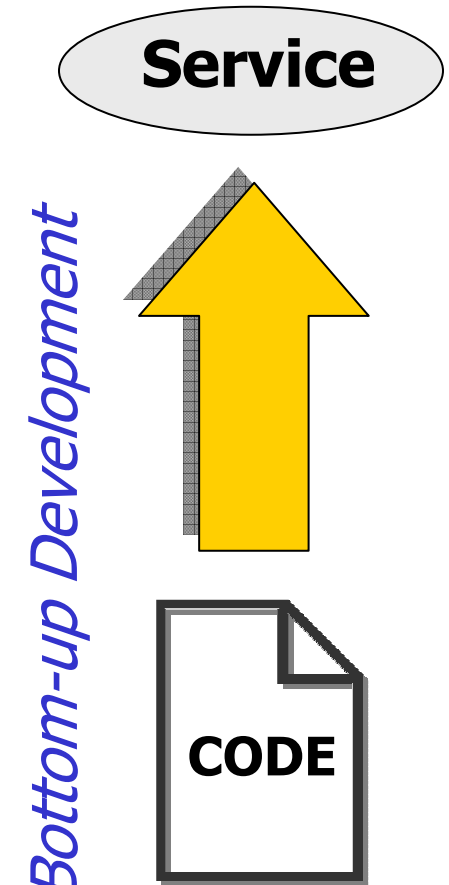


# JSR-181 - Annotations

# Start-from-Java

---

1. Write a Plain Old Java Class (POJO)
2. Annotate a Plain Old Java Class (POJO)
  1. @WebService / @WebMethod
  2. We will now call this .java file the Java Web Service (JWS) file.
3. Generate WSDL



## JSR-181 defines eight annotations

---

### □ Included annotations are:

- ❖ `javax.jws.WebService`
- ❖ `javax.jws.WebMethod`
- ❖ `javax.jws.OneWay`
- ❖ `javax.jws.WebParam`
- ❖ `javax.jws.WebResult`
- ❖ `javax.jws.HandlerChain`
- ❖ `javax.jws.soap.SOAPBinding`
- ❖ `javax.jws.soap.SOAPMessageHandlers`  
(*deprecated*)

# Core Annotations

---

- The first five annotations are the most common
  - ❖ `@WebService` = "Hey, this class is a service"
  - ❖ `@WebMethod` = "Hey, this method should be an operation"
  - ❖ `@OneWay` = "Don't expect an answer from this operation"
  - ❖ `@WebParam` = "Here's some data to go with this call"
  - ❖ `@WebResult` = "Here's the data you wanted, now go away"
  
- They cover the primary Web service needs
  - ❖ Represent the 80% of scenarios
  - ❖ Enable you to describe core service interaction (service, operations, messaging style, parameters, results, etc.)
  - ❖ The other annotations build upon these and cover more advanced capabilities

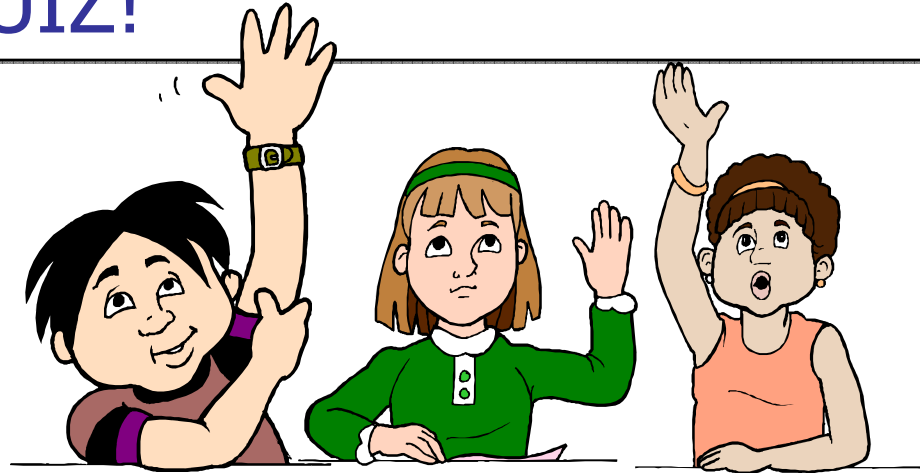
# Hello Word Example

---

```
import javax.jws.WebService;  
import javax.jws.WebMethod;  
@WebService  
public class HelloWorldService {  
    @WebMethod  
    public String helloWorld() {  
        return "Hello World!";  
    }  
}
```

## POP QUIZ!

---



- What's the difference between a service and a service oriented architecture?
- In what way does JAX-WS offer a declarative programming model?
- What other JAX spec does JAX-WS leverage?

# Agenda

---

- Introduction
- SOA Solutions
- Standards
- Eclipse STP**
- Review

# Eclipse SOA Tools Platform Project

---



- ❑ SOA Tools STP offers a broad set of SOA development capabilities
  - ❖ JAX-WS Code-First and WSDL-First
  - ❖ Service Component Architecture (SCA) support for service composition, packaging, and deployment
  - ❖ WS-Policy support
  - ❖ Business Process Modeling Notation (BPMN) design and process development
  - ❖ BPEL to Java mapping
  - ❖ Client/service auto-generation and testing

# Review

---

## □ The Good

- ❖ Free, open source
- ❖ Strong Web service support
- ❖ Good modeling support
- ❖ More than just services

## □ The Bad

- ❖ Limited infrastructure options (Apache or Sun for server, no registry, no policy enforcement)
- ❖ Full lifecycle design – development – test for processes and composite services does not exist

## □ The Ugly

- ❖ Configuration is tricky
- ❖ Runtime integration is not smooth

## Wrap-up

---

- ❑ Questions, comments, thoughts or concerns?



- ❑ *To explore this subject further, stop by my blog, SOA Matters – [www.soamatters.com](http://www.soamatters.com)*

## Additional Resources

---

❑ Book: *Service Oriented Architecture – Field Guide for Executives*



❑ Blog: SOA Matters – *soamatters.com*

❑ Training and Mentoring

❖ United States

- **1-877-517-6540**
- *getinfousa@webagesolutions.com*

❖ Canada

- **1-866-206-4644**
- *getinfo@webagesolutions.com*

