JAVA DAVE ENVIRONMENT

Web services, SOAP, Hibernate & More…

Java Boot Camp, August, 2017.

Dr. Kishore Biswas (Forrest/柯修)

PhD. Artificial Intelligence—Natural Language Processing.

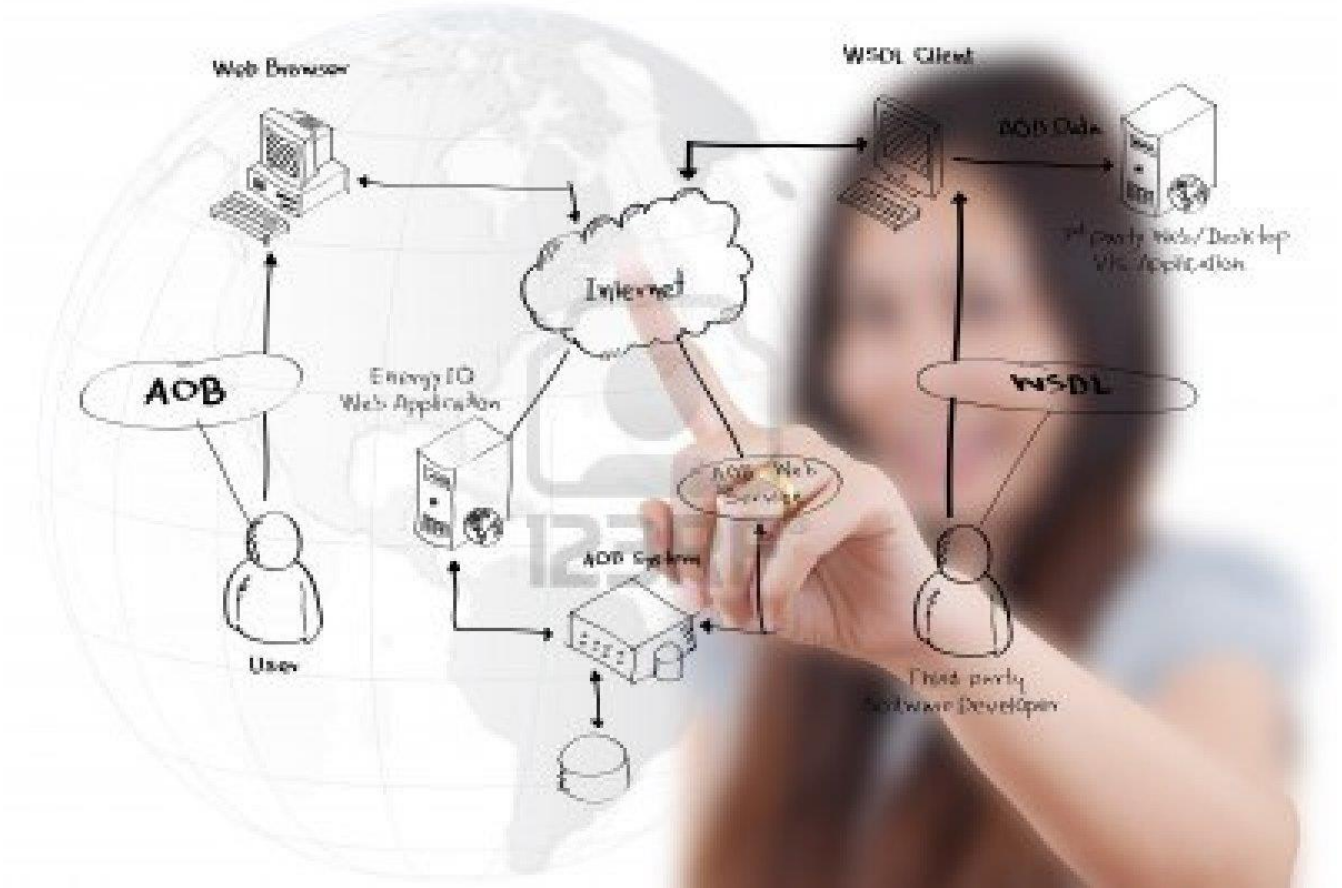
CanWay IT Training ®.

CONTENTS

1) Web services (SOAP, Restful)
2) Spring framework review…
3) Introduction to Hibernate (continued…)

1) Web services

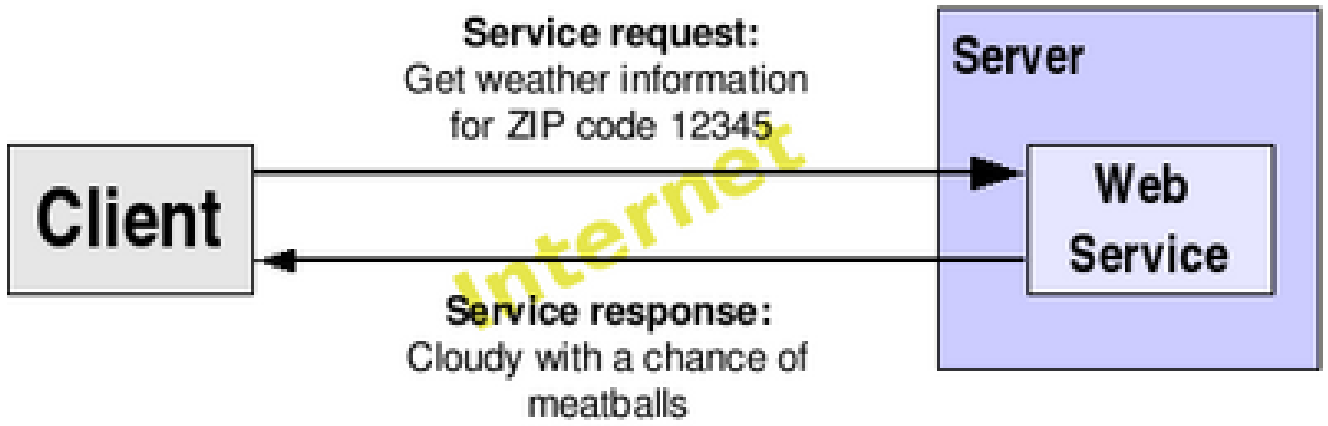
USING WEB SERVICES



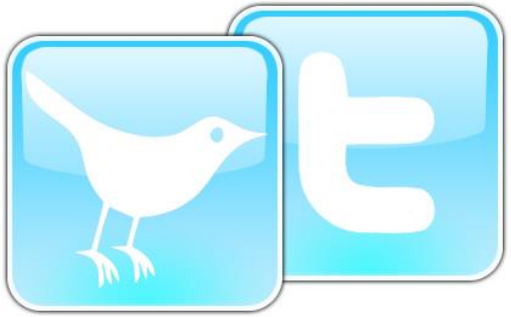
What is a Web Service?

- W3C definition: a software system designed to support interoperable machine-to-machine interaction over a network.
- A method of integrating web-based applications using open standards including but not limited to HTTP, XML, SOAP, WSDL, and JSON.

- Client and server exchange information in the form of **service request** and **service response**
- Information is passed in a *technology-independent* format
- Facilitates *loosely coupled* architecture—where client and server components may have no prior knowledge of each other except for when they need to invoke/respond to a service



EXAMPLE WEB SERVICES

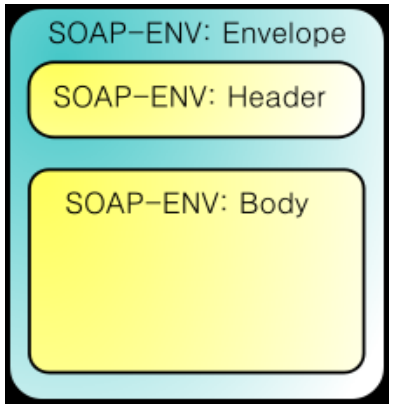


Web APIs

- Application Programming Interface (API)
  - a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API.
  - serves as an **interface** between different software programs and facilitates their interaction
- Web API
  - typically a defined set of HTTP request messages expressed in SOAP or REST along with a definition of the structure of response messages, typically expressed in JSON or XML.

SOAP

- Simple Object Access Protocol (SOAP)



- SOAP Tutorial:
  - http://www.w3schools.com/soap/soap_intro.asp

- A SOAP message is an ordinary XML document containing the following elements:
  - An Envelope element that identifies the XML document as a SOAP message
  - A Header element that contains header information
    - contains application-specific information about the SOAP message
    - optional
    - must be the first child element of the Envelope element
  - A Body element that contains call and response information
  - A Fault element containing errors and status information

REST

- Representational State Transfer (REST)
  - Use HTTP method to invoke remote services (not XML)
- The response of remote service can be in XML or any textual format
- Benefits:
  - Easy to develop
  - Easy to debug (with standard browser)
  - Leverage existing web application infrastructure
- We will focus on REST services programming for the rest of the slides.

SERVER RESPONSES

- Really Simple Syndication (RSS, Atom)
  - XML-based standard
  - Designed for news-oriented websites to “Push” content to readers
  - Excellent to monitor new content from websites
- JavaScript Object Notation (JSON)
  - Lightweight data-interchange format
  - Human readable and writable and also machine friendly
  - Wide support from most languages (Java, C, C#, PHP, Ruby, Python…)

JSON EXAMPLE

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

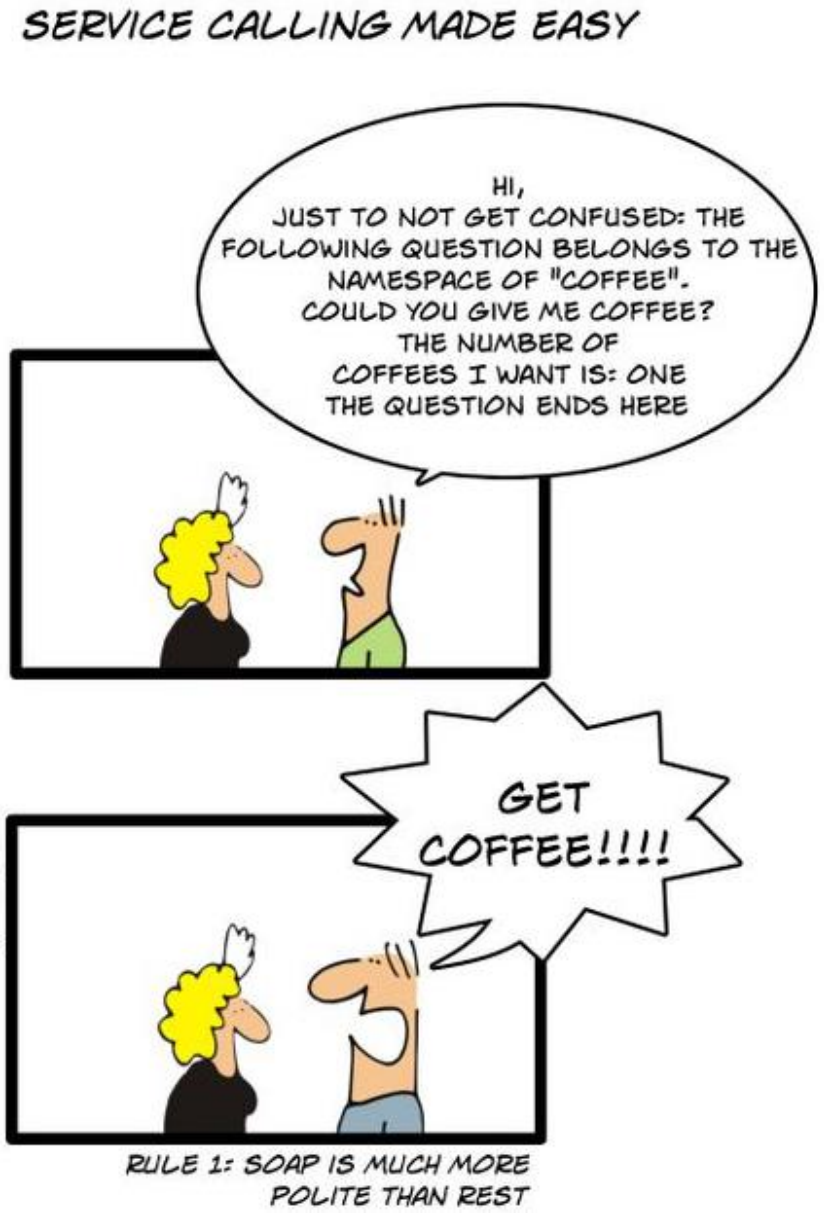
The same text expressed as XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

PRIMARY WEB SERVICES PROTOCOLS

- SOAP
  - **S**imple **O**bject **A**ccess **P**rotocol
  - Not so simple—requires toolkits to build Web services

- REST
  - **RE**presentational **S**tate **T**ransfer
  - Easy to build—requires basic knowledge of HTTP and XML



REST AND SOAP–SERVICE REQUEST

REST

http://webservice.com/weather/22102

SOAP

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
        <soap:Body>
            <m:GetWeather xmlns:m="http://mywebservice.com/weather">
                <m:Zip> 22102 </m:Zip>
            </m:GetWeather>
        </soap:Body>
</soap:Envelope>
```

XQuery Basics: What is it?

- MicroStrategy uses XQuery to access Web Services as a data source
- XQuery is a query language designed by the W3C to extract data from XML files and format/transform the results.
- XQuery is to XML as SQL is to tables in a relational database.
- XQuery allows you to:
  - Query XML data elements of interest using XPath expressions
  - Transform and restructure XML data
  - Select information based on specific criteria
  - Search and join data from multiple documents
  - Perform arithmetic calculations on number and dates
  - Manipulate strings

Developing an XQuery Report in MicroStrategy

- Web Services reports are developed by writing XQuery statements in the Freeform XQuery Editor
- MicroStrategy XQuery Editor and Generator (XEG) can be used to automate the generation of XQuery scripts
- Steps for generating an XQuery report:
  - Identify the Web Service APIs
  - (optional) Test Web Service API from the server running I-Server using a browser (REST) or browser plug-in (when header or message body must be defined)
  - Generate the XQuery script in XEG:
    - Identify source type (REST, SOAP, XML, WebDAV, REST-JSON)
    - Specify authentication mode
    - Invoke the web service to capture the XML response
    - Define report columns from the available XML nodes
    - Generate the XQuery script
  - Copy-and-paste the XQuery script into Free Form XQuery Editor

Web service concept review

Web service is a means by which computers talk to each other over the web using HTTP and other universally supported protocols.

A Web service is an application that:

- Runs on a Web server
- Exposes Web methods to interested callers
- Listens for HTTP requests representing commands to invoke Web methods
- Executes Web methods and returns the results

Web services is based on:

- **HTTP (Hypertext Transport Protocol)**
- **SOAP (Simple Object Access Protocol)**
- **UDDI (Universal Description, Discovery and Integration)**
- **WS-POLICY (Web Services Policy)**

Most Web services expect their Web methods to be invoked using HTTP requests containing SOAP messages. SOAP is an XML-based vocabulary for performing remote procedure calls using HTTP and other protocols.

SAMPLE WEB SERVICE

Calc.asmx

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/WebService.cs" Class="WebService" %>
using System;
using System.Web.Services;

[WebService (Name="Calculator Web Service",
Description = "Perform simple math over the Web")]
class CalcService
{
[WebMethod (Description = "Computes the sum of two integers")]
public int Add (int a, int b) { return a+b;}
[WebMethod (Description = "Computes the difference between two integers")]
public int Subtract (int a, int b) { return a-b;}
}
```

The example demonstrates several important principles of Web service programming using the .NET Framework:

- Web services are implemented in ASMX files. ASMX is a special file name extension registered to ASP.NET (specifically, to an ASP.NET HTTP handler) in Machine.config.

- ASMX files begin with @ WebService directives. At a minimum, the directive must contain a Class attribute identifying the class that makes up the Web service.

- Web service classes can be attributed with optional WebService attributes. The one in the previous example assigns the Web service a name and a description that show up in the HTML page generated when a user calls up Calc.asmx in his or her browser.

- Web methods are declared by tagging public methods in the Web service class with WebMethod attributes. You can build helper methods into a Web service—methods that are used internally by Web methods but that are not exposed as Web methods themselves—by omitting the Webmethod attribute.

TESTING A WEB SERVICE :

ASP.NET responds to the HTTP request for Calc.asmx by generating an HTML page that describes the Web service.

- The name and description in the ASMX file's WebService attribute appear at the top of the page.
- Underneath is a list of Web methods that the service exposes, complete with the descriptions spelled out in the WebMethod attributes.

Click "Add" near the top of the page, and ASP.NET displays a page that you can use to test the Add method .

- ASP.NET knows the method name and signature because it reads them from the metadata in the DLL it compiled from Calc.asmx. It even generates an HTML form that you can use to call the Add method with your choice of inputs.
- The XML returned by the Web method appears in a separate browser window

The forms that ASP.NET generates on the fly from ASMX files enable you to test the Web services that you write without writing special clients to test them with.

Suppose you write a Web service that publishes Web methods named Add and Subtract that callers can use to add and subtract simple integers. If the service's URL is www.wintellect.com/calc.asmx, here's how a client would invoke the Add method by transmitting a SOAP envelope in an HTTP request. This example adds 2 and 2:

```
POST /calc.asmx HTTP/1.1
Host: www.wintellect.com
Content-Type: text/xml; charset=utf-8
Content-Length: 338
SOAPAction: http://tempuri.org/Add

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema instance"
    xmlns: xsd=http://www.w3.org/2001/XMLSchema
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
     <soap:Body>
        <Add xmlns="http://tempuri.org/">
          <a>2</a>
            <b>2</b>
        </Add>
    </soap:Body>
  </soap:Envelope>
```

And here’s how the Web service would respond:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 353

<?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:xsd=http://www.w3.org/2001/XMLSchema
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <AddResponse xmlns="http://tempuri.org/">
        <AddResult>4</AddResult>
      </AddResponse>
    </soap:Body>
  </soap:Envelope>
```

The Web service's job is to parse the SOAP envelope containing the inputs, add 2 and 2, formulate a SOAP envelope containing the sum of 2 and 2, and return it to the client in the body of the HTTP response. This, at the most elemental level, is what Web services are all about.

Web services written with the .NET Framework also allow their Web methods to be invoked using ordinary HTTP GET and POST commands. The following GET command adds 2 and 2 by invoking the Web service's Add method:

```
GET /calc.asmx/Add?a=2&b=2 HTTP/1.1
Host: www.wintellect.com
```

The Web service responds as follows:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 80

<?xml version="1.0" encoding="utf-8"?>
  <int xmlns="http://tempuri.org/">4</int>
```

Here’s a POST command that adds 2 and 2:

```
POST /calc.asmx/Add HTTP/1.1
Host: www.wintellect.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

a=2&b=2
```

And here’s the Web service’s response:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 80

<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://tempuri.org/">4</int>
```

The hard part of writing a Web service is parsing HTTP requests and generating HTTP responses. The .NET Framework insulates developers from the low-level details of HTTP, SOAP, and XML and provides a high-level framework for writing Web services and Web service clients alike.

Web Services Description Language - WSDL

If other developers are to consume (that is, write clients for) a Web service that you author, they need to know :

- What Web methods your service publishes
- What protocols it supports
- The signatures of its methods
- The Web service's location (URL)

All this information and more can be expressed in a language called the Web Services Description Language, or WSDL for short.

WSDL is an XML vocabulary

Web Service Discovery—DISCO and UDDI

Once a client has a WSDL contract describing a Web service, it has all the information it needs to make calls to that Web service.

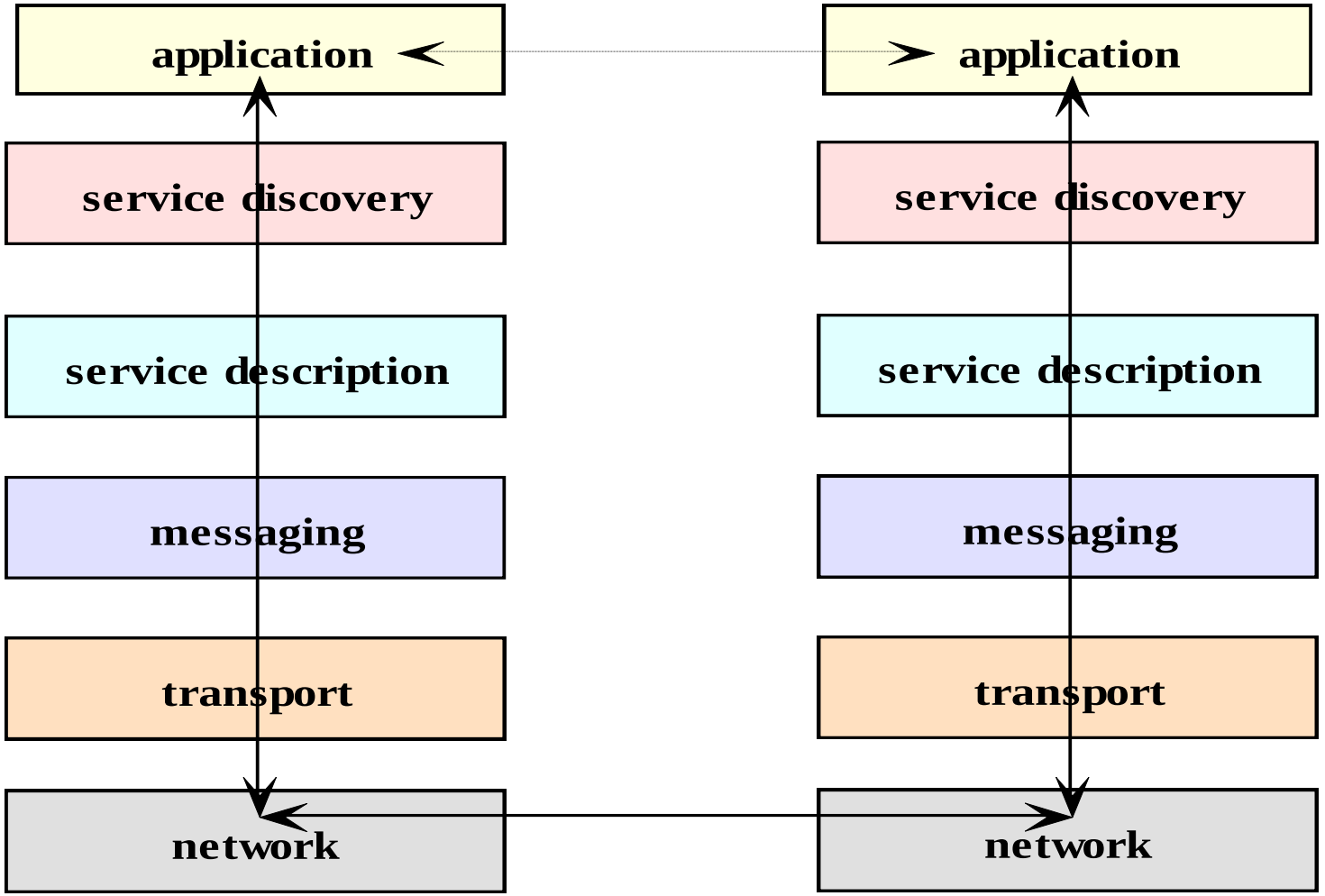
But when you publish a Web service by making it available on a Web server, how do clients find out where to get a WSDL contract? For that matter, how do clients know that your Web service exists in the first place?

The answer comes in two parts:

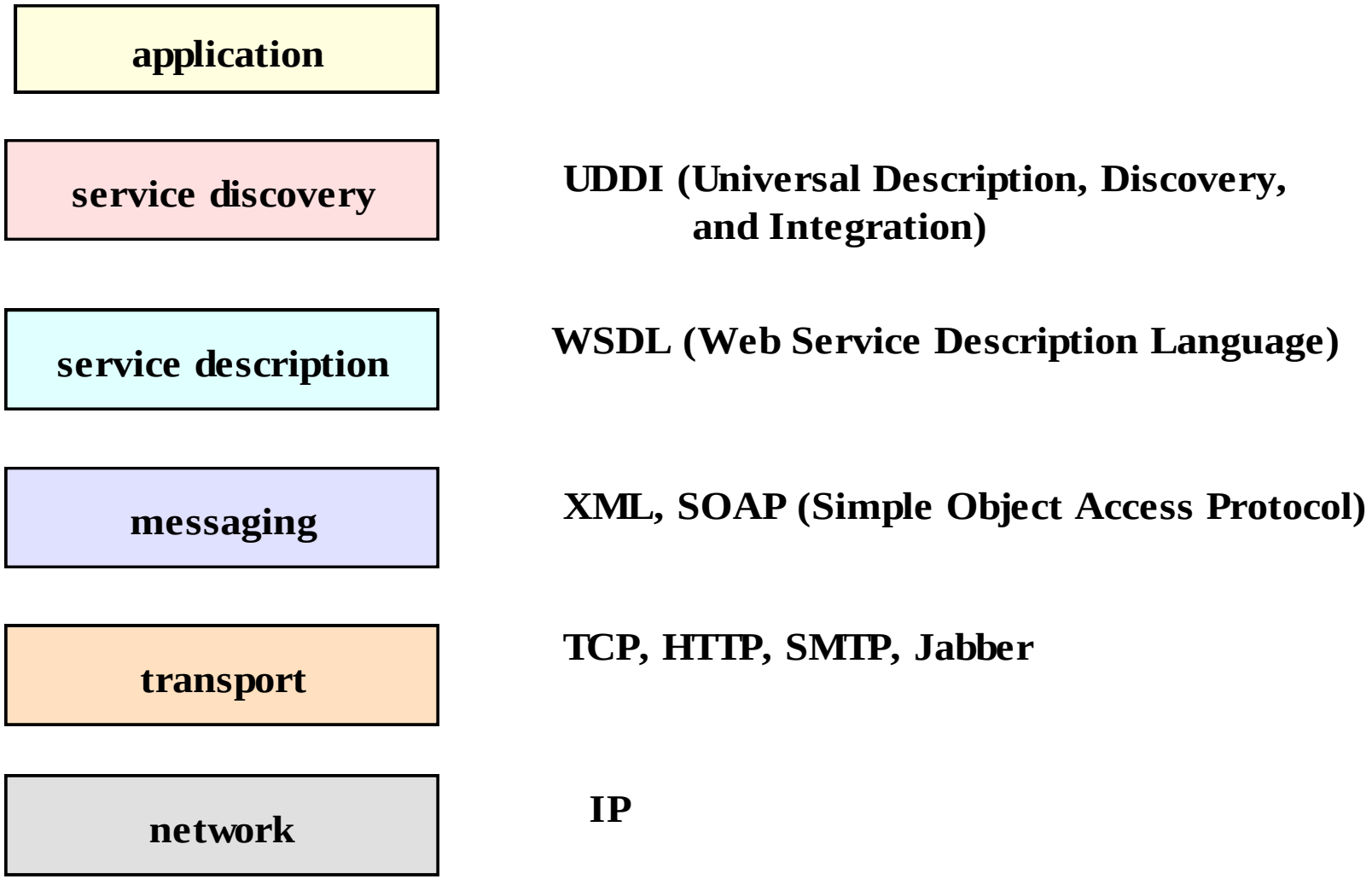
DISCO and Universal Description, Discovery, and Integration (UDDI)

- DISCO is a file-based mechanism for local Web service discovery—that is, for getting a list of available Web services from DISCO files deployed on Web servers.
- UDDI is a global Web service directory that is itself implemented as a Web service.

Web service protocol stack



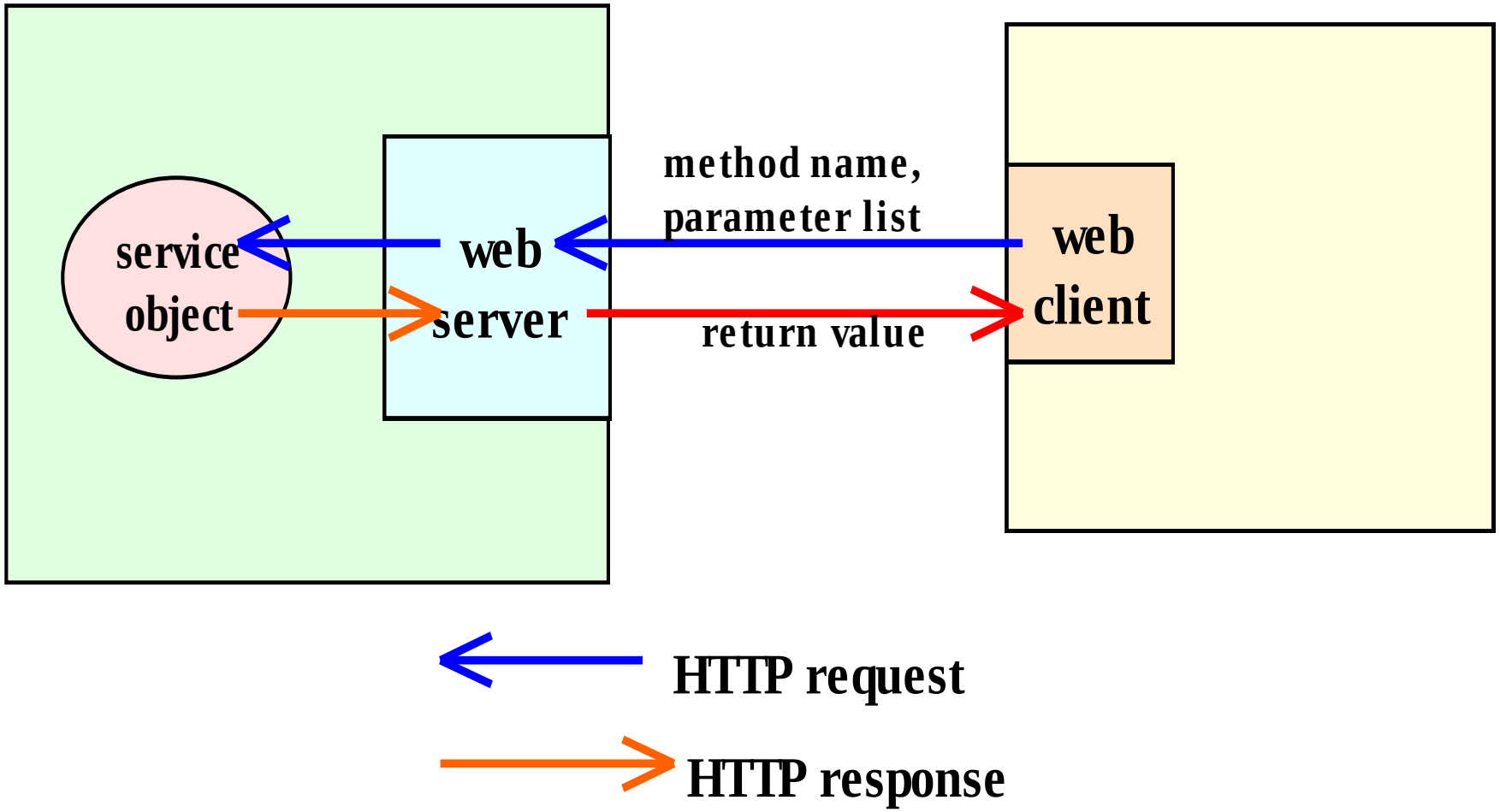
WEB SERVICE PROTOCOLS



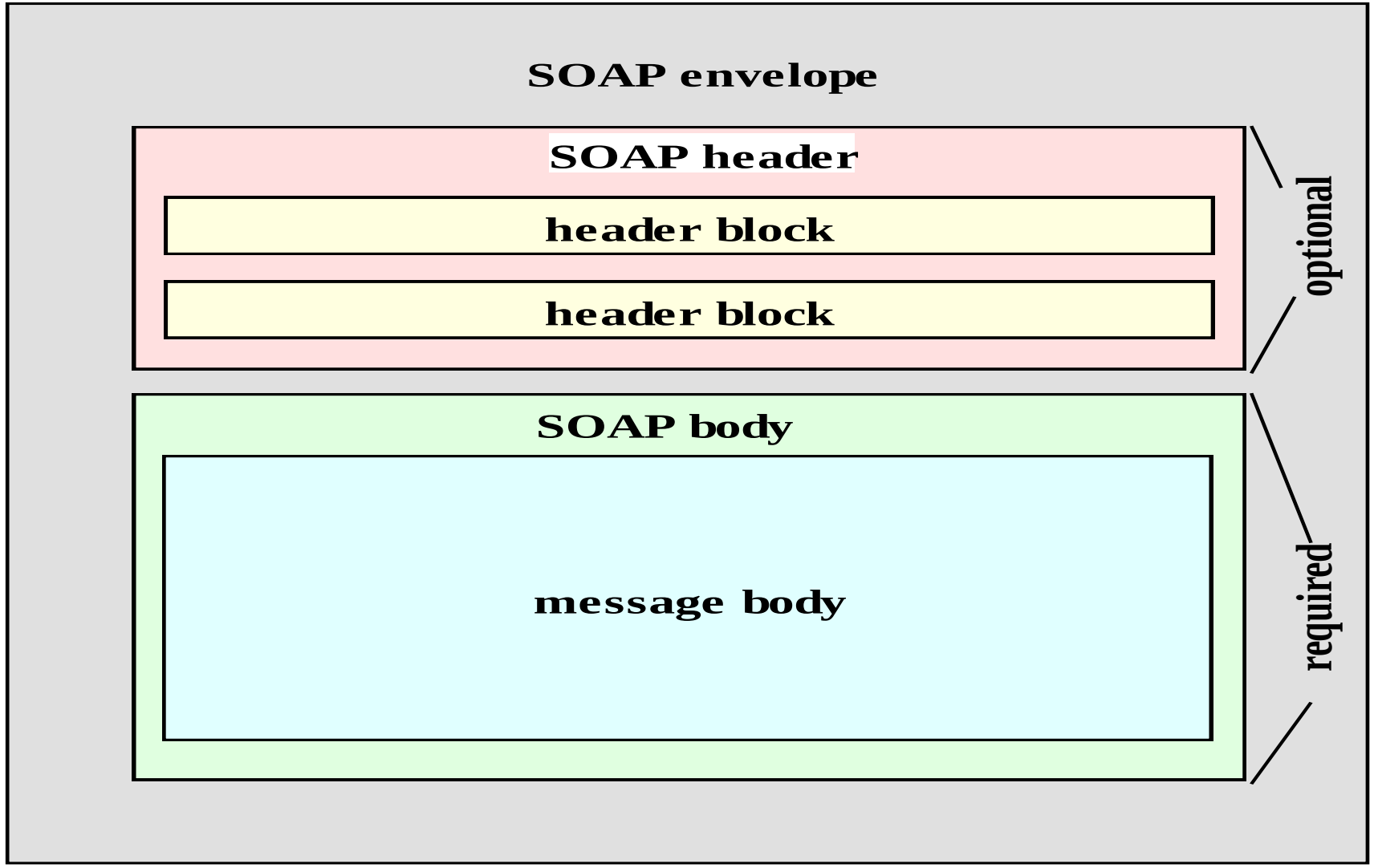
SOAP

- **SOAP** is a protocol, which applies XML for message exchange in support of remote method calls over the Internet.
- Compared to Remote Method Invocation (RMI) or CORBA-based facilities:
  - SOAP is web-based or "wired" and hence is not subject to firewall restrictions
  - is Language-independent
  - provides just-in-time service integration--as requesters use a service broker (e.g., UDDI--Universal Description, Discovery, and Integration) to find services, the discovery will take place dynamically.

REMOTE PROCEDURE CALL USING HTTP



SOAP MESSAGES



SOAP MESSAGES

```
<soap:Envelope xmlns:soap='http://www.w3.org/2001/10/soap-envelope'>
  <soap:Header>

    <-- Headers go here -->

  </soap:Header>
  <soap:Body>

    <-- Request goes here -->

  </soap:Body>
</soap:Envelope>
```

Source: http://www.xml.com/

EXAMPLE OF A SOAP MESSAGE

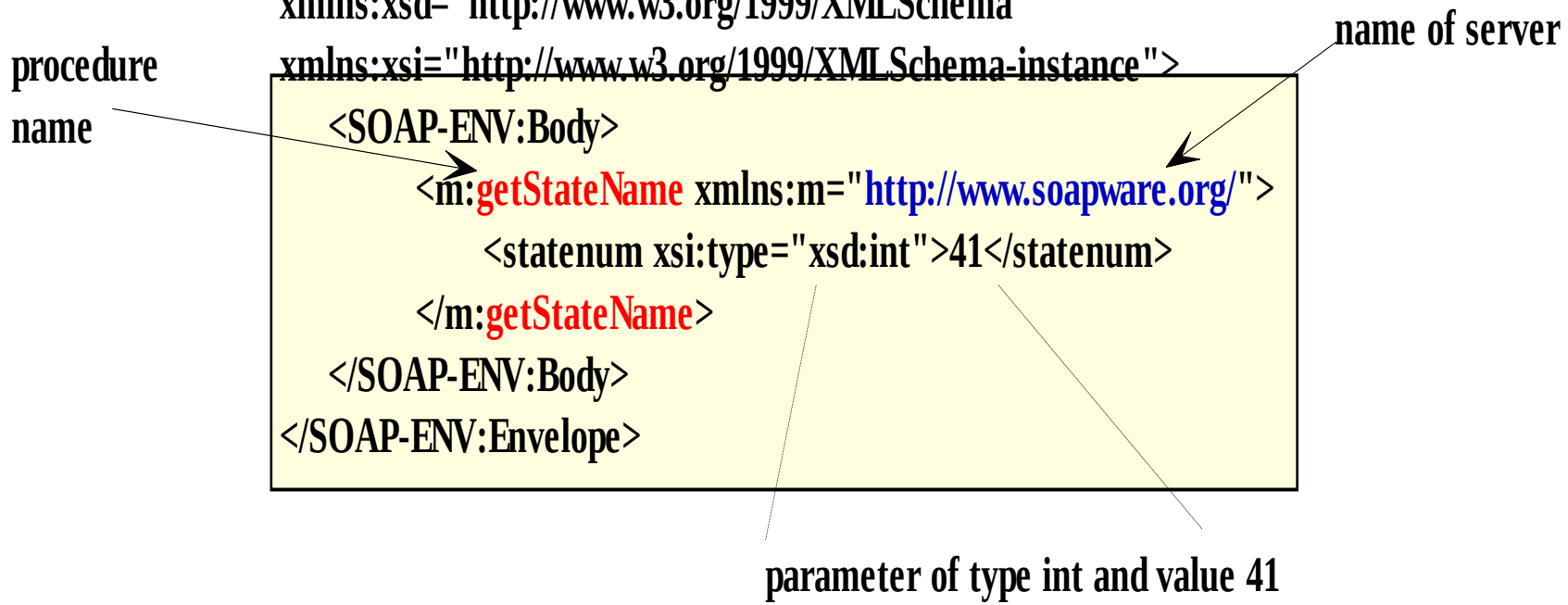
```
<soap:Envelope xmlns:soap='http://www.w3.org/2001/10/soap-envelope'>
  <soap:Header>
    <h:Log xmlns:h='http://example.org/cvs/logging'>
      <trace>4</trace>
    </h:Log>
    <h:PServer xmlns:h='http://example.org/cvs/pserver'
            soap:mustUnderstand='true' >
      <username>anoncvs@example.org</username>
      <password>anoncvs</password>
    </h:PServer>
  </soap:Header>
  <soap:Body>
    <m:Checkout xmlns:m='http://example.org/cvs'>
      <source>/xml/soap/webservices/cvs</source>
      <revision>1.1</revision>
      <destination>/etc/usr/marting/source/xml</destination>
    </m:Checkout>
  </soap:Body>
</soap:Envelope>
```

source: http://www.xml.com

An Example SOAP Request

Source: (http://www.soaprpc.com/tutorials/) A Busy Developer's Guide to SOAP1.1

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <SOAP-ENV:Body>
        <m:getStateName xmlns:m="http://www.soapware.org/">
            <statenum xsi:type="xsd:int">41</statenum>
        </m:getStateName>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



RESPONSE EXAMPLE

SOURCE: (HTTP://WWW.SOAPRPC.COM/TUTORIALS/) A BUSY DEVELOPER'S GUIDE TO SOAP1.1

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope SOAP-ENV:
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <m:getStateNameResponse xmlns:m="http://www.soapware.org/">
      <Result xsi:type="xsd:string">South Dakota</Result>
    </m:getStateNameResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

procedure name

name of server

returned value

Example of a SOAP message for an error RPC response

http://www.soapware.org/bdg

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope SOAP-ENV:
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:Client</faultcode>
            <faultstring>
                Can't call getStateName because there are
                too many parameters.
            </faultstring>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

HTTP AND SOAP RPC REQUEST

SOURCE: (HTTP://WWW.SOAPRPC.COM/TUTORIALS/) A BUSY DEVELOPER'S GUIDE TO SOAP1.1

A SOAP message can be used to transport a SOAP remote procedure request/response, as follows:

```
POST    /examples    HTTP/1.1
User-Agent: Radio UserLand/7.0 (WinNT)
Host: localhost:81
Content-Type: text/xml; charset=utf-8
Content-length: 474
SOAPAction: "/examples"
<blank line>
<text for SOAP message>
```

An HTTP REQUEST that carries a SOAP RPC request

Source: (http://www.soaprpc.com/tutorials/) A Busy Developer's Guide to SOAP1.1

```
 POST    /examples    HTTP/1.1
User-Agent: Radio UserLand/7.0 (WinNT)
Host: localhost:81
Content-Type: text/xml; charset=utf-8
Content-length: 474
SOAPAction: "/examples"

<?xml version="1.0"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=

 "http://schemas.xmlsoap.org/soap/encoding/" xmlns:

 SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:

 SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
<SOAP-ENV:Body>
      <m:getStateName xmlns:m="http://www.soapware.org/">
        <statenum xsi:type="xsd:int">41</statenum>
        </m:getStateName>
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

AN HTTP REQUEST THAT CARRIES A SOAP RPC RESPONSE

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 499
Content-Type: text/xml; charset=utf-8
Date: Wed, 28 Mar 2001 05:05:04 GMT
Server: UserLand Frontier/7.0-WinNT

<?xml version="1.0"?>
<SOAP-ENV:Envelope SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <m:getStateNameResponse xmlns:m="http://www.soapware.org/">
      <Result xsi:type="xsd:string">South Dakota</Result>
      </m:getStateNameResponse>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

AN HTTP REQUEST THAT CARRIES A SOAP ERROR MESSAGE

```
HTTP/1.1 500 Server Error
Connection: close
Content-Length: 511
Content-Type: text/xml; charset=utf-8
Date: Wed, 28 Mar 2001 05:06:32 GMT
Server: UserLand Frontier/7.0-WinNT

 <?xml version="1.0"?>
<SOAP-ENV:Envelope SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode> SOAP-ENV:Client </faultcode>
      <faultstring> Can't call getStateName because there are too many
            parameters.</faultstring>
      </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Advantages of Web Services

- Web services provide **interoperability** between various software applications running on disparate platforms.
- Web services use open standards and protocols. Protocols and data formats are text-based where possible, making it easy for developers to comprehend.
- By utilizing **HTTP**, web services can work through many common firewall security measures without requiring changes to the firewall filtering rules. Other forms of RPC may more often be blocked.
- Web services allow software and services from different companies and locations to be combined easily to provide an integrated service.

Disadvantages of Web Services

- Lacks of standard features, such as transactions, while comparing to other open standards, such as CORBA.
- Poor performance due to using of XML text-formats.

EXAMPLE OF WEB SERVICE REQUEST

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:q0="http://wsv.mci.com/ETECapability"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance">
  <SOAP-ENV:Body>
        <q0:BuildingID>rvc</q0:BuildingID>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

EXAMPLE OF WEB SERVICE RESPONSE

```
<?xml version="1.0" encoding="utf-8" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <soapenv:Header />
<soapenv:Body>
<ETESitesForBuilding xmlns="http://wsv.mci.com/ETECapability">
    <BuildingID>rvc</BuildingID>
    <OnNet>1</OnNet>
    <BuildingStatus>active</BuildingStatus>
    <BuildingServiceType>fiber</BuildingServiceType>
    <Sites>
        <SiteCode>80</SiteCode>
        <SiteCLLI>CHCGILTOH44</SiteCLLI>
        <PortTypes>
                <EquipmentType>ee4000</EquipmentType>
                <PortType>10/100</PortType>
                <PortType>GigE</PortType>
        </PortTypes>
        <Dedicated>0</Dedicated>
        <CSiteID xsi:nil="true" />
        <ColloRestrictedFlag>0</ColloRestrictedFlag>
        <SiteStatus>active</SiteStatus>
        <SiteType>node</SiteType>
    </Sites>
</ETESitesForBuilding>
</soapenv:Body>
</soapenv:Envelope>
```

SOAP PACKAGES

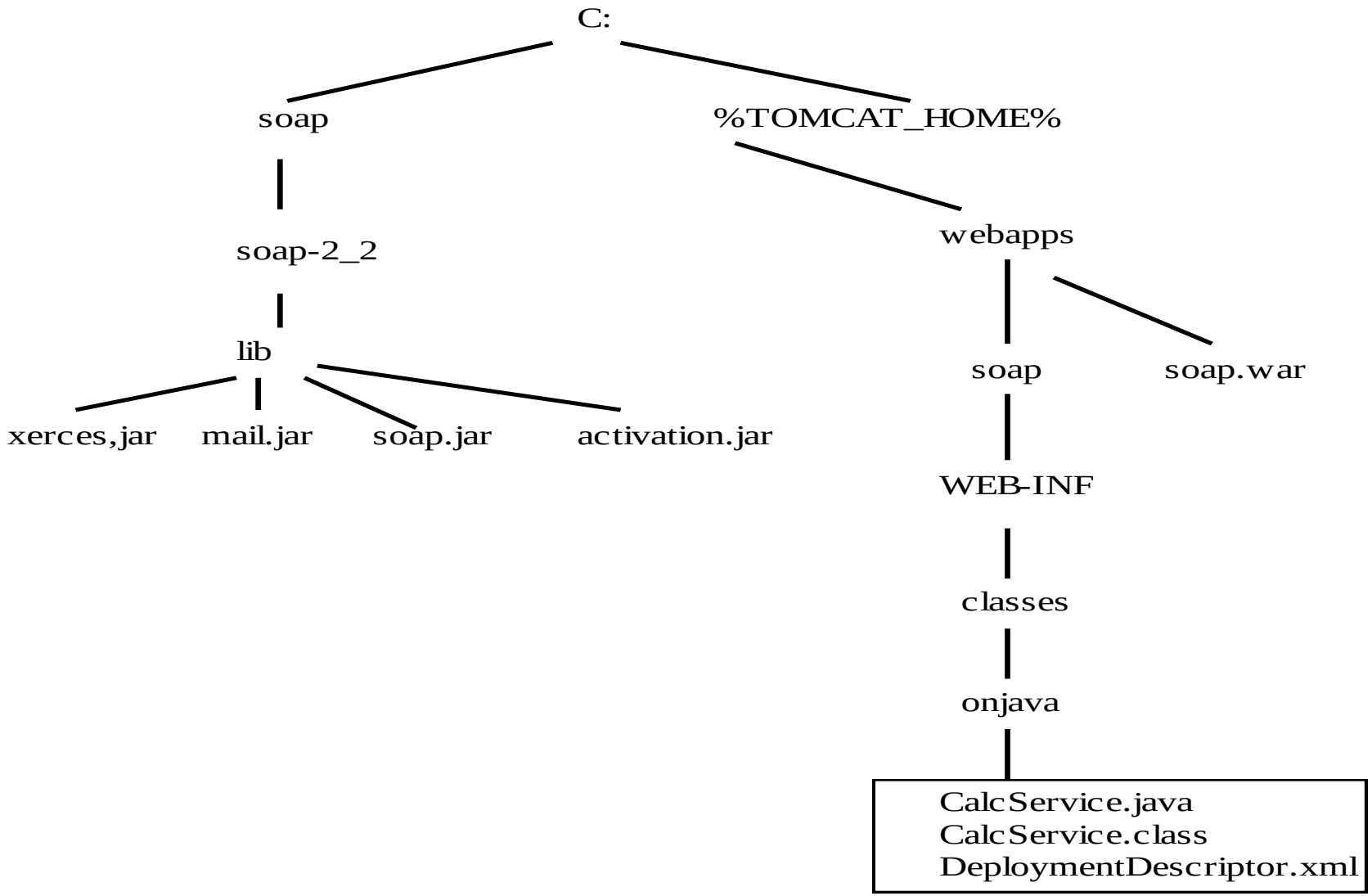
- Apache SOAP for Java
- Apache Axis for Java
- Idoox WASP for C++
- Microsoft SOAP Toolkit (part of the .net framework)
- SOAP::Lite for Perl

Apache SOAP

- Allows clients and services to be written in Java
- Part of the Apache-XML project (http://xml.apache.org/)
- SOAP package downloadable: http://xml.apache.org/dist/soap/
- Installation instruction:

  http://www.xmethods.com/gettingstarted/apache.html

APACHE SOAP INSTALLATION



CLASSPATH SETTING

```
set CLASSPATH=
    C:\soap\soap-2_2\lib\xerces.jar;
    C:\jdk1.3\bin;
    C:\jdk1.3\lib\tools.jar;
    C:\soap\soap-2_2\lib\mail.jar;
    C:\soap\soap-2_2\lib\soap.jar;
    C:\soap\soap-2_2\lib\activation.jar;
    C:\tomcat\lib\servlet.jar;
    .;
```

Writing a Client Application using Apache SOAP

Classpath
The CLASSPATH environment variable should have both the "soap.jar" and "xerces.jar" JAR files included. **Importing packages**
For basic SOAP method invocation:

```
// Required due to use of URL class , required by Call class
import java.net.*;

// Required due to use of Vector class
import java.util.*;

// Apache SOAP classes used by client
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;
```

SAMPLE SOAP SERVICE

Found at ***http://www.xmethods.com***

Weather - Temperature

Analyze WSDL | View RPC Profile | http://www.xmethods.net/sd/2001/TemperatureService.wsdl
XMethods ID 8
Service Owner: xmethods.net
Contact Email: support@xmethods.net
Service Home Page:
Description: Current temperature in a given U.S. zipcode region.
SOAP Implementation: Apache SOAP

RPC PROFILE FOR SERVICE

RPC Profile for Service "Weather - Temperature"

Method Name getTemp
Endpoint URL http://services.xmethods.net:80/soap/servlet/rpcrouter
SOAPAction
Method Namespace URI urn:xmethods-Temperature
Input Parameters zipcode **string**

Output Parameters return **float**

Client Program Samples

- See samples in client folder:
  - TempClient.java
  - StockQuoteClient.java
  - CurrencyClient.java

SOAP Data Types

- SOAP Data Types
  http://www.sdc.iup.edu/outreach/spring2002/webservices/datatypes.html
- Uses XML Schema data types
- **Primitive Types**

  string, boolean, decimal, float, double, duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth, hexBinary, base64Binary, anyURI, QName, NOTATION

SOAP DATA TYPES

Derived Types

- Simple types (derived from a single primitive type)
  * integer is derived from decimal
  * int (-2147483648 <= int <= 2147483647) is derived from long which is derived from integer
  * 5-digit zip code can be derived from int
  * may use regular expressions to specify derived types, such as ([A-Z]){2,3}-\d{5}

COMPLEX TYPE

Complex types (struct or array)

- Struct example

```
<instructor>
  <firstname xsi:type="xsd:string">Ed</firstname>
  <lastname xsi:type="xsd:string">Donley</lastname>
</instructor>
```

- Array example

```
<mathcourses xsi:type=
  "SOAP-ENC:Array" SOAP ENC:arrayType="se:string[3]">
  <se:string>10452C</se:string>
  <se:string>10454C</se:string>
  <se:string>11123T</se:string>
</mathcourses>
```

Creating Web Services (Server-side SOAP)

- O'Reilly Network: Using SOAP with Tomcat
  http://www.onjava.com/pub/a/onjava/2002/02/27/tomcat.htm
- Apache SOAP allows you to create and deploy a SOAP web service.
- Some jar files must be installed, and must be accessible by the CLASSPATH.

Algorithm:

1. Develop a class for providing the service.
2. Create a deployment descriptor in XML.
3. Deploy the service with the service manager.

The Apache SOAP Service Manager

- The service manager is itself implemented as a SOAP service.
- To see what services are deployed on your system:

  java org.apache.soap.server.ServiceManagerClient http://localhost:8080/soap/servlet/rpcrouter list

- To deploy a service:

  java org.apache.soap.server.ServiceManagerClient http://localhost:8080/soap/servlet/rpcrouter deploy foo.xml

Creating a SOAP Service

- A SOAP service can be just about any Java class that exposes public methods for invocation.
- The class does not need to know anything about SOAP, or even that it is being executed as a SOAP service.
- The method parameters of a SOAP service must be *serializable*. The available types that can be used as SOAP service parameters are shown next.

SOAP Service Parameter Types

- All Java primitive types and their corresponding wrapper classes
- Java arrays
- java.lang.String
- java.util.Date
- java.util.GregorianCalendar
- java.util.Vector
- java.util.Hashtable
- java.util.Map

SOAP Service Parameter Types

- java.math.BigDecimal
- javax.mail.internet.MimeBodyPart
- java.io.InputStream
- javax.activation.DataSource
- javax.activation.DataHandler
- org.apache.soap.util.xml.QName
- org.apache.soap.rpc.Parameter
- java.lang.Object (must be a JavaBean)

Sample SOAP Service Implementations

- TempService: A temperature service
- Exchange: currency exchange service and client

Sample SOAP service class

```
// A sample SOAP service class

package onjava;
public class CalcService {
  public int add (int p1, int p2) {
            return p1 + p2;   }

  public int subtract (int p1, int p2) {
    return p1 - p2;  }
}
```

DEPLOYMENT DESCRIPTOR

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
    id="urn:onjavaserver">
        <isd:provider type="java"
            scope="Application"
            methods="add subtract">
            <isd:java class="onjava.CalcService"/>
        </isd:provider> <isd:faultListener>org.apache.soap.server.DOMFaultListener
                                                    </isd:faultListener>
</isd:service>
```

SOURCES OF INFORMATION

- Programming Web Services with SOAP, by Snell et al, O'Reilly
- SOAP Tutorial, http://www.w3schools.com/soap/default.asp
- SoapRPC.com: Tutorials, **http://www.soaprpc.com/tutorials/--** Developer's Guide To Soap1.1
- DaveNet : XML-RPC for Newbies, http://davenet.userland.com/1998/07/14/xmlRpcForNewbies
- SoapRPC.com: Other resources, http://www.soaprpc.com/resources/

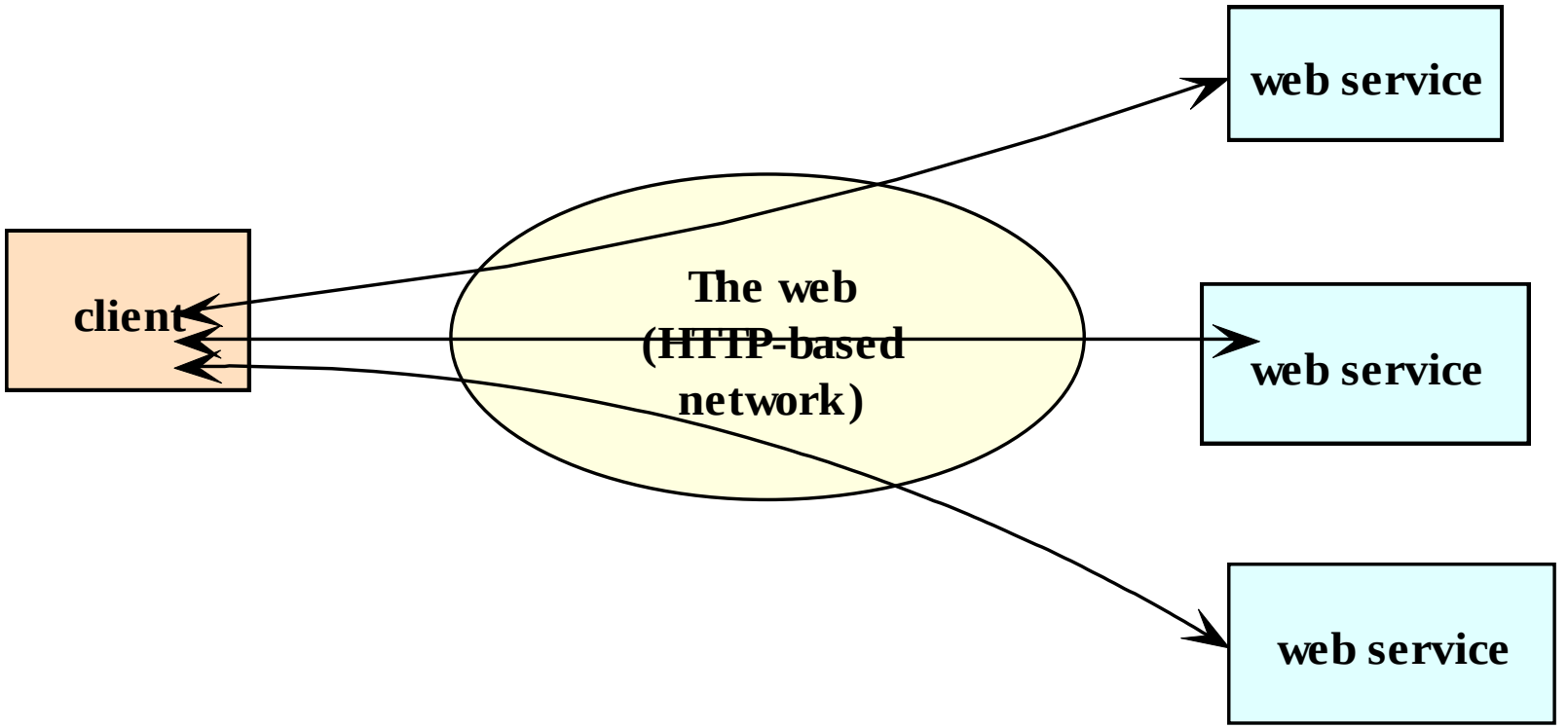
Web Services – Summary

- A **Web Service** is a software component that could be described via **WSDL** (Web Service Description Language) and is capable of being accessed via standard network protocols, such as **SOAP**, **over HTTP**

- **Web Services** define a **platform** and **language-**independent standard based on XML to communicate within distributed systems.

- **SOAP** is a lightweight protocol defined by the World Wide Web Consortium (W3C), which defines how to exchange information in a decentralized, distributed systems based on **XML**.

Web services – summary 2



Web services -summary 3

- Use of web service:
  - RPC-based: Java API for XML-based remote procedure calls (JAX-RPC) defines mappings between Java types and XML types
  - Message-based (SOA): SOAP messages
  - REpresentational State Transfer (REST): "*a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use*." -- Dr. Roy T. Fielding

SUMMARY

- **SOAP** is a protocol that makes use of HTTP requests and responses to effect remote method calls to web services.
- A SOAP method call is encoded in **XML** and is embedded in an HTTP request
- The return value of a method call is likewise embedded and encoded in an HTTP response
- A number of SOAP APIs are available for programming web services and client method calls. The Apache API was introduced.

2) Spring Framework Review

INTRODUCTION TO SPRING FRAMEWORK

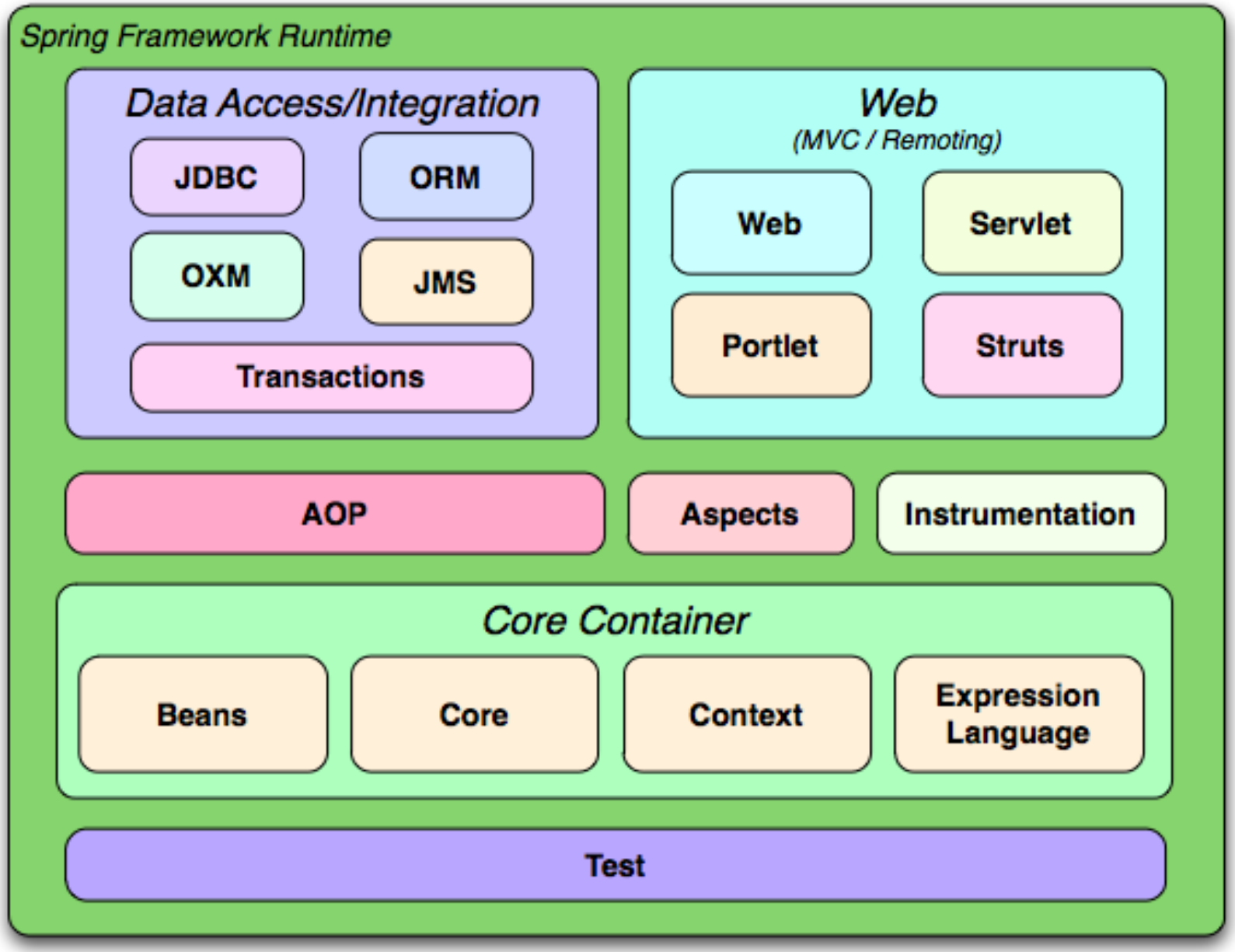
- An **open source** Java platform
- Initially written by Rod Johnson, first released under the Apache 2.0 license in June **2003**
- Spring is **lightweight**: the basic version = 2MB
- The core features can be used in any Java application
- But there are extensions for web applications on top of Java EE platform
- Spring targets to make J2EE development easier to use
- Promote good programming practice
- By enabling a POJO-based programming model

ABOUT SPRING

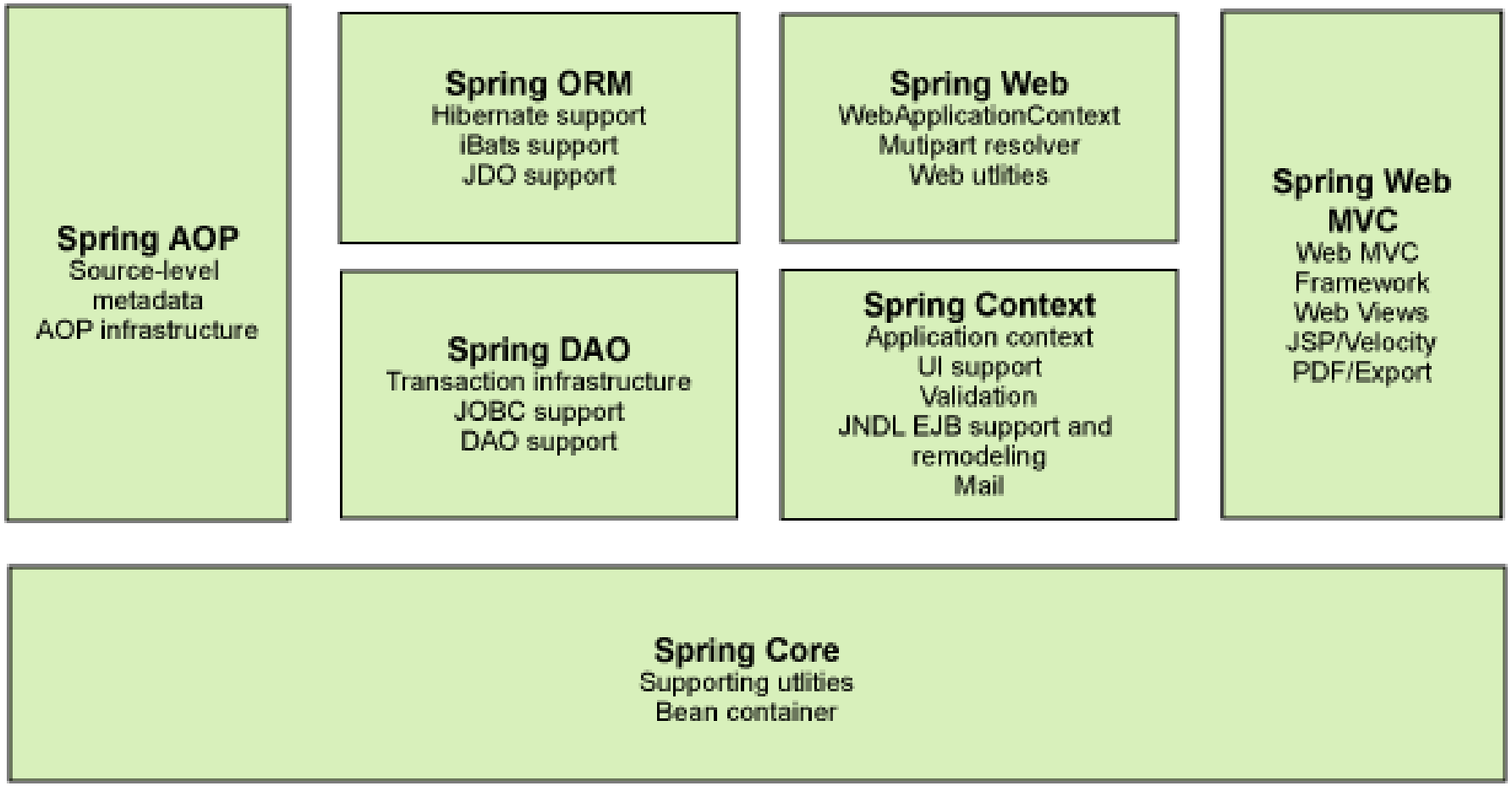
- Provides to create high performing, easily testable and
- reusable code.
- is organized in a modular fashion
- simplifies java development

SPRING MODULES

- Spring is modular
- Allowing you to choose which modules are applicable to you
- Provides about 20 modules


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
(MVC / Remoting)
Web
Servlet
Portlet
Struts
AOP
Aspects
Instrumentation
Core Container
Beans
Core
Context
Expression Language
Test

SPRING MODULES (CONT'D)



Two Key Components of Spring

- Dependency Injection (DI)
- Aspect Oriented Programming (AOP)

Dependency injection (DI)

- Inversion of Control (IoC)
  - Object coupling is bound at run time
  - The binding process is achieved through dependency injection
  - Some argue that a “service locator” also provides IoC
- Dependency injection is a software design pattern
- DI allows the removal of hard-coded dependencies
- Makes it possible to change them at run-time
- E.g.
  - as a simple way to load plugins dynamically
  - to choose stubs or mock objects in test environments vs. real objects in production environments.

Dependency Injection (cont'd)

- application classes should be as independent as possible
  - To increase the possibility to reuse these classes
  - and to test them independently
- **Dependency**: an association between two classes
  - E.g., class A is dependent on class B
- **Injection**: class B will get injected into class A by the IoC
- Dependency injection
  - in the way of passing parameters to the constructor
  - or by post-construction using setter methods

Aspect Oriented Programming (AOP)

- **cross-cutting concerns**
  - The functions that span multiple points of an application
- cross-cutting concerns are conceptually separate from the application's business logic
  - E.g., logging, declarative transactions, security, and caching
- The key unit of modularity
  - in OOP: the class
  - in AOP: the aspect.
- DI helps you decouple application objects from each other
- AOP helps you decouple cross-cutting concerns from the objects that they affect

SPRING VS. EJB

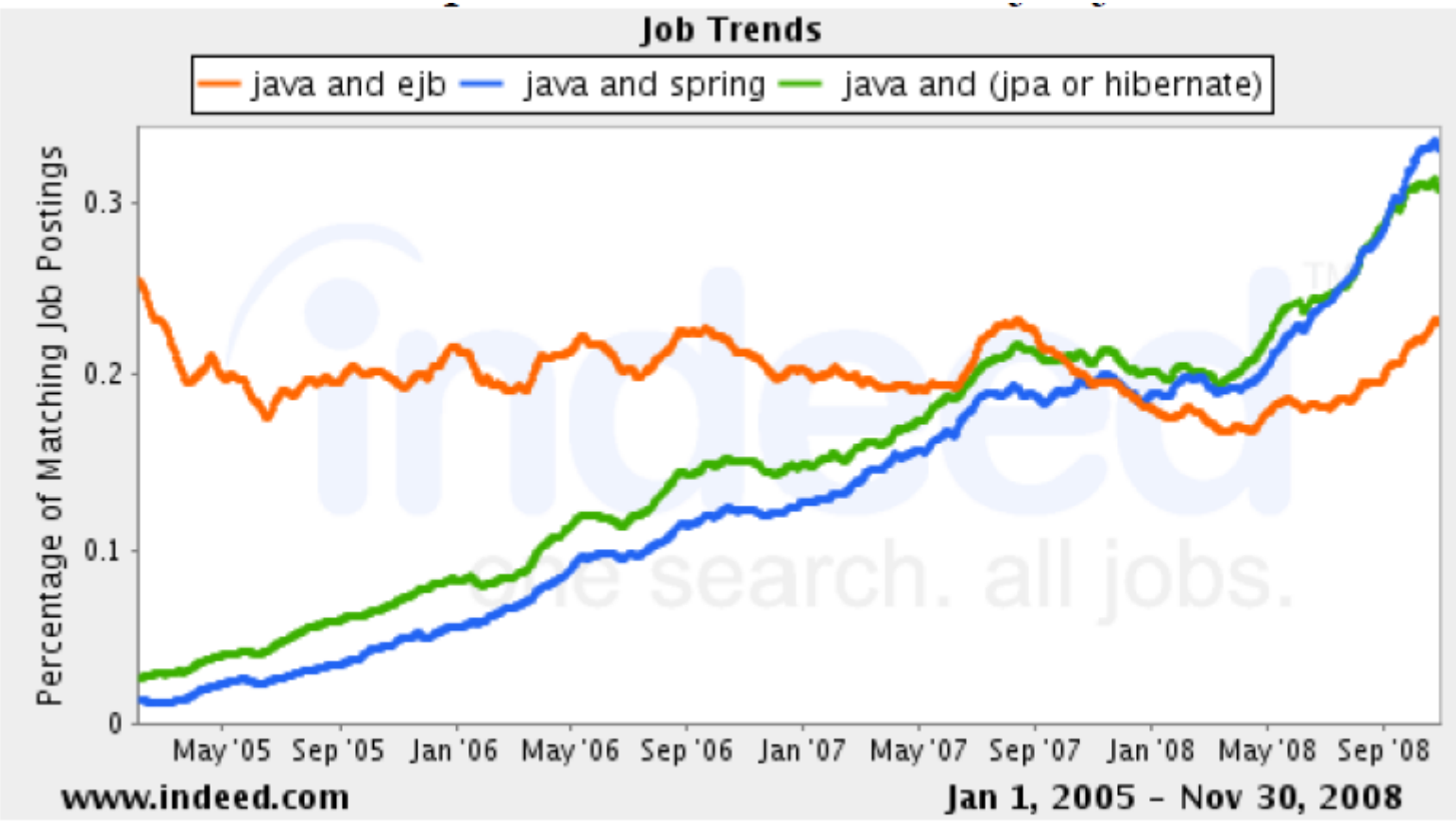
- Spring is light-weight
- And portable
- Offers many features of EJBs
- But simpler and easier
- No need to application server
- Core features: non-web applications
- Spring integrates many existing frameworks
  - ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, other view technologies

Spring vs. EJB (cont'd)

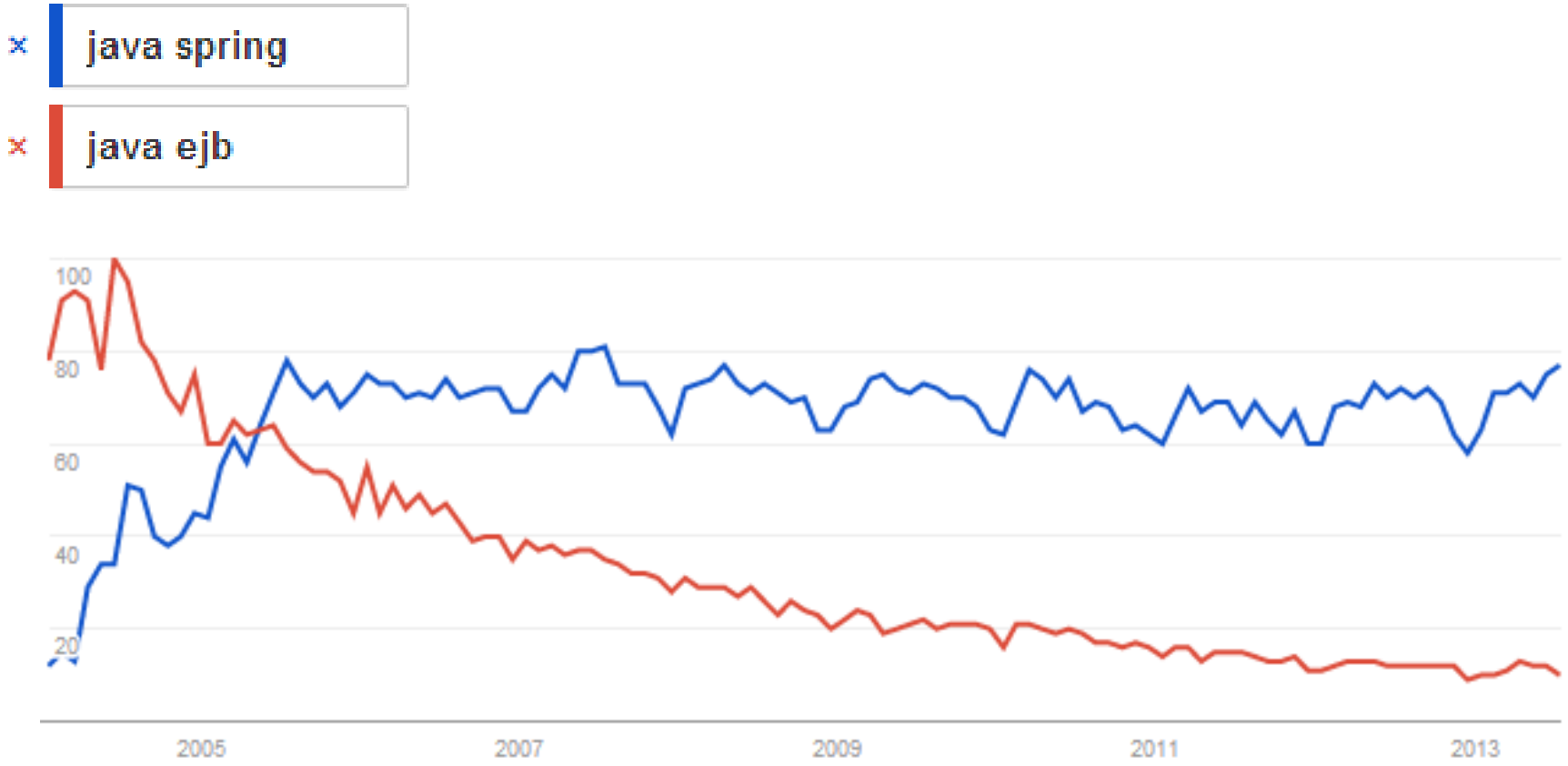
- Testing is simpler
  - Testing in non-web contexts
  - Injection of mock objects
- Lightweight IoC containers
- a consistent transaction management interface
- scale down to a local transaction
  - using a single database
- scale up to global transactions
  - using JTA

What is Spring?

- Spring is the most popular application development framework for enterprise Java



TRENDS: SPRING VS. EJB



SPRING SETUP

- Spring is some jar files
  - Download from: http://www.springsource.org/spring-community-download
- And dependencies to some others
  - Apache Common Logging API
    - Download from: http://commons.apache.org/logging/
  - AspectJ
    - aspectjrt.jar
    - aspectjweaver.jar
    - aspectj.jar
    - Aopalliance.jar

Spring – Hello World

- Create your java project
  - Simple application
  - Web application
- Add required Jar libraries
- Create source files
  - Class of beans
- Create bean configuration file (XML)
- Retrieve beans

SOURCE: BEAN CLASSES

```
package com.tutorialspoint;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

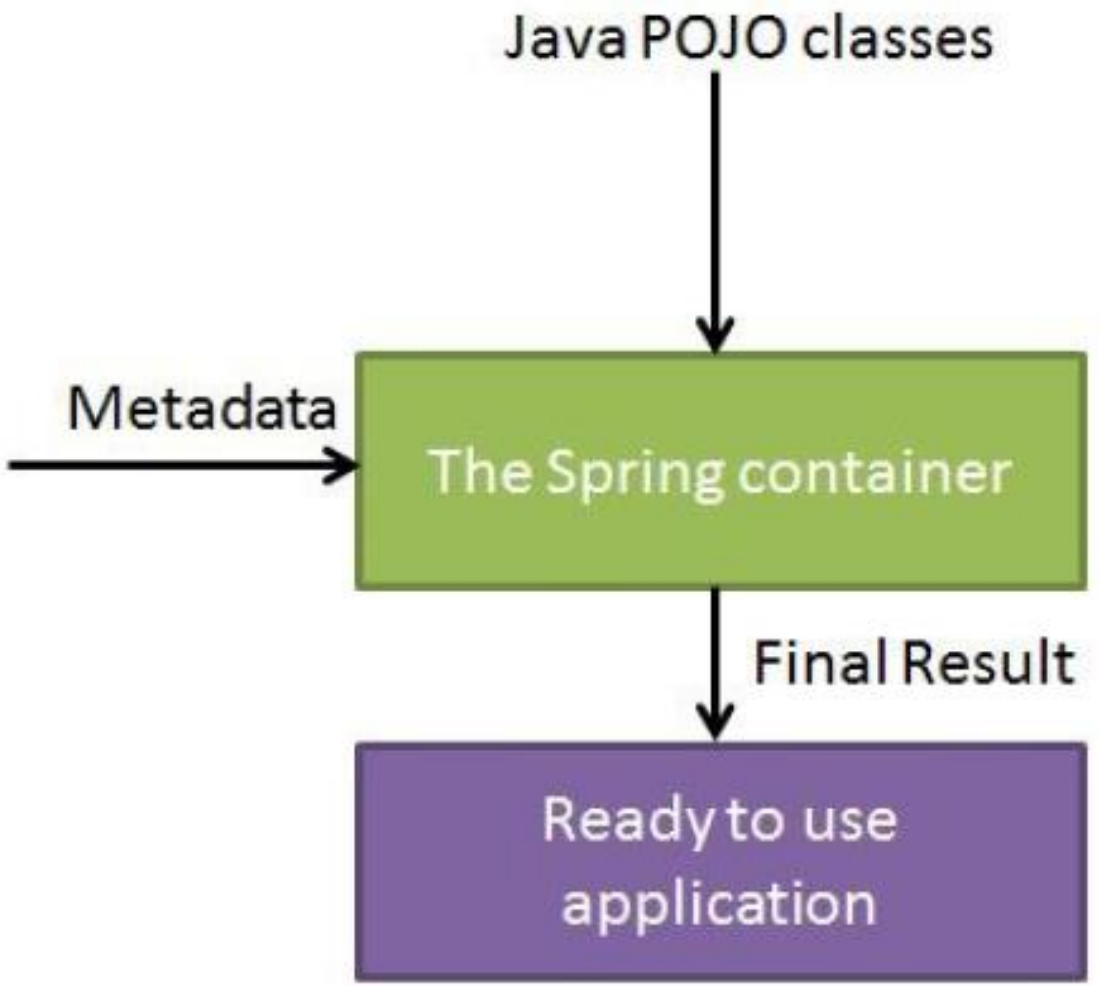
BEAN CONFIGURATION FILE

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
      <property name="message" value="Hello World!"/>
   </bean>

</beans>
```

RETRIEVE BEANS

```
public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
            new ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

      obj.getMessage();
   }
}
```

SPRING CONTAINER



SPRING CONFIGURATION METADATA

- XML based configuration file.
- Annotation-based configuration
- Java-based configuration

Spring IOC Containers (Cont'd)

- Spring BeanFactory Container
  - the simplest container providing basic support for DI
  - for the purposes of backward compatibility
  - E.g.
    - XmlBeanFactory factory = new XmlBeanFactory (new ClassPathResource("Beans.xml"));
- Spring ApplicationContext Container
  - This container adds more enterprise-specific functionality
  - E.g.
    - ApplicationContext context = new FileSystemXmlApplicationContext ("C:/Beans.xml");

Spring Bean Definition

- class
- name (id)
- scope
- constructor-arg
- properties
- autowiring mode
- lazy-initialization mode
- initialization method
  - A callback, invoked just after all properties on the bean have been set
  - For the sake of post-processing the bean creation
- destruction method
  - A callback, invoked when the container is destroyed.

SPRING BEAN SCOPES

- Common Scopes
  - singleton
  - prototype
    - container creates new bean instance of the object every time a request for that specific bean is made.
- Web-aware applications
  - request
  - session
  - global-session

```
<bean id="..." class="..." scope="singleton">
    <!-- collaborators and configuration for this bean go here -->
</bean>
```

HINT

- Think about it:
- Spring bean container is not a web framework
- How a servlet container may prepare web-contexts (request, session and global-session) for beans?
- Listeners

NOTE

- Singleton pattern:
  - singleton per class-loader
- Singleton in Spring:
  - singleton per bean definition, per container

DEFAULT INITIALIZATION AND DESTROY METHODS

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
    default-init-method="init"
    default-destroy-method="destroy">
```

BEAN EXAMPLE

```
<!-- A simple bean definition -->
 <bean id="..." class="...">
<!-- collaborators and configuration for this bean go here -->
 </bean>

<!-- A bean definition with lazy init set on -->
 <bean id="..." class="..." lazy-init="true">
<!-- collaborators and configuration for this bean go here -->
 </bean>

 <!-- A bean definition with initialization method -->
 <bean id="..." class="..." init-method="...">
<!-- collaborators and configuration for this bean go here -->
 </bean>

<!-- A bean definition with destruction method -->
 <bean id="..." class="..." destroy-method="...">
<!-- collaborators and configuration for this bean go here -->
 </bean>
```

Bean Definition Inheritance

- Parent property for beans

```
<bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
    <property name="message1" value="Hello World!"/>
    <property name="message2" value="Hello Second World!"/>
</bean>

<bean id="helloIndia" class="com.tutorialspoint.HelloIndia"
    parent="helloWorld">
    <property name="message1" value="Hello India!"/>
    <property name="message3" value="Namaste India!"/>
</bean>
```

Bean Definition Inheritance (cont'd)

- Abstract beans
  - Bean Definition Template

```
<bean id="beanTeamplate" abstract="true">
    <property name="message1" value="Hello World!"/>
    <property name="message2" value="Hello Second World!"/>
    <property name="message3" value="Namaste India!"/>
</bean>

<bean id="helloIndia" class="com.tutorialspoint.HelloIndia"
    parent="beanTeamplate">
    <property name="message1" value="Hello India!"/>
    <property name="message3" value="Namaste India!"/>
</bean>
```

NOTE

- Relation of parent and abstract to OOP concepts?

- Inheritance in Spring is different from OOP inheritance
  - Parent has a different meaning
- Abstract in Spring is different from OOP abstract

- Parent and abstract is defined for beans (objects) not for classes

DEPENDENCY INJECTION

- Every java based application has a few objects that work together
- In a complex Java application, application classes should be as independent as possible
- To increase the possibility to reuse these classes
- and to test them independently
- Dependency Injection (or sometime called **wiring**) helps in gluing these classes together
- and same time keeping them independent.

Example of a dependency:

- What is wrong with this code?
- we have created a dependency between the TextEditor and the SpellChecker concrete class

```
public class TextEditor {
    private SpellChecker spellChecker;
    public TextEditor() {
        spellChecker = new SpellChecker();
    }
}
```

SOLUTION

- inversion of control
- like this:

```
public class TextEditor {
    private SpellChecker spellChecker;
    public TextEditor(SpellChecker spellChecker) {
        this.spellChecker = spellChecker;
    }
}
```

Dependency injection types

- Constructor-based dependency injection
- Setter-based dependency injection

CONSTRUCTOR-BASED DEPENDENCY INJECTION

```
<!-- Definition for textEditor bean -->
<bean id="textEditor" class="com.tutorialspoint.TextEditor">
   <constructor-arg ref="spellChecker"/>
</bean>

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
</bean>
```

Constructor-based Dependency Injection (2)

```
<beans>
   <bean id="foo" class="x.y.Foo">
      <constructor-arg ref="bar"/>
      <constructor-arg ref="baz"/>
   </bean>

   <bean id="bar" class="x.y.Bar"/>
   <bean id="baz" class="x.y.Baz"/>
</beans>
```

Constructor-based Dependency Injection (3)

```
<beans>

   <bean id="exampleBean" class="examples.ExampleBean">
      <constructor-arg type="int" value="2001"/>
      <constructor-arg type="java.lang.String" value="Zara"/>
   </bean>

</beans>
```

Constructor-based Dependency Injection (4)

- Perhap

```
<beans>

   <bean id="exampleBean" class="examples.ExampleBean">
      <constructor-arg index="0" value="2001"/>
      <constructor-arg index="1" value="Zara"/>
   </bean>

</beans>
```

SETTER-BASED

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"

   <!-- Definition for textEditor bean -->
   <bean id="textEditor" class="com.tutorialspoint.TextEditor">
      <property name="spellChecker" ref="spellChecker"/>
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
   </bean>

</beans>
```

XML CONFIGURATION USING P-NAMESPACE

```
<bean id="john-classic" class="com.example.Person">
   <property name="name" value="John Doe"/>
   <property name="spouse" ref="jane"/>
</bean>

<bean name="jane" class="com.example.Person">
   <property name="name" value="John Doe"/>
</bean>
```

```
<bean id="john-classic" class="com.example.Person"
   p:name="John Doe"
   p:spouse-ref="jane"/>
</bean>

<bean name="jane" class="com.example.Person"
   p:name="John Doe"/>
</bean>
```

INJECTING COLLECTIONS (LIST)

```
<property name="addressList">
  <list>
     <value>INDIA</value>
     <value>Pakistan</value>
     <value>USA</value>
     <value>USA</value>
  </list>
</property>
```

INJECTING COLLECTIONS (SET)

```
<property name="addressSet">
   <set>
      <value>INDIA</value>
      <value>Pakistan</value>
      <value>USA</value>
      <value>USA</value>
   </set>
 </property>
```

INJECTING COLLECTIONS (MAP)

```
<property name="addressMap">
   <map>
      <entry key="1" value="NDIA"/>
      <entry key="2" value="Pakistan"/>
      <entry key="3" value="USA"/>
      <entry key="4" value="USA"/>
   </map>
 </property>
```

INJECTING COLLECTIONS (PROPERTIES)

```
class Properties extends Hashtable<Object,Object>
```

```
<property name="addressProp">
   <props>
      <prop key="one">INDIA</prop>
      <prop key="two">Pakistan</prop>
      <prop key="three">USA</prop>
      <prop key="four">USA</prop>
   </props>
 </property>
```

INJECTING REFERENCES IN COLLECTIONS

```
<property name="addressSet">
   <set>
      <ref bean="address1"/>
      <ref bean="address2"/>
      <value>Pakistan</value>
   </set>
 </property>
```

```
<property name="addressMap">
   <map>
      <entry key="one" value="NDIA"/>
      <entry key ="two" value-ref="address1"/>
      <entry key ="three" value-ref="address2"/>
   </map>
 </property>
```

INNER BEANS

- Defined within the scope of another bean
- a <bean/> element inside the **<property/>** or **<constructor-arg/>** elements
- Similar to inner classes, but:
- It is all about the scope of beans, nothing about OOP

```
<bean id="outerBean" class="...">
    <property name="target">
        <bean id="innerBean" class="..."/>
    </property>
</bean>
```

Injecting Null and Empty String Values

```
<bean id="..." class="exampleBean">
   <property name="email" value=""/>
</bean>
```

```
<bean id="..." class="exampleBean">
   <property name="email"><null/></property>
</bean>
```

AUTO-WIRING

- We can **autowire** relationships between collaborating beans
- Without using <constructor-arg> or <property> elements
- Decreases the amount of XML configuration you write
- Use the **autowire** attribute of the <bean/> element
  - no
  - byName
  - byType
  - constructor

AUTO-WIRING : BYNAME

- Autowiring by property name.
- Spring looks at the properties of the beans on which *autowire* attribute is set to *byName*
- Tries to match and wire its properties with the beans defined by the same names
- If matches are not found, it will throw exceptions

```
public class TextEditor {
   private SpellChecker spellChecker;
   private String name;
```

```
<bean id="textEditor" class="com.t
   autowire="byName">
   <property name="name" value="Ge
</bean>

<!-- Definition for spellChecker b
<bean id="spellChecker" class="com
</bean>
```

AUTO-WIRING : BYTYPE

- Autowiring by property datatype
- Spring looks at the properties of the beans on which ***autowire*** attribute is set to ***byType***
- Tries to match and wire a property if its **type** matches with **exactly one** of the beans
- If more than one such beans exists, a fatal exception is thrown

```
<bean id="textEditor" class="com.t
   autowire="byType">
   <property name="name" value="Ge
</bean>
```

Auto-Wiring : Constructor

- Similar to byType
- but type applies to constructor arguments
- If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised

```
<bean id="textEditor" class="com.tut
   autowire="constructor">
   <constructor-arg value="Generic Tex
</bean>
```

Limitations with Autowiring

Limitations	Description
Overriding possibility	You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.
Primitive data types	You cannot autowire so-called simple properties such as primitives, Strings, and Classes.
Confusing nature	Autowiring is less exact than explicit wiring, so if possible prefer using explict wiring.

ANNOTATION BASED CONFIGURATION

- Since Spring 2.5
- to configure the dependency injection using **annotations**
- instead of using XML to describe a bean wiring
- move the bean configuration into the component class itself by using annotations

Annotation-based configuration examples

```
@Required
public void setAge(Integer age) {
    this.age = age;
}
```

```
@Autowired
public void setSpellChecker( SpellChecker spellChecker ){
    this.spellChecker = spellChecker;
}
```

```
@Autowired
private SpellChecker spellChecker;
```

```
@Autowired
public TextEditor(SpellChecker spellChecker){
    System.out.println("Inside TextEditor co
    this.spellChecker = spellChecker;
}
```

Spring Java Based Configuration

- To write most of your Spring configuration without XML
- But with few Java-based annotations
- **@Configuration:** the class can be used by the Spring IoC container as a source of bean definitions
  - Instead of writing xml files
- **@Bean:** the annotated method will return an object that should be registered as a bean in the Spring application context
  - Instead of writing <bean> tags

EXAMPLE

```
@Configuration
public class HelloWorldConfig {

    @Bean
    public HelloWorld helloWorld(){
        return new HelloWorld();
    }
}
```

- Above code will be equivalent to the following XML configuration:

```
<beans>
   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld" />
</beans>
```

Creating the Context

```
AnnotationConfigApplicationContext ctx =
new AnnotationConfigApplicationContext();

ctx.register(AppConfig.class, OtherConfig.class);
ctx.register(AdditionalConfig.class);
ctx.refresh();

MyService myService = ctx.getBean(MyService.class);
myService.doStuff();
```

INJECTING BEAN DEPENDENCIES

```java
@Configuration
public class TextEditorConfig {

    @Bean
    public TextEditor textEditor(){
        return new TextEditor( spellChecker() );
    }

    @Bean
    public SpellChecker spellChecker(){
        return new SpellChecker( );
    }
```

```java
@Configuration
public class AppConfig {
    @Bean
    public Foo foo() {
        return new Foo(bar());
    }
    @Bean
    public Bar bar() {
        return new Bar();
```

AOP WITH SPRING FRAMEWORK

- Aspect oriented programming
- cross-cutting concerns
  - logging, auditing, declarative transactions, security, and caching
- The key unit of modularity in OOP is the class
- in AOP the unit of modularity is the aspect
- Dependency Injection helps you decouple your application objects from each other
- AOP helps you decouple cross-cutting concerns from the objects that they affect
- AOP is like triggers
- Spring AOP module provides interceptors
  - for example, when a method is executed
  - you can add extra functionality before or after the method execution

AOP TERMINOLOGY

- Aspect
  - A module which has APIs providing cross-cutting requirements
  - E.g., a logging module
- Join point
  - the actual place in the application where an action will be taken
  - E.g., before deposit() method
- Advice
  - the actual action to be taken
- Point-cut
  - a set of one or more join-points where an advice should be executed
  - Expressed using expressions or patterns

AOP Terminology (2)

- Introduction
  - to add new methods or attributes to existing classes
- Target object
  - The object being advised by one or more aspects
  - will always be a proxied object
  - Also referred to as the advised object
- Weaving
  - The process of linking aspects with other application types or objects, to create an advised object
  - This can be done at compile time, load time, or at runtime.

Types of Advice

- before
  - before the a method execution
- after
- around
  - before and after
- after-returning
  - only if method completes successfully
- after-throwing
  - only if method exits by throwing an exception

XML SCHEMA BASED AOP WITH SPRING

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

    <!-- bean definition & AOP specific configuration -->

</beans>
```

AOP NEEDED JARS:

- aspectjrt.jar
- aspectjweaver.jar
- aspectj.jar
- aopalliance.jar

EXAMPLE: AOP IN SPRING

```
<aop:config>
    <aop:aspect id="carAspect" ref="car">
        <aop:pointcut id="businessService"
            expression="within(*)"  />
        <aop:before pointcut-ref="businessService"
            method="logBefore" />
        <aop:after pointcut-ref="businessService"
            method="logAfter" />
        <aop:after-returning pointcut-ref="businessService"
            returning="retVal" method="logRet" />
        <aop:after-throwing pointcut-ref="businessService"
            throwing="ex" method="logEx" />
        <aop:around pointcut-ref="businessService"
            method="log" />
    </aop:aspect>
</aop:config>
```

```
<bean id="car" class="ir.navaco.Car">
    <property name="name" value="peikan" />
    <property name="wheel" ref="wheel" />
```

ANNOTATION-BASED ASPECTS

- enabled by including ***<aop:aspectj-autoproxy/>*** inside XML Schema-based configuration file

```
@Pointcut("execution(* com.xyz.myapp.service.*.*(..))") //
expression
private void businessService() {}  // signature
```

- The actual body of the businessService() is irrelevant
  - should be empty

```
@Before("businessService()")
public void doBeforeTask(){
  ...
}
```

```
@After("businessService()")
public void doAfterTask(){
  ...
}
```

SPRING JDBC FRAMEWORK

- Simplifies JDBC programming
- Takes care of all the low-level details
  - opening the connection
  - prepare and execute the SQL statement
  - process exceptions
  - handle transactions
  - finally close the connection

DATASOURCE

```
<bean id="dataSource"
 class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName"
        value="com.mysql.jdbc.Driver" />
      <property name="url"
        value="jdbc:mysql://localhost:3306/test" />
      <property name="username" value="root" />
      <property name="password" value="123" />
</bean>
```

JDBCTEMPLATE CLASS

- executes SQL queries
- update statements
- stored procedure calls
- performs iteration over ResultSets and extraction of returned parameter values

- Instances of the JdbcTemplate class are **threadsafe**
- single instance of a *JdbcTemplate is sufficient*
- safely inject this shared reference into multiple DAOs

JDBCTEMPLATE EXAMPLES

```
String SQL = "select count(*) from Student";
int rowCount = jdbcTemplateObject.queryForInt( SQL
);
```

```
String SQL = "select age from Student where id = ?";
int age = jdbcTemplateObject.queryForInt(SQL,
        new Object[]{10});
```

```
String SQL = "select name from Student where id = ?";
String name = jdbcTemplateObject.queryForObject(
        SQL, new Object[]{10}, String.class);
```

JDBCTEMPLATE EXAMPLES (CONT'D)

String SQL = "select * from Student where id = ?";

Student student = jdbcTemplateObject.**queryForObject**(

```
public class StudentMapper implements RowMapper<Student> {
   public Student mapRow(ResultSet rs, int rowNum) throws SQ
{
      Student student = new Student();
      student.setID(rs.getInt("id"));
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));
      return student;
   }
}
```

Data Access Object (DAO)

- DAOs provide a means to read/write data to the database

```
<bean id="studentJDBCTemplate"
   class="com.tutorialspoint.StudentJDBCTemplate">
   <property name="dataSource"  ref="dataSource" />
</bean>
```

```
public class StudentJDBCTemplate implements StudentDAO {
   private DataSource dataSource;
   private JdbcTemplate jdbcTemplateObject;

   public void setDataSource(DataSource dataSource) {
      this.dataSource = dataSource;
      this.jdbcTemplateObject = new JdbcTemplate(dataSource);
   }
```

STUDENT DAO (CONT'D)

```
public Student getStudent(Integer id) {
   String SQL = "select * from Student where id = ?";
   Student student = jdbcTemplateObject.queryForObject(SQL,
                     new Object[]{id}, new StudentMapper());
   return student;
}
```

```
public void update(Integer id, Integer age){
   String SQL = "update Student set age = ? where id = ?";
   jdbcTemplateObject.update(SQL, age, id);
   System.out.println("Updated Record with ID = " + id );
   return;
}
```

SPRING TRANSACTION MANAGEMENT

- A **database transaction** is a sequence of actions that are treated as a single unit of work
- four key properties described as **ACID**
  - **Atomicity**
  - **Consistency**
  - **Isolation**
  - **Durability**
- Local vs. Global Transactions
- Programmatic vs. Declarative
- Declarative: annotations or XML based configuration
  - Declarative transaction management is preferable
  - Spring supports declarative transaction management through AOP

PROGRAMMATIC TRANSACTION MANAGEMENT

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.Drive
   <property name="driverClassName" value="com.my
   <property name="url" value="jdbc:mysql://local
   <property name="username" value="root"/>
   <property name="password" value="password"/>
</bean>
```

```
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager
">
   <property name="dataSource"  ref="dataSource" />
</bean>
```

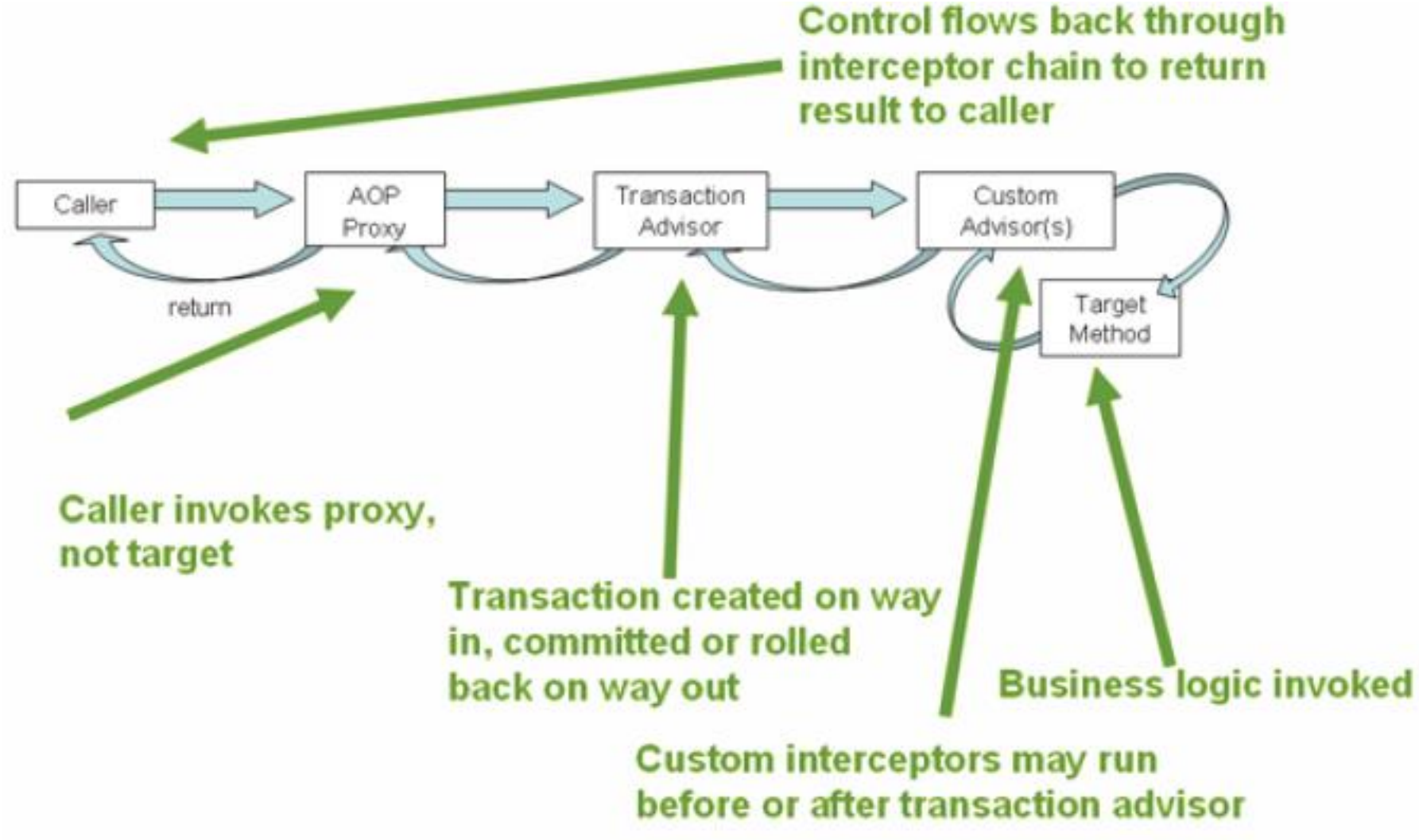
```
<bean id="studentJDBCTemplate"
   class="com.tutorialspoint.StudentJDBCTemplate">
   <property name="dataSource"  ref="dataSource" />
   <property name="transactionManager"  ref="transactionManager" />
</bean>
```

```
   public void create(String name, Integer age, Integer marks, Integer
year){

      TransactionDefinition def = new DefaultTransactionDefinition();
      TransactionStatus status =
transactionManager.getTransaction(def);

      try {
         String SQL1 = "insert into Student (name, age) values (?, ?)";
         jdbcTemplateObject.update( SQL1, name, age);

         // Get the latest student id to be used in Marks table
         String SQL2 = "select max(id) from Student";
         int sid = jdbcTemplateObject.queryForInt( SQL2 );

         String SQL3 = "insert into Marks(sid, marks, year) " +
                       "values (?, ?, ?)";
         jdbcTemplateObject.update( SQL3, sid, marks, year);

         System.out.println("Created Name = " + name + ", Age = " +
age);
         transactionManager.commit(status);
      } catch (DataAccessException e) {
         System.out.println("Error in creating record, rolling back");
         transactionManager.rollback(status);
         throw e;
      }
      return;
   }
```

Declarative Transaction Management

- Use Advices which creates a transaction-handling advice
  - **<tx:advice** /> tag
- Define a pointcut that matches all methods we wish to make transactional
- Reference the transactional advice
- If a method name has been included in the transactional configuration then created advice will begin the transaction before calling the method
- Target method will be executed in a *try / catch* block.
- If the method finishes normally, the AOP advice **commit**s the transaction successfully otherwise it performs a **rollback**

Spring's Declarative Transaction Management



BEAN CONFIGURATION FOR TRANSACTIONS

```
<tx:advice id="txAdvice"  transaction-manager="transactionManager">
   <tx:attributes>
   <tx:method name="create"/>
   </tx:attributes>
</tx:advice>
```

```
    <aop:config>
     <aop:pointcut id="createOperation"
     expression="execution(*
com.tutorialspoint.StudentJDBCTemplate.create(..))"/>
       <aop:advisor advice-ref="txAdvice" pointcut-
ref="createOperation"/>
   </aop:config>
```

USING @TRANSACTIONAL

```
<!-- enable the configuration of transactional behavior based on annotations -->
<tx:annotation-driven transaction-manager="txManager"/>

<!-- a PlatformTransactionManager is still required -->
<bean id="txManager"
class="org.springframework.jdbc.datasource.
  DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
</bean>
```

- You can use **@Transactional** for methods or Classes

SPRING FRAMEWORK ALTERNATIVES

- Google Guice
  - provides support for dependency injection using annotations
- Java EE 6 new standards
  - Contexts and Dependency Injection (CDI)
  - specified by JSR 299
  - JBoss Weld is its reference implementation
- **CDI**: a JCP specification included in Java EE 6 (JSR 299)
- **Weld**: the reference implementation of CDI
- **Seam 3**: a set of modules which extend CDI to provide functionality beyond that offered by Java EE 6
- Red Hat➔JBoss➔Seam Framwork ➔ Weld

DECISION POINT

- Standards or (non-standard) frameworks?
- The benefits of standard technologies

- Standards
  - JSP, JSF, JPA, …
- Non-standard technologies
  - Spring, GWT, …

JavaEE6's Advocates Say:

- The Age of Frameworks is Over
- The J2EE platform had its shortcoming
  - Spring quickly became the most prominent framework for easily building enterprise-ready applications
  - Helping developers avoid many of the pitfalls of the old enterprise Java platform
- But J2EE represents the past, and Java EE 6 represents the future!
- Java EE 6 promises us the ability to go beyond frameworks.

JavaEE6's Advocates Say: (Cont'd)

- Frameworks like Spring are really just a bridge between the mistakes of the J2EE past and the success of the Java EE 6 future.
- Frameworks are out, and extensions to the Java EE 6 platform are in.
- Now is the time to start looking past Spring, and looking forward to Seam and Weld and CDI technologies.
- And to start shifting into a world of extensions, and moving away from frameworks,

But, Spring's advocates say:

- In fact, Spring is more than a DI framework
- Perhaps, it is still the most popular application development framework for enterprise Java
- Many extensions for easy integration with other frameworks

Review a real application

- Beans for data access (DAO)
  - E.g., Hibernate or JDBC classes
- Beans for business logic
- Beans for presentation
  - E.g., GWT or JSF pages

- How are the dependencies?
- How are the beans retrieved from bean container?
- How many beans:
  - Data-sources?
  - DAOs?
  - Logic classes?
  - Presentation classes?

3) Spring Hibernate

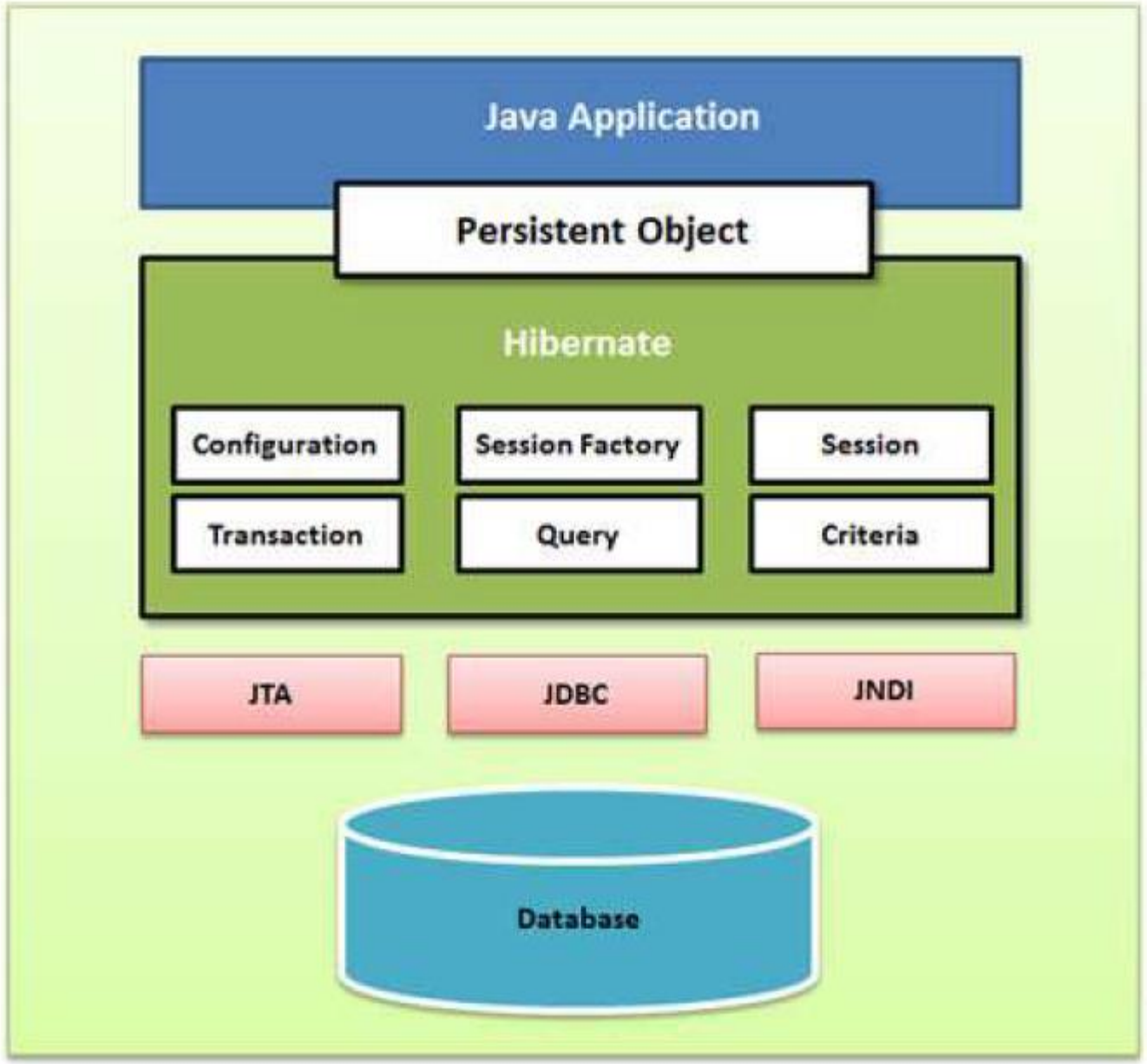
HIBERNATE

- http://www.hibernate.org/
  - Hosted by the JBoss Community
- Open Source
- created by Gavin King in 2001
- Hibernate is a high-performance Object/Relational persistence and query service
- Mapping Java objects to relational databases
- Today
  - Collection of projects/frameworks for extended use of POJO (plain old Java objects)

ROLE OF HIBERNATE

- Hibernate sits between traditional Java objects and database server
- To handle all the work in persisting those objects
- Based on the appropriate O/R mechanisms and patterns

HIBERNATE POSITION



OBJECT-RELATIONAL MAPPING

- It is a programming technique for converting object-type data of an object oriented programming language into database tables.
- Hibernate is used convert object data in JAVA to relational database tables.

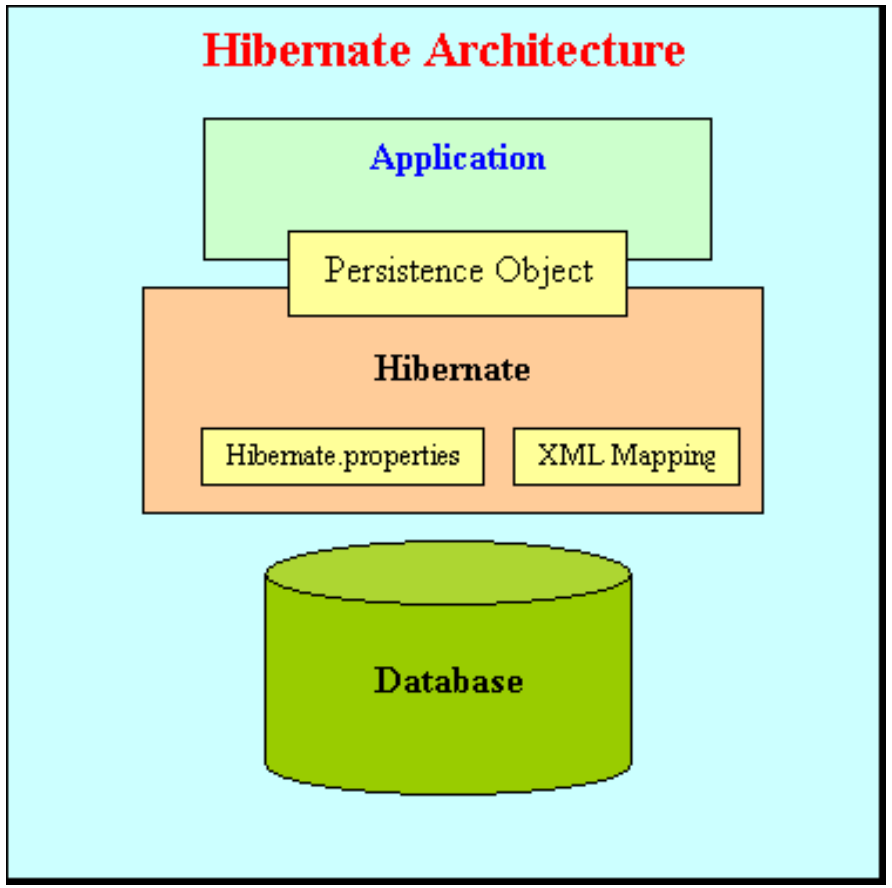
What is Hibernate?

- It is open source object-relational mapping (ORM) for Java.
- Hibernate is responsible for making data persistent by storing it in a database.

Why Hibernate and not JDBC?

- JDBC maps Java classes to database tables (and from Java data types to SQL data types)
- Hibernate automatically generates the SQL queries.
- Hibernate provides data query and retrieval facilities and can significantly reduce development time as more time is required to manually handle data in SQL and JDBC.
- It makes an application portable to all SQL databases.

ARCHITECTURE



Hibernate sits between your code and the database

Maps persistent objects to tables in the database

HIBERNATE

- **Configuration Object:**
  - The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. The Configuration object provides two keys components:
  - **Database Connection:** This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
  - **Class Mapping Setup**
    This component creates the connection between the Java classes and database tables.
- **SessionFactory Object:**
  - Configuration object is used to create a SessionFactory object which inturn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated.
  - SessionFactory is a thread safe object and used by all the threads of an application.
  - The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use.
  - You would need one SessionFactory object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactory objects.

HIBERNATE

- **Session Object:**
  - A Session is used to get a physical connection with a database.
  - It is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.
  - The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed as needed.
- **Transaction Object:**
  - A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).
  - This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.
- **Query Object:**
  - Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters and to execute the query.

HIBERNATE TERMINOLOGY

- Configuration Object
  - Database Connection
  - Class Mapping Setup
- SessionFactory Object
  - Factory for sessions
  - Thread-safe
  - heavyweight object
  - created during application start up and kept for later use
  - At least one SessionFactory object per database

Hibernate Terminology (2)

- Session Object
  - For physical connection with a database
  - lightweight
  - instantiated each time an interaction is needed with DB
  - Not thread-safe
- Transaction Object
- Query Object
  - String-based
  - HQL: some OO facilities is available
  - SQL: usually is not recommended. Why?
- Criteria Object
  - Fully object oriented queries

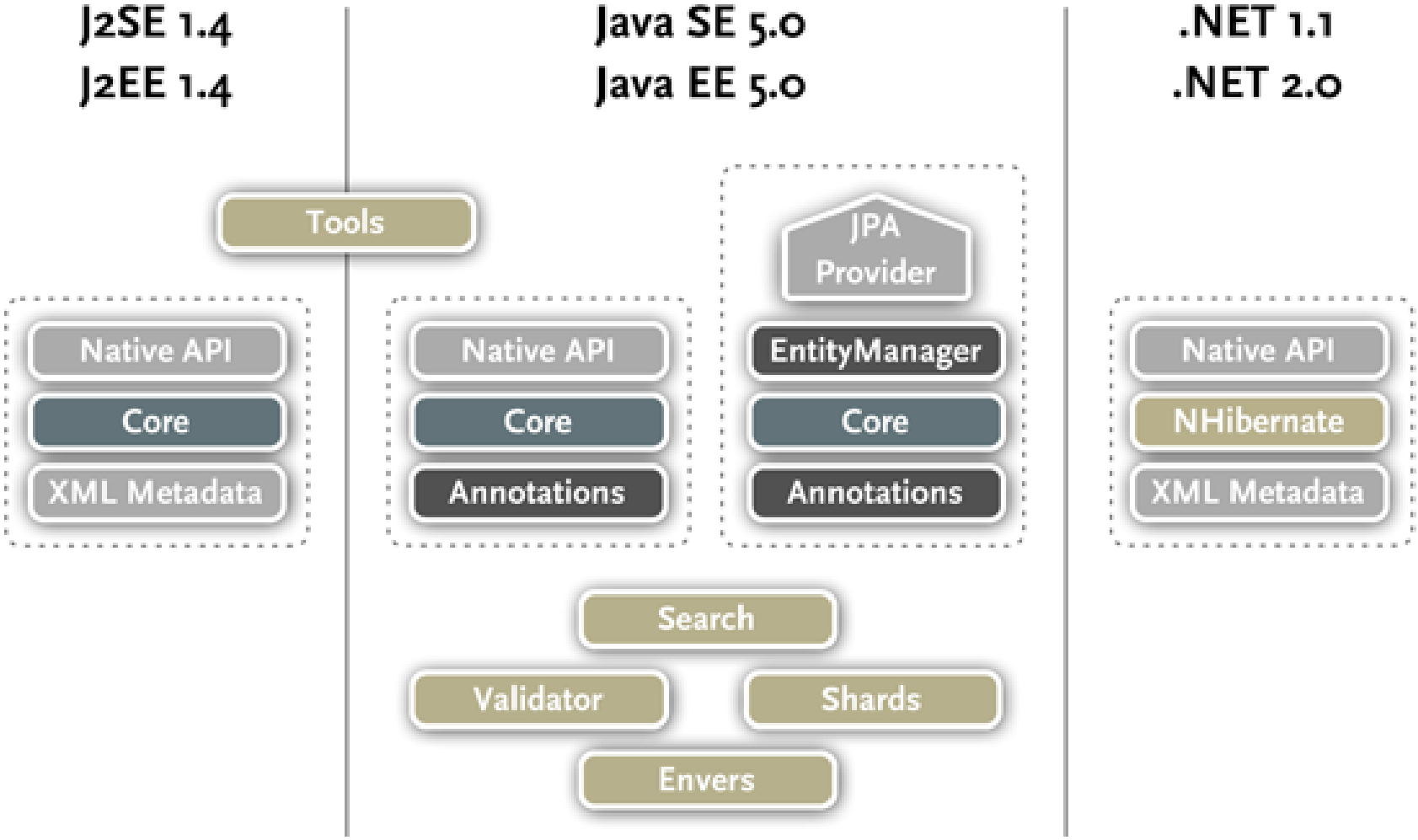
Benefits of Hibernate

- Simpler data persistence
- Automatically handles mapping SQL to Object and vice versa
- Automatic creation of database schemas
- Automatic updating of database schemas
  - Add a field to an object; Hibernate converts your existing database for you.
- Provides search functionality

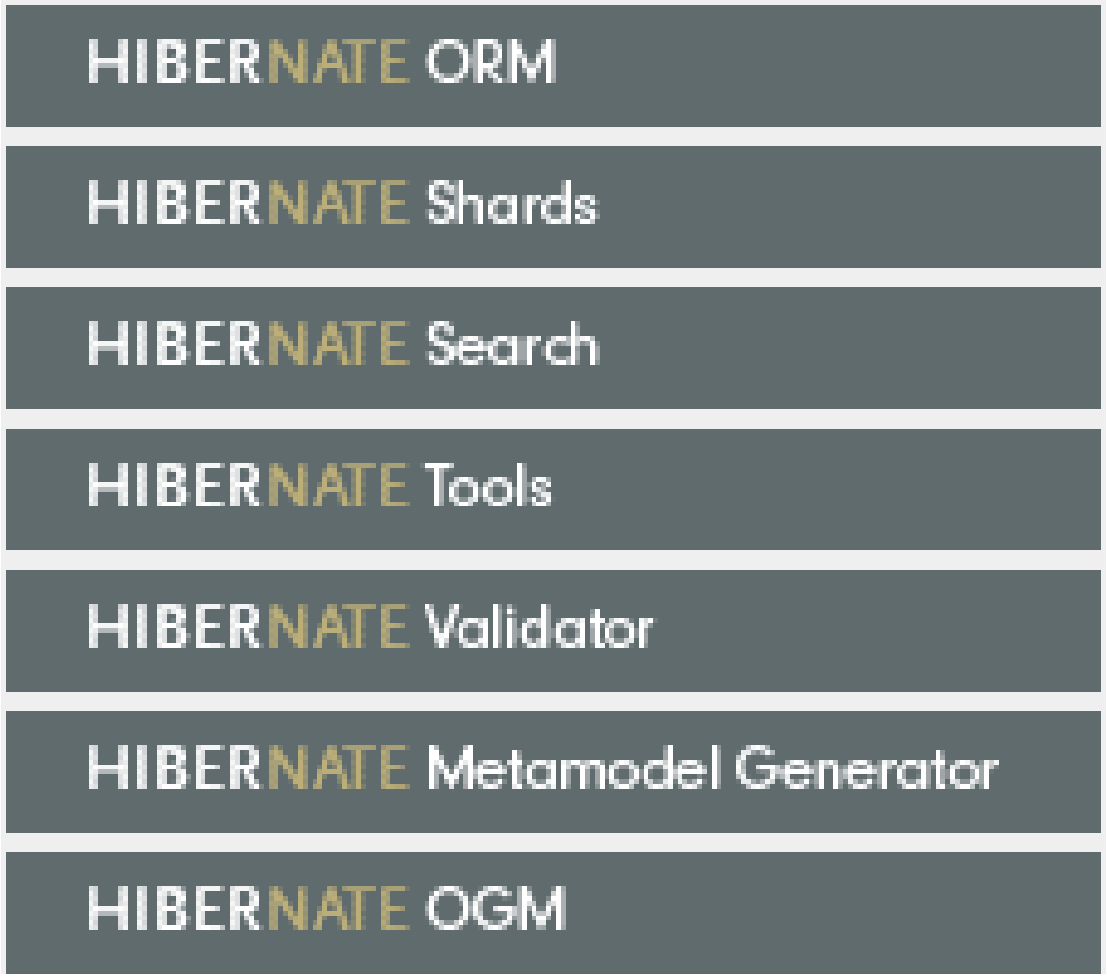
Benefits of Hibernate (2)

- Simpler database management
- No JDBC code or SQL code needed
  - Yet you can still use SQL if you want
- Easy to swap out database engines by a simple configuration change
  - DBMS independence
  - No need to create the schema on the new database
- It’s free
  - LGPL (use in open or closed source project)
  - Open source and standards = no vendor lock-in

HIBERNATE STACK



HIBERNATE PROJECTS



HIBERNATE PROJECTS (2)

- Core
  - Provides the core functionality of object relational mapping and persistence
- Shards
  - Provides for horizontal partitioning of Core so you can put object data in multiple databases
  - Sharding: Rows of a database table are held separately
- Search
  - Combines Apache Lucene full-text search engine with Core.
  - Extends the basic query capabilities of Core.
- Tools
  - Eclipse plugins to facilitate
    - Creating Hibernate mapping files
    - Database connection configurations
    - Reverse engineering existing databases into Java class code

HIBERNATE PROJECTS (3)

- Validator
  - JSR 303 - Bean Validation
  - Standardized way of annotating JavaBean fields with value constraints
  - @NotNull on a bean field also gets set in the database schema
- Metamodel Generator
  - Auto-creation of Criteria classes for use in JPA Criteria API
  - Non-string (strongly-typed) based API for object-based queries
- OGM (Object/Grid Mapper)
  - Allows use of NoSQL data stores versus SQL relational
  - providing Java Persistence (JPA) support for NoSQL solutions

JDBC

- Java DataBase Connectivity
- API for accessing relational databases from a Java program
- SQL-based
- Flexible architecture to write DBMS-independent applications
  - Fully independent? no.

JDBC Pros and Cons

- Pros
  - Clean and simple SQL processing
  - Good performance with large data
  - Very good for small applications
  - Simple syntax so easy to learn
- Cons
  - Complex if it is used in large projects
  - Large programming overhead
  - No encapsulation
  - Query is DBMS specific

JDBC EXAMPLE

```
Class.forName("com.mysql.jdbc.Driver");
String SQL = "SELECT * FROM classes";
try (
    Connection conn = DriverManager.getConnection(
        "jdbc:mysql://localhost/cse3330a",
        "root", "aSecret123");
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(SQL)
    ) {

  while (rs.next()) {
    System.out.println(rs.getString("prof"));
  }
}
```

JDBC + SPRING

- DataSource object
- JdbcTemplate object
- RowMapper
  - a simple ORM for select queries

```
String SQL = "select age from Student where id = ?";
int age = jdbcTemplate.queryForInt(SQL, new Object[]{10});


String SQL = "select * from Student where id = ?";
Student student = jdbcTemplate.queryForObject(
SQL, new Object[]{10}, new StudentMapper());
```

Object Relational Mapping

- RDBMSs represent data in a tabular format
  - Relational
- Object-oriented languages, such as Java or C# represent data as an interconnected graph of objects
- Context gap

Hibernate Class Entities

- Class attributes
  - Hibernate uses reflection to populate
  - Can be private or whatever
- Class requirements
  - Default constructor
- JavaBean pattern common
- Do not make the entity classes **final**
- 3 methods of serialization definition
  - Following slides

Hibernate Annotation Mappings

- Annotations in code
  - Beginning of class
  - Indicate class is Entity
    - Class doesn’t have to implement java.lang.Serializable
  - Define database table
  - Define which attributes to map to columns
    - Supports auto-increment IDs too
    - Can dictate value restrictions (not null, etc)
    - Can dictate value storage type
- Existed before JPA standard (later slides)
- Doesn’t require a separate hbm.xml mapping file (discussed later)
  - But is tied to code

HIBERNATE ANNOTATION EXAMPLE

```
@Entity
@Table( name = "EVENTS" )
public class Event {
   private Long id;
…
    @Id
    @GeneratedValue(generator="increment”
    @GenericGenerator(name="increment", strategy = "increment")
    public Long getId() {  return id;    }

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "EVENT_DATE")
    public Date getDate() { return date;    }
    public void setDate(Date date) { this.date = date;    }
…
}
```

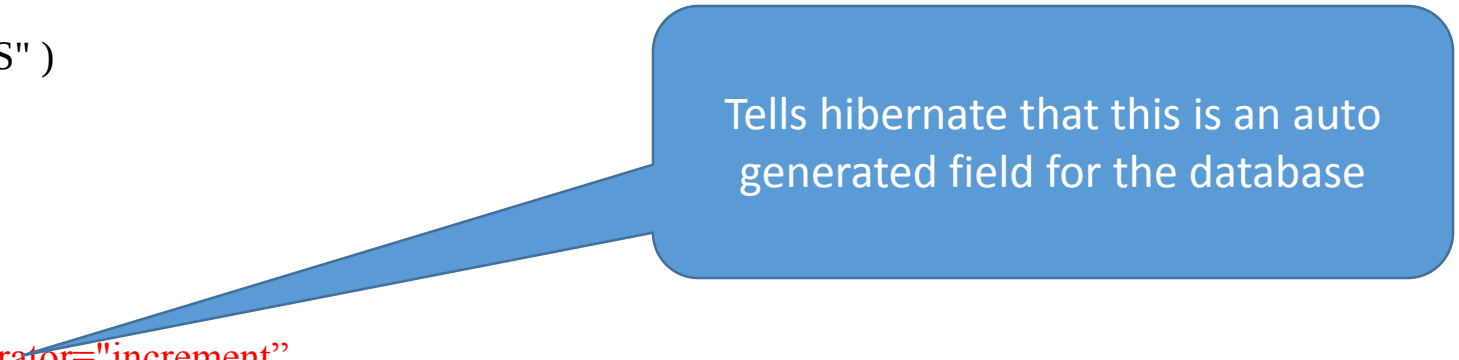
HIBERNATE ANNOTATION EXAMPLE (2)

```
@Entity
@Table( name = "EVENTS" )
public class Event {
   private Long id;
…
    @Id
    @GeneratedValue(generator="increment”
    @GenericGenerator(name="increment", strategy = "increment")
    public Long getId() {  return id;    }

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "EVENT_DATE")
    public Date getDate() { return date;    }
    public void setDate(Date date) { this.date = date;    }
…
}
```

Tells hibernate this goes into the EVENTS table

HIBERNATE ANNOTATION EXAMPLE (3)

```
@Entity
@Table( name = "EVENTS" )
public class Event {
   private Long id;
…
    @Id
    @GeneratedValue(generator="increment”
    @GenericGenerator(name="increment", strategy = "increment")
    public Long getId() {  return id;    }

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "EVENT_DATE")
    public Date getDate() { return date;    }
    public void setDate(Date date) { this.date = date;    }
…
}
```



HIBERNATE ANNOTATION EXAMPLE (4)

```
@Entity
@Table( name = "EVENTS" )
public class Event {
   private Long id;
…
    @Id
    @GeneratedValue(generator="increment”
    @GenericGenerator(name="increment", strategy = "increment")
    public Long getId() {  return id;    }

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "EVENT_DATE")
    public Date getDate() { return date;    }

    public void setDate(Date date) { this.date = date;    }
…
}
```

Note that you don’t need any annotations on the actual private fields or setters if you use the standard JavaBean pattern. The getter defines it.

Hibernate Annotation Example (5)

```
@Entity
@Table( name = "EVENTS" )
public class Event {
…
   private String title;

   public String getTitle() {  return title;    }

   public void setTitle(String title) {         this.title = title;    }
…
}
```

Also note that this is automatically stored with a column name of “title” so we didn’t have to add any annotations

JPA ANNOTATION

- Became standard
  - Came after Hibernate annotations
- Works almost like Hibernate annotations
  - Requires “META-INF/persistence.xml” file
    - Defines data source configuration
    - HibernatePersistence provider
      - Auto-detects any annotated classes
      - Auto-detects any hbm.xml class mapping files
        - (later slides)
      - Allows explicit class loading for mapping
- Annotation syntax
  - Same as Hibernate
  - Hibernate has a few extensions (see docs)

JPA 2 XML

- JPA 2 XML
  - like Hibernate's hbm.xml discussed later
- Separate from code unlike in-line annotations

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm ;
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
    version="1.0">
    <description>My First JPA XML Application</description>
    <package>entity</package>
    <entity class="entity.Employee" name="Employee">
        <table name="EMPLOYEETABLE"/>
        <attributes>
            <id name="empId">
                <generated-value strategy="TABLE"/>
            </id>
            <basic name="empName">
                <column name="EMP_NAME" length="100"/>
            </basic>
            <basic name="empSalary">
            </basic>
        </attributes>
   </entity>
</entity-mappings>
```

Hibernate *.hbm.xml Mappings

- For each class
  - Define ClassName.hbm.xml
    - ClassName not required convention
  - Usually stored in same package
    - I like the convention of
      - src/main/java (actual .java source)
      - src/main/resources (any configuration files, et cetera)
      - src/main/hibernate (I put all of my .hbm.xml here)
        - Matching folder/package structure to src/main/java
  - Optionally multiple classes in one .hbm.xml possible
    - Not common convention though
- Becoming legacy in favor of JPA 2 XML or Annotations

Hibernate *.hbm.xml mappings (2)

```
<?xml version="1.0"?>


<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="org.hibernate.tutorial.hbm">

    <class name="Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="increment"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>

</hibernate-mapping>
```

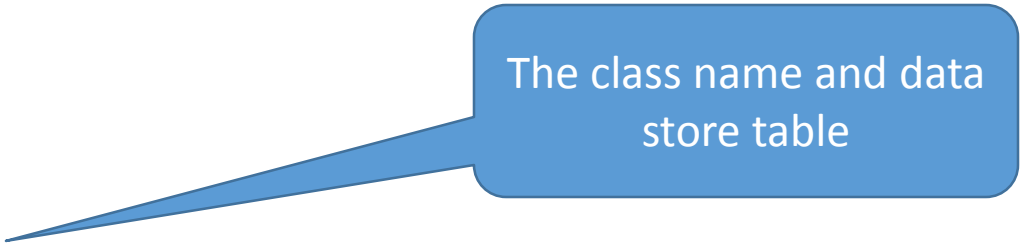
Hibernate *.hbm.xml mappings (3)

```
<?xml version="1.0"?>


<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="org.hibernate.tutorial.hbm">

    <class name="Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="increment"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>

</hibernate-mapping>
```

Define what package for the class(es)

Hibernate *.hbm.xml mappings (4)

```
<?xml version="1.0"?>


<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="org.hibernate.tutorial.hbm">

    <class name="Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="increment"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>

</hibernate-mapping>
```



HIBERNATE *.HBM.XML MAPPINGS (5)

```
<?xml version="1.0"?>


<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="org.hibernate.tutorial.hbm">

    <class name="Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="increment"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>

</hibernate-mapping>
```

The properties of the class to save (getAttribute() or class.attribute style)

HIBERNATE PRIMITIVE TYPES

Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

DATE TYPES

date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

BINARY AND LARGE OBJECT TYPES

Mapping type	Java type	ANSI SQL Type
binary	byte[]	VARBINARY (or BLOB)
text	java.lang.String	CLOB
serializable	any Java class that implements java.io.Serializable	VARBINARY (or BLOB)
clob	java.sql.Clob	CLOB
blob	java.sql.Blob	BLOB

Hibernate Configuration

- Need to define
  - Data source (database) connection
    - Engine, URI, credentials, etc
    - Pooling mechanism (Hibernate provides C3P0)
  - Mapping XML definitions or classes with annotations
    - You specify which ones to actually activate
    - Also allows for multiple databases and mappings
    - **Many IDE and Ant tools exist to auto-create .hbm.xml and hibernate.cfg.xml/persistence.xml**
- JPA
  - Provides some auto-discovery at startup
    - Limited to jar files
- Hibernate
  - Has XML or .properties (not used much)
- Approaches
  - XML configuration file
  - Programmatic

HIBERNATE.CFG.XML

- Usually placed in root of the classpath (/hibernate.cfg.xml) for auto-detection.
- You can also programmatically load it in code.

```
<hibernate-configuration>
    <session-factory>
…
    </session-factory>
</hibernate-configuration>
```

<SESSION-FACTORY>

```
<property name="connection.driver_class">org.h2.Driver</property>
<property name="connection.url">jdbc:h2:mem:db1 </property>
<property name="connection.username">sa</property>
<property name="connection.password">123</property>

<property name="connection.pool_size">1</property>

<property name="dialect">org.hibernate.dialect.H2Dialect</property>

<property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

<property name="show_sql">true</property>

<property name="hbm2ddl.auto">create</property>

<mapping class="org.hibernate.tutorial.annotations.Event"/>
```

HIBERNATE.CFG.XML (2)

```
<!-- Database connection settings -->
<property name="connection.driver_class">org.h2.Driver</property>
<property name="connection.url">jdbc:h2:mem:db1</property>
<property name="connection.username">sa</property>
<property name="connection.password"></property>
```

- You define the database settings in this manner
- There are third party library extensions that allow the password inside hibernate.cfg.xml to be encrypted as well
  - Or you could use programmatic configuration instead
- You can’t use system/environment variables inside the config

HIBERNATE.CFG.XML (3)

```
<!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.H2Dialect</property>
```

- Depending on the database/data source you use you need to let Hibernate know how to talk to it. Hibernate supports many data sources.

```
    <!-- Drop and re-create the database schema on startup -->
    <property name="hbm2ddl.auto">create</property>
```

- You can have Hibernate:
  - “validate” - Exists and has expected tables/columns; don’t touch data/schema
  - “update” – Create if needed and update tables/columns if class has new fields
  - “create” – Drop any existing and create new (only at start of session)
  - “create-drop” – Same as create but also drop at end of session

Caveats of hbm2ddl.auto

- Some people don't recommend
  - "update" can mess up
    - Doesn't remove renamed columns/attributes
      - Makes new one by default unless you update mapping
    - Migrations not perfect
      - Suppose add not null constraint after the fact
        - Existing nulls would blow up
    - Not recommended for production use anyway
- Hibernate class tools like
  - SchemaExport and SchemaUpdate
  - Useful for getting auto-generated SQL migration/creation scripts
- When set to "create" or "create-drop"
  - hbm2ddl.import_file – allow specifying */path/somefile.sql* to run
  - (used to be hard coded "/import.sql")
  - Needs to be in classpath

HIBERNATE.CFG.XML (4)

<!-- Names the annotated entity class -->

<mapping **class**="org.hibernate.tutorial.annotations.Event"/>

- For each class with annotations you provide a <mapping/> entry

- For each .hbm.xml it looks like this instead
  - <mapping **resource**="org/hibernate/tutorial/domain/Event.hbm.xml"/>
  - Event.hbm.xml is in the classpath
  - No *.hbm.xml is possible, sorry. See persistence.xml for wildcards or programmatic configuration.

PERSISTENCE.XML

- JPA 2 XML method
  - Similar idea to hibernate.cfg.xml
- Additions
  - Allows defining EJB Persistence provider
    - Usually HibernatePersistence suffices
  - jar-file option
    - Allows auto-inclusion of any classes with annotations
      - No need for manual mapping like hibernate.cfg.xml
    - Also auto-includes any .hbm.xml files

Programmatic Configuration

```
Configuration cfg = new Configuration();
cfg.addResource("Item.hbm.xml");
cfg.addResource("Bid.hbm.xml");
```

- Assumes the .hbm.xml are in the classpath
  - This example assumes the root

Programmatic configuration *(2)*

```
Configuration cfg = new Configuration();
cfg.addClass(org.hibernate.auction.Item.class);
cfg.addClass(org.hibernate.auction.Bid.class);
```

- Translates to “/org/hibernate/auction/Item.hbm.xml”
  - Again in classpath
  - Avoids hardcoded filenames (less work than updating hibernate.cfg.xml)
- If class is annotated uses annotation versus xml

Programmatic configuration (3)

```
Configuration cfg = new Configuration()
    .addClass(org.hibernate.auction.Item.class)
    .addClass(org.hibernate.auction.Bid.class)
    .setProperty("hibernate.dialect",
  "org.hibernate.dialect.MySQLInnoDBDialect")
    .setProperty("hibernate.connection.datasource",
  "java:comp/env/jdbc/test")
    .setProperty("hibernate.order_updates", "true");
```

- Database settings can be configured as well
  - Note the “hibernate.” prefix on the options this time

Programmatic Configuration (4)

```
Configuration conf = new Configuration();
conf.configure().setProperty("hibernate.connection.url",
   "jdbc:mysql://"+host+"/"+dbname);

conf.setProperty("hibernate.connection.username", user);

conf.setProperty("hibernate.connection.password", password);
```

- Load /hibernate.cfg.xml as defaults
  - Mappings still defined inside XML
- Override database settings with run-time values

Mapping & Configuration Summary

- Each Java class
  - Annotations or XML mapping file (.hbm.xml)
    - or JPA persistence orm.xml
    - You can mix annotations and XML mappings
- Hibernate XML Configuration
  - Each class/.hbm.xml gets <mapping/> entry
  - Configure database
  - hibernate.cfg.xml
    - or for JPA use persistence.xml
  - Alternative
    - Programmatic configuration
    - Hybrid approach

INITIALIZING HIBERNATE

```
try {
        // Create the SessionFactory from hibernate.cfg.xml
        return new Configuration().configure().buildSessionFactory();
} catch (Throwable ex) {
        // Make sure you log the exception, as it might be swallowed
        System.err.println("Initial SessionFactory creation failed." + ex);
        throw new ExceptionInInitializerError(ex);
}
```

- SessionFactory returned is used later

Saving data

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();

Event theEvent = new Event();
theEvent.setTitle(title);
theEvent.setDate(theDate);

session.save(theEvent);

session.getTransaction().commit();
```

- Just start a transaction similar to how you do in databases

LOADING DATA

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();  // important even for query
List result = session.createQuery("from Event").list();
session.getTransaction().commit();
return result;
```

- Note the “from Event” which is SQL like
  - Known as **HQL**
  - More complex queries possible
  - See http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/queryhql.html

HIBERNATE SESSION

- A Session is used to get a physical connection with DB
- The Session object is lightweight
- designed to be instantiated each time an interaction is needed with DB
- Persistent objects are saved and retrieved through a Session object
- Session vs. DB connection
- Every object that is loaded by Hibernate from the database is associated with the session
  - Allowing Hibernate to automatically persist objects that are modified
  - Allowing Hibernate to implement functionality such as **lazy-loading**

Sessions and Threads

- Hibernate sessions are not thread-safe
- you shouldn't pass a Hibernate session into a new thread
- you must not share Hibernate-managed objects between threads
- Spring's transaction management places the Hibernate session in a ThreadLocal variable
  - accessed via the sessionFactory
  - DAOs use that ThreadLocal
  - When you create a new thread you no longer have access to the Hibernate session for that thread

OpenSessionInView Filter

- The SessionInViewFilter
- Opens a Hibernate session which is then available for the entire web request

- The pattern has pros and cons

Querying Data

- Remember don’t use string concatenation to form queries
  - Bad: “from Users where id=” + paramUserId
  - Why? **SQL Injection** Vulnerabilities
    - paramUserId = “1 OR 1=1”
- Use Queries instead
  - List cats = sess.createCriteria(Cat.class)
    .add(Restrictions.like("name", "Fritz%"))
    .add(Restrictions.between("weight", minWeight, max))
    .list();
    - http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/#querycriteria
  - List cats = session.createQuery(
    “from Cat as cat where cat.birthdate < **?**“)
    .setDate(0, date)
    .list();
    - http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/#objectstate-querying

CLOSING CONNECTION

sessionFactory.close();

- Closes resources
  - Connection pools
  - Caches
  - Etc.

Session and Thread Safety

- org.hibernate.Session
  - Don’t use the deprecated class org.hibernate.classic.Session
- **NOT** thread safe
  - Each thread should call SessionFactory.getCurrentSession()
    - Versus .openSession() which has
      - No automatic close management
      - No automatic association to a thread

HQL

- Hibernate Query Language
- Currently a superset of JPQL
- Select, insert, delete, update queries
- SQL-Similar syntax
- With most of its features
  - Join
  - Order
  - Group
  - Aggregations
  - …

HQL (Examples)

- from Cat
- select cat.mate from Cat cat
- select cat.name from DomesticCat cat where cat.name like 'fri%‘
- select avg(cat.weight), sum(cat.weight) from Cat cat

- HQL is OO
- The names are properties (not columns)

FIND BY CRITERIA

```
Criteria criteria =
session.createCriteria(Person.class);
SimpleExpression like =
   Restrictions.like("firstName", "%i%");
Criteria exampleCriteria = criteria.add(like);
List<Person> list = exampleCriteria.list();
for (Person person2 : list) {
 System.out.println(person2.getFirstName());
}
```

RESTRICTIONS

- Eq
- Gt
- Le
- Like
- And
- Or
- …

FIND BY EXAMPLES

```
Person person = new Person();
person.setNationalCode(123);
person.setFirstName("Taghi");
person.setLastName("Taghavi");
Criteria criteria =
    session.createCriteria(Person.class);
Criteria exampleCriteria =
criteria.add(Example.create(person));
List<Person> list = exampleCriteria.list();
for (Person person2 : list) {
     System.out.println(person2.getFirstName());
}
```

Associations Mapping

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many

MANY-TO-ONE

```
<class name="Employee" table="EMPLOYEE">
    <meta attribute="class-description"> This clas
    <id name="id" type="int" column="id">
        <generator class="native" />
    </id>
    <property name="firstName" column="first_name"
    <property name="lastName" column="last_name" t
    <property name="salary" column="salary" type="
    <many-to-one name="address" column="address"
    class="Address" not-null="true" />
</class>
```

MANY-TO-ONE (CONT'D)

```
public class Employee{
   private int id;
   private String firstName;
   private String lastName;
   private int salary;
   private Address address;
```

- + getters and setters

MAY-TO-ONE BY ANNOTATION

```
@Entity
@Table(name="cats")
public class Cat {
@ManyToOne
@JoinColumn(name="mother_id")
public Cat getMother() { return mother; }
…
```

ONE-TO-ONE

- similar to many-to-one association
- the column will be set as unique
- E.g. an address associated with a single employee
- Applications?

```
public class Employee{
   private int id;
   private String firstName;
   private String lastName;
   private int salary;
   private Address address;
```

ONE-TO-MANY

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <set name="addresses">
        <key column="personId"
            not-null="true"/>
        <one-to-many class="Address"/>
    </set>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
```

ONE-TO-MANY (CONT'D)

```
public class Product {
   private String serialNumber;
   private Set<Part> parts;

   public String getSerialNumber() {return serialNumber; }
   void setSerialNumber(String sn) { serialNumber = sn; }

   public Set<Part> getParts() { return parts; }
   void setParts(Set parts) { this.parts = parts; }
}
```

ONE-TO-MANY BY ANNOTATIONS

```
@Entity
public class Product {
    private String serialNumber;
    private Set<Part> parts;

    @Id
    public String getSerialNumber() {return serialNumber; }
    void setSerialNumber(String sn) { serialNumber = sn; }

    @OneToMany
    @JoinColumn(name="PART_ID")
    public Set<Part> getParts() { return parts; }
    void setParts(Set parts) { this.parts = parts; }
}
```

MANY-TO-MANY

```
@Entity
public class Store {
  @ManyToMany
  public Set<City> getCities() {          ...     }
}


@Entity
public class City {     ... //no bidirectional relationshi
p }
```

Many-to-Many (cont'd)

```
@Entity
public class Store {
    @ManyToMany(cascade = {CascadeType.PERSIST,
    public Set<Customer> getCustomers() {
        ...
    }
}

@Entity
public class Customer {
    @ManyToMany(mappedBy="customers")
    public Set<Store> getStores() {
        ...
    }
}
```

Collection Mapping Types

- java.util.Set
  - mapped with a <set> element, initialized with HashSet
- java.util.SortedSet
  - mapped with a <set> element, initialized with java.util.TreeSet.
  - The **sort** attribute
- java.util.List
  - mapped with a <list> element, initialized with java.util.ArrayList

Collection Mapping Types (Cont'd)

- java.util.Collection
  - mapped with a <bag> or <ibag> element , initialized with ArrayList
- java.util.Map
  - mapped with a <map> element
  - initialized with java.util.HashMap
- java.util.SortedMap
  - mapped with a <map> element
  - initialized with java.util.TreeMap.
  - The **sort** attribute

CASCADE

- Cascades the operation to the children
- Cascade-types:
  - none
  - Save-update
  - Delete
  - Delete-orphan
  - …

EXAMPLE: CASCADE=SAVE-UPDATE

```
<!-- Stock.hbm.xml -->
<set name="stockDailyRecords" cascade="save-update" table="stock_
        <key>
                <column name="STOCK_ID" not-null="true" />
        </key>
        <one-to-many class="com.mkyong.common.StockDailyRecord" />
</set>
```

```
Stock stock = new Stock();
StockDailyRecord stockDailyRecords = new StockDailyRecord();
//set the stock and stockDailyRecords  data

stockDailyRecords.setStock(stock);
stock.getStockDailyRecords().add(stockDailyRecords);

session.save(stock);
```

Saves some stockDailyRecords too

EXAMPLE: CASCADE=DELETE

```
<!-- Stock.hbm.xml -->
<set name="stockDailyRecords" cascade="delete" table="stock_daily
      <key>
            <column name="STOCK_ID" not-null="true" />
      </key>
      <one-to-many class="com.mkyong.common.StockDailyRecord" />
</set>
```

```
Query q = session.createQuery("from Stock where stockCode = :stockCode ");
q.setParameter("stockCode", "4715");
Stock stock = (Stock)q.list().get(0);
session.delete(stock);
```

EXAMPLE: CASCADE=DELETE-ORPHAN

```
<!-- Stock.hbm.xml -->
<set name="stockDailyRecords" cascade="delete-orphan" table="stoc
        <key>
                <column name="STOCK_ID" not-null="true" />
        </key>
        <one-to-many class="com.mkyong.common.StockDailyRecord" />
</set>
```

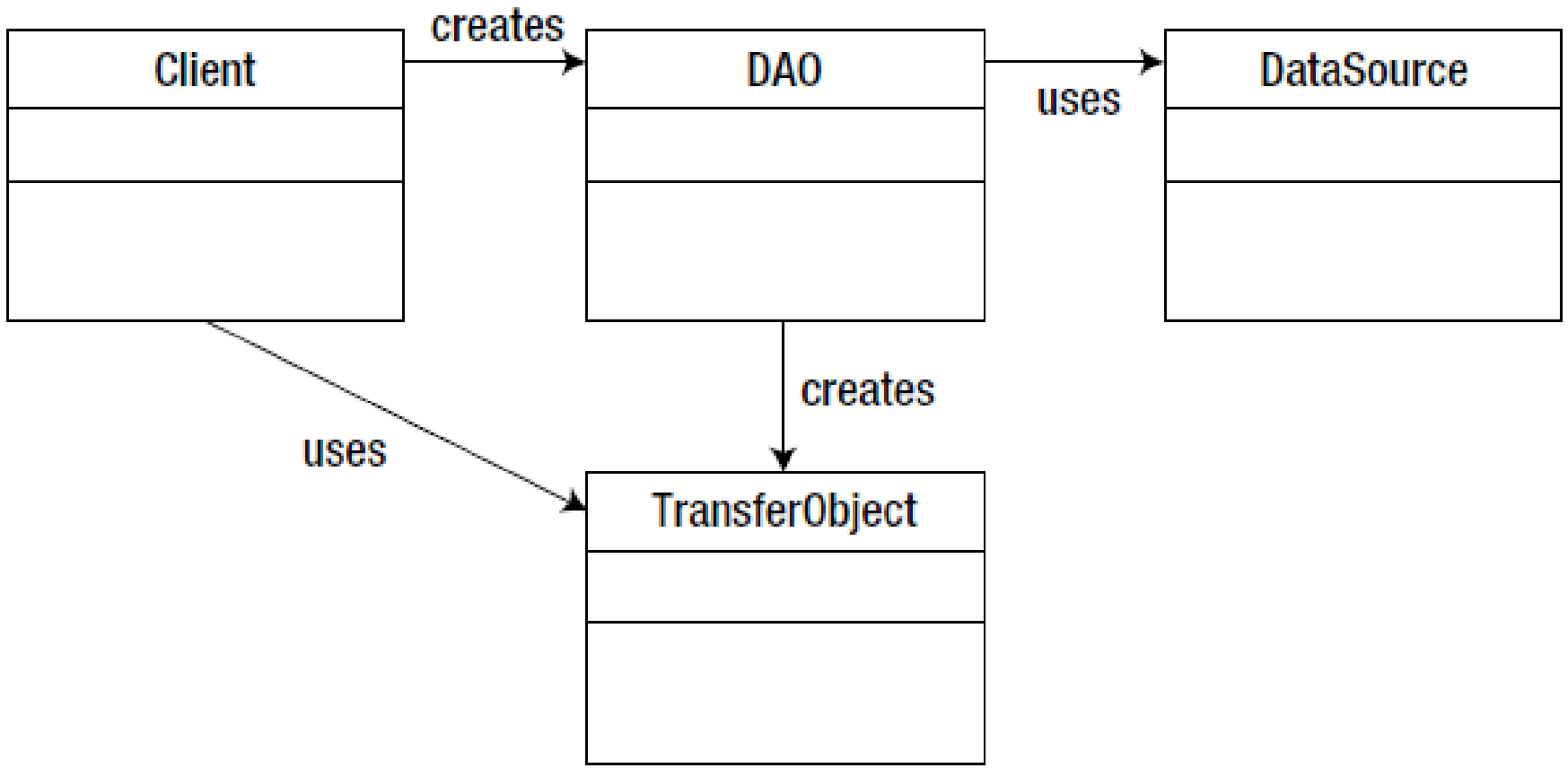
```
StockDailyRecord sdr1 = (StockDailyRecord)session.get(StockDailyRecord.class,
                                        new Integer(56));
StockDailyRecord sdr2 = (StockDailyRecord)session.get(StockDailyRecord.class,
                                        new Integer(57));

Stock stock = (Stock)session.get(Stock.class, new Integer(2));
stock.getStockDailyRecords().remove(sdr1);
stock.getStockDailyRecords().remove(sdr2);

session.saveOrUpdate(stock);
```

Lazy Collection Fetching

- A collection is fetched when the application invokes an operation upon that collection
- This is the default for collections

- lazy=true | false

Data Access Object Pattern (DAO)



DAO

- Abstracts the details of the underlying persistence mechanism
- Hides the implementation details of the data source from its clients
- **Loose coupling** between core business logic and persistence mechanism

DAO: EXAMPLE

```
public interface UserDAO {

   void insertUser(User user);

   User getUserById(int userId);

   User getUser(String username);

   List<User> getUsers();
}
```

HIBERNATE + SPRING

- Springs helps you write hibernate codes
  - better and simpler
- In creating and managing objects (beans)
  - Datasource
  - SessionFactory
  - Sessions
- In Transaction Management

Spring Integration:
1. Write Entity Classes

```
@Entity
@Table(name="USER")
public class User {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  @Column(name="ID", nullable = false)
  private int id;

  @Column(name="USERNAME", nullable = false)
  private String username;

  @Column(name="NAME", nullable = false)
  private String name;

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }
```

Spring Integration:
2. DOA Interfaces

```
public interface UserDAO {

   void insertUser(User user);

   User getUserById(int userId);

   User getUser(String username);

   List<User> getUsers();
}
```

Spring Integration:
3. DAO Implementations

```
public class UserDAOImpl implements UserDAO {

  @Autowired
  private SessionFactory sessionFactory;

  @Override
  public void insertUser(User user) {
    sessionFactory.getCurrentSession().save(user);
  }

  @Override
  public User getUserById(int userId) {
    return (User) sessionFactory.
      getCurrentSession().
      get(User.class, userId);
  }
```

Spring Integration:
4. Service (Manager, Business) Interface

```
public interface UserManager {

  void insertUser(User user);

  User getUserById(int userId);

  User getUser(String username);

  List<User> getUsers();
}
```

Spring Integration:
5. Service Implementation

```
public class UserManagerImpl implements UserManager {

  @Autowired
  private UserDAO userDAO;

  @Override
  @Transactional
  public void insertUser(User user) {
    userDAO.insertUser(user);
  }

  @Override
  @Transactional
  public User getUserById(int userId) {
    return userDAO.getUserById(userId);
  }
```

Spring Integration:
6. Spring Configuration (1)

```
<tx:annotation-driven />
```

SPRING INTEGRATION:
6. SPRING CONFIGURATION (2)

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
  destroy-method="close">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost:3306/TEST" />
  <property name="username" value="testuser" />
  <property name="password" value="testpasswd" />
</bean>
```

SPRING INTEGRATION:
6. SPRING CONFIGURATION (3)

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalS
  <property name="dataSource" ref="dataSource"></property>
  <property name="hibernateProperties">
    <props>
      <prop
       key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>
      <prop key="hibernate.show_sql">true</prop>
    </props>
  </property>
  <property name="packagesToScan" value="com.byteslounge.spring.tx.model"
</bean>
```

Spring Integration:
6. Spring Configuration (4)

```
<bean id="transactionManager"
  class="org.springframework.orm.hibernate4.HibernateTransactionManager"
  p:sessionFactory-ref="sessionFactory">
</bean>
```

Spring Integration:
6. Spring Configuration (5)

- Define beans
  - DAO beans
  - Service beans
- By XML or annotations

Spring Integration:
7. Using the Service (Manager) Beans

```
ApplicationContext ctx =
  new ClassPathXmlApplicationContext("spring.xml");
UserManager userManager =
  (UserManager) ctx.getBean("userManagerImpl");

User user = new User();
user.setUsername("johndoe");
user.setName("John Doe");

userManager.insertUser(user);
```

Generic DAO

- The problem with many DAO implementations
- Similar methods for
  - Load
  - Save, update, delete
  - Search
- The solution?

Generic DAO (cont'd)

```
public class GenericDaoHibernateImpl <T, PK extends Serializable>
    implements GenericDao<T, PK>, FinderExecutor {
    private Class<T> type;

    public GenericDaoHibernateImpl(Class<T> type) {
        this.type = type;
    }

    public PK create(T o) {
        return (PK) getSession().save(o);
    }

    public T read(PK id) {
        return (T) getSession().get(type, id);
    }

    public void update(T o) {
        getSession().update(o);
    }

    public void delete(T o) {
        getSession().delete(o);
    }
```

Hibernate Alternatives

- ORM
  - Enterprise JavaBeans (Entity Beans)
  - Java Data Objects (JDO)
  - Castor
  - Spring DAO
- Other JPA implementations
  - TopLink , OpenJPA, …

EXERCISES (JAVA SCRIPT)

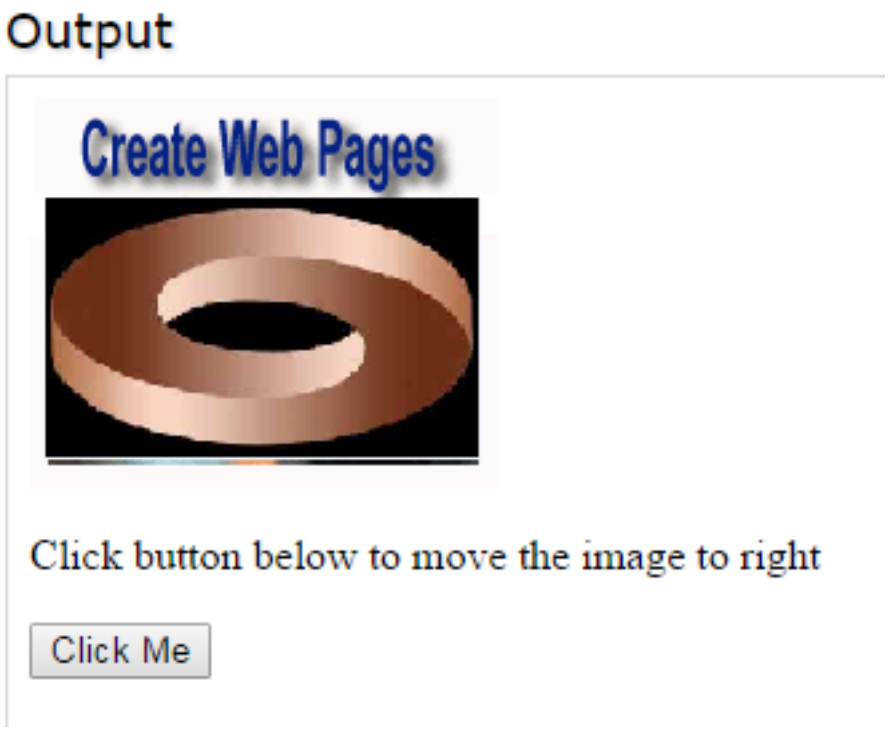
1) Write a JavaScript function that creates a table, accept row, column numbers from the user, and input row-column number as content (e.g. Row-0 Column-0) of a cell.

Sample HTML file :

- <!DOCTYPE html>
- <html>
- <head>
- <meta charset=utf-8 />
- <title>Change the content of a cell</title>
- <style type="text/css">
- body {margin: 30px;}
- </style>
- </head><body>
- <table id="myTable" border="1">
- </table><form>
- <input type="button" onclick="createTable()" value="Create the table">

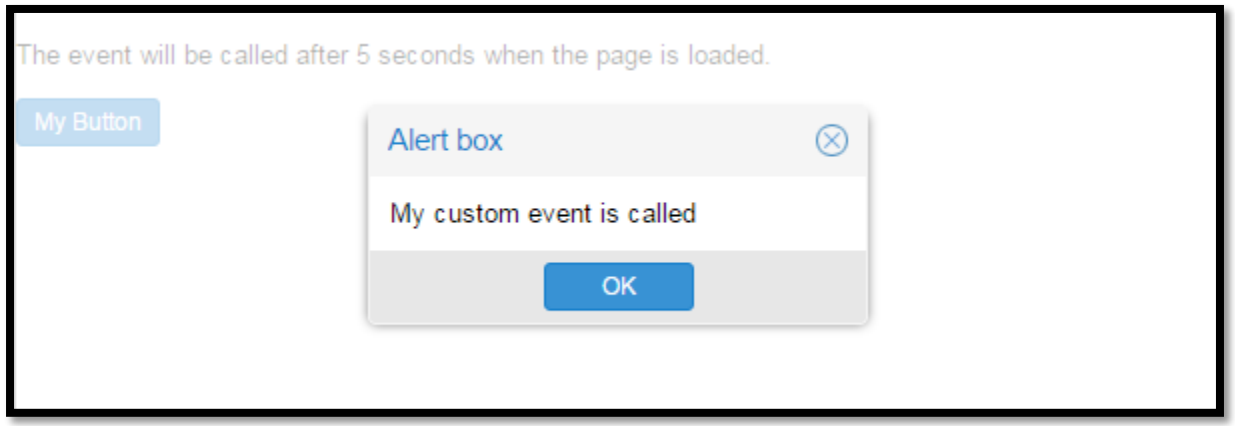
EXERCISES (JAVA SCRIPT)

- **2)** Use JavaScript to create a complex animation having, but not limited to, elements Object movements.



EXERCISES (JAVA SCRIPT)

- **3)** Write custom events in ext JS and fire the events with fireEvent method. Try different events and event listeners. Then have it moving in different direction using drag operation.



EXERCISES (JAVA)

- **4)** Create a java program that returns the file number, file names, directory number, directory name in a given directory in your local machine.

Input sample: A location in your hdd : D:\\testFolder\

Output sample: total directory number: 6

test1\

test2\

……

total file number: 2

myfile.txt

EXERCISES (JAVA)

- **5)** Write a java program to remove all white spaces and non-English characters (if any) from a given string.

- **6)** Write a java program to detect the deadlocked threads.

- **7)** Write java code to create simple text files on your disk with some string written in it. Then copy those data to another file.

EXERCISES (JAVA)

- **8)** ArrayList gives the flexibility of adding multiple null elements, duplicate elements and also maintains the insertion order of elements. First create an ArrayList with duplicate entry and then remove duplicate elements from already constructed ArrayList. Try to input and manipulate different language strings as input than English.

- **9)** Write a java program which opens a new instance of notepad and windows media player using exec() method which takes system command as a string.

EXERCISES (JAVA)

- 10) Write a java program that reads and downloads the files from a given url, saves that in your local directory. It appends (not overwrite) the content in that same file when another url is accessed.

- Sample input: www.anysite.com
- www.anothersite.com
- Sample output. A simple text or .doc file containing all the contents of those site.

REMARKS

- 1) More on Spring/ Hibernate application/ lazy loading
- 2) More on P/L SQL, packages
- 3) More on Ext JS application, MVC, real life example.
- 4) Using OEM
- 5) Spring MVC CRUD in action.
- 6) JSON parsing
- 7) Ext JS tool, tips & tricks.