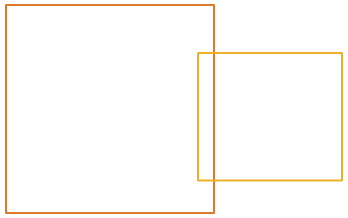
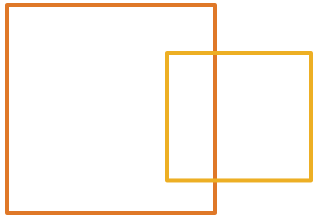
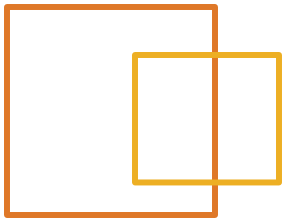
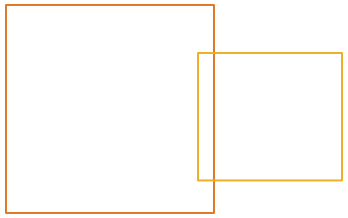
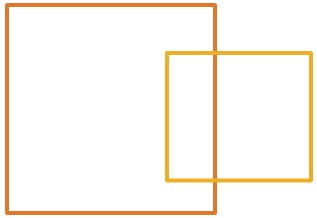
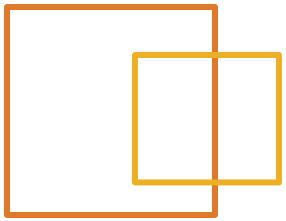





Fast Track to Java

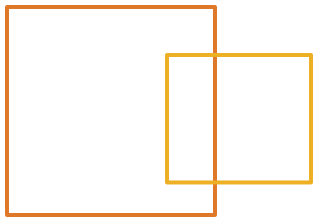
Customized for Starbucks

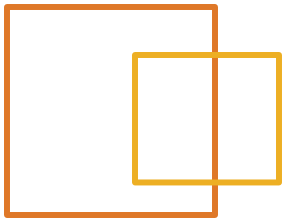
Delivered by DevelopIntelligence


XML Handling & Web Services

Objectives

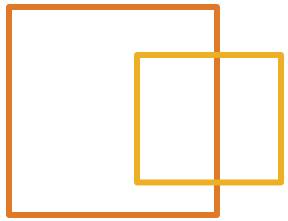




At the end of this module you should be able to:

- Manipulate XML data using:
  - SAX
  - DOM
  - JAX-B
- Create a REST Web Service using JAX-RS
- Write a client for a REST Web Service
- Write a SOAP Web Service using JAX-WS
- Write a SOAP WS Client using JAX-WS

Major XML APIs in Java



- `javax.xml.parsers.SAXParser`
  - Input only (cannot write XML documents out)
  - Lowest memory footprint option (document not stored)
  - Error handling may be managed by client
  -
- `javax.xml.parsers.DocumentBuilder`
  - Entire document represented in memory
  - Traverse nodes
  - Insert, delete, modify nodes
  - Output supported using Transformer

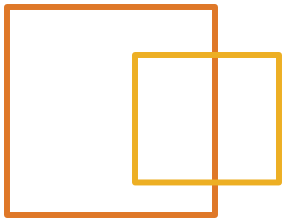
Major XML APIs in Java

- Build a Java data object from the input
  - JAX-B Java API for XML Binding
  - Converts XML data into Java Objects
  - Creates Java classes to suit XSD
  - Converts Java objects into XML data
  - Generates/consumes XSD

Stream Processing With SAX

- Callback/event oriented processing
- `SAXParserFactory` creates new parser instance
- Connect the parser to the input document
- Get a callback for each token parsed
  - Many callbacks can be generated, use an adaptor class to simplify listener implementation

SAXParser Example



- Start the parser

```
public static void main(String[] args)
  throws Throwable {

  FileInputStream fis =
    new FileInputStream("something.xml");
  InputSource xis = new InputSource(fis);

  SAXParser parser = SAXParserFactory.newInstance()
    .newSAXParser();

  parser.parse(xis, new MySaxHandler());
}
```

SAXParser Example

- Key callbacks
- `startElement`, `endElement`—indicate
  - <element> and </element>
  - `characters`—indicates text in the body of an element
  - `ignorableWhitespace`—might not care about this
  - `warning`, `error`, `fatalError`—report problems with parsing
- Parameters provided with the callbacks vary based on what's being described

SAXParser Example

```
public class MySaxHandler extends DefaultHandler {
  @Override public void startElement(String uri,
    String localName, String qName, Attributes atts)
    throws SAXException {
    System.out.println("startElement " + uri + " "
      + localName + " " + qName + " " + atts);
  }
  @Override public void endElement(String uri,
    String localName, String qName)
    throws SAXException {
    System.out.println("endElement " + uri + " "
      + localName + " " + qName);
  }
[...]
```

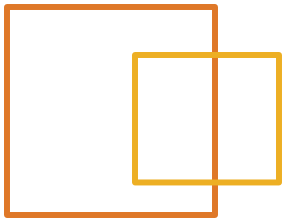
DOM Parser Example

```
FileInputStream fis = new
FileInputStream("something.xml");

DocumentBuilder db =
  DocumentBuilderFactory.newInstance()
  .newDocumentBuilder();

Document d = db.parse(fis); // d is root Node

processNode(d, 0); // investigate the document tree
```

DOM Parser Example

```
public static void processNode(Node n, int level) {
  System.out.println(indent(level) + ""
    + n.getNodeName());
  NodeList nList = n.getChildNodes();
  int count = nList.getLength();
  for (int i = 0; i < count; i++) {
    processNode(nList.item(i), level + 1);
  }
}
public static String indent(int level) {
  String[] spaces = {"", "  ", "    ", "      ",
    "        ", "          ", "            "};
  if (level < spaces.length) { return spaces[level]; }
  else {
    return spaces[spaces.length - 1] + indent(level - spaces.length + 1);
  }
}
```

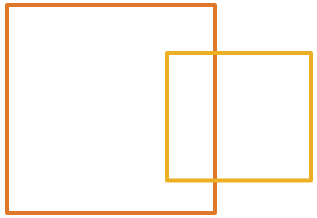
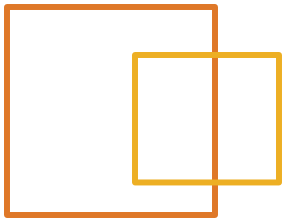
Updating A DOM Tree



```
// ask the Document object to create new element
// for that document (nodes may not be freely
// interchanged between Documents
Element c1 = d.createElement("Something-New");
// Set the text content of the element
c1.setTextContent("Something new in the document");
// add this new node to the end of an existing node
existingNode.appendChild(c1);
// Another new element
Element c2 = d.createElement("Something-Borrowed");
// put an attribute into this node
c1.setAttribute("item-color", "Blue");
existingNode.appendChild(c2);
```

Writing An XML Document

```
TransformerFactory transformerFactory =
  TransformerFactory.newInstance();

Transformer transformer =
  transformerFactory.newTransformer();

// d is our Document
DOMSource source = new DOMSource(d);

// Send XML representation to the console
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
```

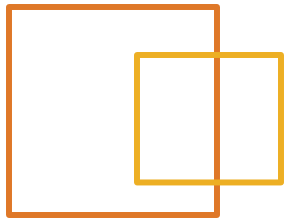




- JAX-B API provides for:
  - Reading an XML Schema definition (XSD) file and creating transformation code and a Java classfile
  - Reading a classfile and creating transformation code and XSD file
  - Parsing XML to create a Java object in memory
  - Creating XML representing a Java object in memory

Creating Java Types From XSD

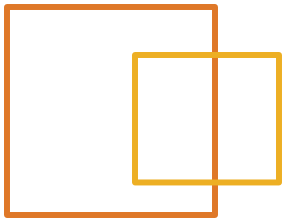
- Given a file data.xsd defining an XML specification of a data type “data”:
  - `xjc –p packageName data.xsd`
  - Creates Java files in the package `packageName`
  - Creates the data type(s) and supporting types, such as list types for sequences
  - Creates an object factory for creating instances of the Java data types
- `schemagen` tool creates xsd from Java class or source files

Example XSD Schema



```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
            jxb:version="2.0">

<xsd:element name="Greetings" type="GreetingListType"/>

<xsd:complexType name="GreetingListType">
  <xsd:sequence>
    <xsd:element name="Greeting" type="GreetingType"
                 maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Example XSD Schema

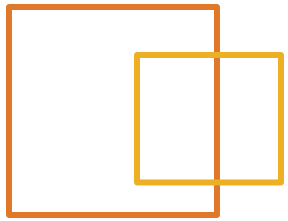


```
<xsd:complexType name="GreetingType">
  <xsd:sequence>
    <xsd:element name="Text" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="language" type="xsd:language"/>
</xsd:complexType>

</xsd:schema>
```

- To generate Java output, execute:

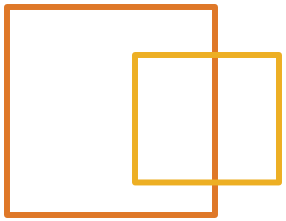
```
xjc –p greetingpkg theXsdFile.xsd
```

Generated Java



```
public class GreetingListType {
  protected List<GreetingType> greeting;
  public List<GreetingType> getGreeting() ...
}

public class GreetingType {
  protected String text;
  protected String language;
  public String getText() ...
  public void setText(String value) ...
  public String getLanguage() ...
  public void setLanguage(String value) ...
  public String toString() ...
}
```

Generated Java



```
public class ObjectFactory {
  public ObjectFactory() ...
  public GreetingListType createGreetingListType() ...
  public GreetingType createGreetingType() ...
  public JAXBElement<GreetingListType>
    createGreetings(GreetingListType value) ...
}
```

- These generated files are shown as skeletons, the files also include:
  - Imports
  - Method implementations
  - ***Annotations*** that tie these classes to the XML elements they represent

Using the Java Classes For Output

```
ObjectFactory of = new ObjectFactory();
GreetingListType grList =
  of.createGreetingListType();
GreetingType g = of.createGreetingType();
g.setText("Bonjour"); g.setLanguage("fr");
grList.getGreeting().add( g );
g = of.createGreetingType(); // create a second entry
g.setText("Gday"); g.setLanguage("en_AU");
grList.getGreeting().add( g );
JAXBElement<GreetingListType> gl =
  of.createGreetings(grList);
JAXBContext jc =
  JAXBContext.newInstance("greetingpkg");
Marshaller m = jc.createMarshaller();
m.marshal(gl, System.out);
```

Using the Java Classes For Input

```
ObjectFactory of = new ObjectFactory();
JAXBContext jc = JAXBContext.newInstance("customers");
Unmarshaller um = jc.createUnmarshaller();
File f = new File("input.xml");
JAXBElement jaxbe = (JAXBElement)(um.unmarshal(f));
CustomerListType customers =
  (CustomerListType) jaxbe.getValue();
List<CustomerDefType> custs = customers.getCustomer();

for (CustomerDefType cust : custs) {
  System.out.println("customer is:   " + cust.getName()
    + "\n at:           " + cust.getAddress1()
    + "\n joined:       " + cust.getJoined()
    + "\n credit limit: " + cust.getCredit());
}
```

XML To Java Type Conversions

- XML numerics should indicate type:

```
<xsd:element name="val" type="xsd:type-info"/>
```

- Type representation in Java:
  - xsd:decimal → `BigDecimal`
  - xsd:integer → `BigInteger`
  - xsd:long → `long`
  - xsd:int → `int`
  - xsd:short → `short`
  - xsd:byte → `byte`

XML To Java Type Conversions

- Unsigned types have larger maximum values, so need larger holders
  - xsd:nonNegativeInteger → `BigInteger`
  - xsd:unsignedLong → `BigInteger`
  - xsd:unsignedInt → `long`
  - xsd:unsignedByte → `short`
  - xsd:date, xsd:time and xsd:dateTime
- → `XMLGregorianCalendar`
  - XMLGregorianCalendar is abstract:

```
df = DatatypeFactory.newInstance()
df.newXMLGregorianCalendarDate(fields);
```

XML To Java Type Conversions

- xsd:list → List<sometype>

```
<xsd:simpleType name="NumberListType">
  <xsd:list itemType="xsd:int"/>
</xsd:simpleType>
```

- Yields:

```
public class ListsType {
  protected List<Integer> numbers;
  public List<Integer> getNumbers
...
```

- Avoid `<xsd:list itemType="xsd:string"/>` as this will generate ambiguous XML
  - Space-separated list looks like spaces in single string element

XML To Java Type Conversions

- Range significant-digits, string-length, and pattern-matching limits are supported in XSD, but are not enforced in JAXB generated Java code
- Nillable types are converted to wrappers
  - `<xsd:element name="s" type="short" nillable="true">` → `Short`
- Fields with Java keyword names get leading underscore
  - `<xsd:element name="long" …>` → `long` **`_long`**

Creating XSD From Java Types

- Create Java class representing desired XML data
- Annotate class using `javax.xml.bind.annotation.XmlType`
- Annotate fields using `javax.xml.bind.annotation.XmlElement`
- Generate the schema using:
- `schemagen package.AccountInfo`

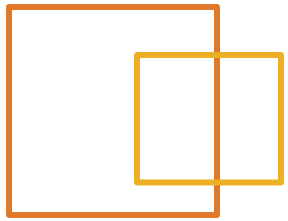
Sample JAX-B Annotations

```
@XmlType
public class ComplexType {
  private String name;
  private int number;
  private ComplexType otherCT;
  private String [] greetings;

  public ComplexType() {} // other constructors as needed

  @XmlElement public String getName() { return name; }
  public void setName(String name) {this.name = name;}

  @XmlElement public int getNumber() { return number; }
  public void setNumber(int num) {number = num;}
```

Sample JAX-B Annotations

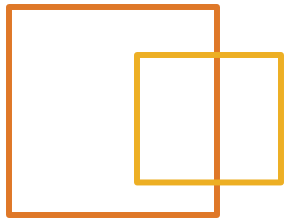
```
  // May also return List<String>
  @XmlElement public String[] getGreetings() {
    return greetings;
  }
  // may have set method for greetings property

  // Circular references will cause errors when generating
  // the XML, but references to complex types are ok
  @XmlElement public ComplexType getOtherCT() {
    return otherCT;
  }
  // may have set method for otherCT property
}
```

JAX-B Compatible Types



- Term JAX-B compatible is really a misnomer
- XSD places some restrictions on objects that may be represented in XML. For Java:
  - No RMI Remote objects
  - No cyclic object graphs (no circular references)
  - Be careful with Collections or other classes you do not know the runtime structure of.
- Notice the restriction of cyclic graphs is a runtime issue, an object *may* have a member of its own type

RESTful Web Services



- Data-structure oriented rather than behavior oriented
  - Not OO, assumes client knows how to handle data
  - Data may be sent in XML, JSON, or other forms
- URL should describe a resource not a request
  - REST WS should be cacheable
- HTTP methods create database like access
  - POST creates resource
  - GET reads resource
  - PUT updates resource
  - DELETE deletes resource

RESTful Web Services With JAX-RS

- JAX-RS provides APIs for REST web services
  - Not included in JAVA SE
  - Reference implementation (“Jersey”) available from http://jersey.java.net
  - Supports both server and client side
  - Ensure JAR files are on path
- Server side is often deployed in a container
  - Jersey provides stand-alone implementation
- Annotation based
  - Requires Java 1.5 or greater

Creating a REST Web Service

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;

@Path("/helloworld/{id}")
public class HelloWorldResource {
  @GET @Produces("text/plain")
  public Response getMessage(@PathParam("id")int id){
    return Response.ok("id is " + id).build();
  }
}
```

Launching The Web Service

- Container-hosted services will have their own deployment
- For Jersey:

```
public class Main {
  private static URI getBaseURI() {
    return
      UriBuilder.fromUri("http://localhost/").port(9998).build();
  }
  public static final URI BASE_URI = getBaseURI();
  protected static HttpServer startServer() throws IOException {
    ResourceConfig rc = new PackagesResourceConfig("resources");
    return GrizzlyServerFactory.createHttpServer(BASE_URI, rc);
  }
```

Launching The Web Service

```
  public static void main(String[] args)
    throws IOException {
    HttpServer httpServer = startServer();
    System.out.println(
      String.format("Jersey app started “
      + "with WADL available at "
      + "%sapplication.wadl\n"
      + "Try %shelloworld\nHit enter to end.",
      BASE_URI, BASE_URI));
    System.in.read();
    httpServer.stop();
  }
}
```

JAX-RS Server Class Annotation

- `@Path("/helloworld/{id}")`
  - Annotates the class providing service
  - Specifies the path by which this will be accessed
  - Relative to the deployment for the whole service
  - {xx} indicates a variable part of the path

```
@Path("/helloworld/{id}")
public class HelloWorldResource
```

JAX-RS Data Type Annotations

- Data type may be specified at the class or the method level, or both
  - E.g. “text/plain” or MediaType class provides constants
  - Class level creates a default
  - Method declaration overrides the default
- `@Produces("text/plain", "...")`
  - Specifies the MIME type(s) offered to the client
- `@Consumes("...")` lists accepted **input** types

```
@GET @Produces("text/plain")
public String getMessage …
```

JAX-RS Server Method Annotations

- @GET, @POST, @PUT, @DELETE
  - At least one method in the class annotated with @Path must have one of these annotations
  - Only one of these may be used on a single method
  - Indicates a service method, specifying the HTTP method it should respond to

JAX-RS Parameter Annotations

- Client originated data may be passed into the service method via arguments
- `… method(@PathParam("id") int id)`
  - `PathParam` pastes the variable part from the first example into the method
  - In this case originating from/matching with `@Path("/helloworld/{id}")`
  - Will respond to http://myserver/helloworld/albert by setting method parameter id to “albert”

```
getMessage(@PathParam("id") int id) {
```

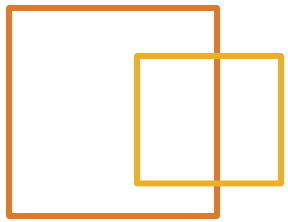
JAX-RS Parameter Annotations

- Additional parameter annotations exist for passing data into methods
- …`method(@`**`QueryParam`**`("author") String author)`
  - Injects a query param (…`/service?author="Fred"`) into the service method
- …`method(@`**`FormParam`**`("count") String count)`
  - Injects a form param (…`/service?author="Fred"`) into the service method

JAX-RS Parameter Annotations

- **`@CookieParam`**
  - Sends a cookie value to the service method
- **`@HeaderParam`**
  - Sends an HTTP header value to the service method

@Context Annotation



- If the service is configured in a container environment, the @Context annotation is available
  - `@Context` injects a variety of context items including:
  - `UriInfo`
  - `HttpHeaders`
  - `ServletConfig`
  - `ServletContext`
  - `HttpServletRequest`
  - `HttpServletResponse`

Response Features

- `WebMethod` can simply return `String`
- `Response` object offers more control of response
- Construct using factory methods in the inner class: `Response.ResponseBuilder`
  - Response.ok(<content>) creates a ResponseBuilder
- `ResponseBuilder` provides static methods that modify the response in preparation, e.g.:
  - `status(<code>)`
  - `header(<headername>, <value>)`

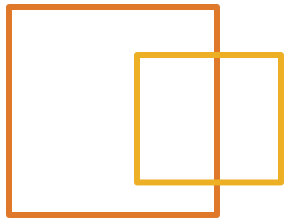
Response Features

- `ResponseBuilder.build()` method constructs the `Response` object ready for returning from the `@WebMethod`
- To generate a resource not found (404) response with non-empty body:

```
return Response.ok("Not found")
  .status(404).build();
```

- (Docs mention factory method `Response.entity(<content>)` but current implementations lack this)

POST Service Example



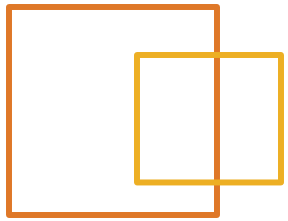
```
@POST
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public String postMessage(
  @FormParam("something") String something) {
  return "Something has the value\n" + something;
}
```

Returning XML From A REST Service

- Create a class that defines the data structure to be returned
- Apply JAX-B annotations to it
  - Ensure the class itself is annotated @XmlRootElement
- Define the web method as:
  - `@Produces(“application/xml”)`
  - Returns a `Response`
  - Code the `ResponseBuilder` as:

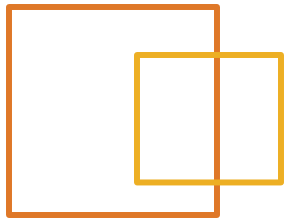
```
Response.ok(new MyJaxBThing(<args>)).build()
```

Example JAX-B Structure



```
import java.util.Date;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement public class DataStructure {
  @XmlElement public String name; // may be private
  @XmlElement public int value;   // with accessors
  @XmlElement public Date today;  // & mutators
  public DataStructure() {} // Need zero arg const.
  public DataStructure(String name, int value) {
    this.name = name;
    this.value = value;
    this.today = new Date();
  }
}
```

Example XML Producer

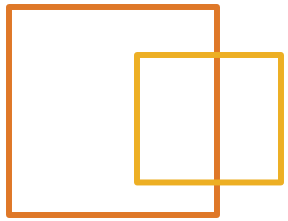


```
@GET
@Produces("application/xml")
public Response getXmlMessage(
    @QueryParam("idx") int idx) {

  DataStructure rv =
    new DataStructure("AsText " + idx, idx * 2)

  return Response.ok(rv).build();
}
```

What Goes Into The XML?

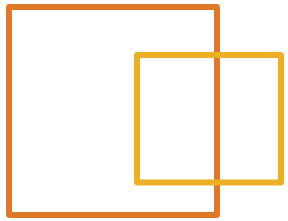
- The `@XmlRootElement`
  - Plus XSD primitive members of it
  - Plus `@XmlElement` members of it even if complex
- XSD primitive members of complex members of the root get all their primitive members
- And their `@XmlElement` members
- In effect, follow `@XmlElement` through each `@XmlType`, and include primitives and one more level of object that's not annotated @XmlType
- Generally, annotate `@XmlElement` / `@XmlType` the items you want to see

Creating JSON Output



```
@GET
@Produces("application/json")
public Response getXmlMessage(
    @QueryParam("idx") int idx) {

  DataStructure rv =
    new DataStructure("AsText " + idx, idx * 2)

  return Response.ok(rv).build();
}
```

Creating A REST Client



- JAX-RS is a server-side API
  - Jersey project provides a client API
- Client can be created using `URLConnection` and `HttpURLConnection`

GET Client With HttpURLConnection

```
URL serverURL =
  new URL("http://localhost:9998/helloworld/12345");
HttpURLConnection conn =
  (HttpURLConnection)(serverURL.openConnection());
conn.setRequestMethod("GET");
conn.setReadTimeout(10000);
conn.connect();
BufferedReader rd  = new BufferedReader(
  new InputStreamReader(conn.getInputStream()));
for (String ln; (ln = rd.readLine()) != null);) {
  System.out.println(ln + '\n');
}
conn.disconnect();
```

POST Client With HttpURLConnection

```
URL serverURL =
  new URL("http://localhost:9998/helloworld/12345");
HttpURLConnection conn =
  (HttpURLConnection)(serverURL.openConnection());
conn.setRequestMethod(“POST");
conn.setDoOutput(true);
conn.setReadTimeout(10000);
conn.connect();
DataOutputStream dos =
  new DataOutputStream(conn.getOutputStream());
dos.writeBytes("something="
  + URLEncoder.encode("A value", "UTF-8"));
// get input stream and read response as GET example
```

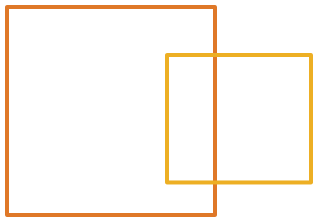
Using The HttpURLConnection

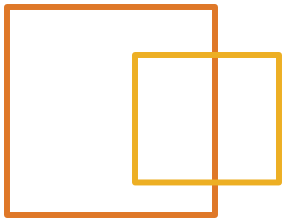
- After `c.connect()`, `c.getResponseCode()`
- After receiving an error code (e.g. 404)
  - `getInputStream()` throws `FileNotFoundException`
  - `c.getErrorStream()` gets content instead

Using The Jersey Client API

- Jersey has a Client API which can make your life a bit easier
- Not a standard
- Good introduction at http://jersey.java.net/nonav/documentation/latest/client-api.html
- Example in **RestClient-HTTP** project.

Lab 11





- REST Web Services
  - Convert the File Information Server from the last lab to expose the same functionality as a REST web service.
  - Convert the File Information Client to access the service as a REST client. You can use any client API you like.
  - Solution: RESTFileLab

Using JAX-WS

- JAX-WS provides for SOAP type web services
  - Historically several products, tools, and APIs have provided this with varying levels of complexity and standardization
- JAX-WS currently provides for WS-I Basic Profile 1.1
  - Therefore supports non-Java clients and services
- Annotation based
- Creates clients and servers

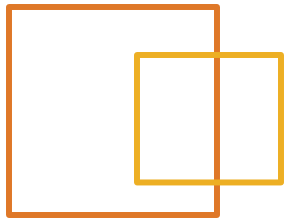
Design Approaches

- Create client support code from WSDL & XSD
  - `wsimport`
- Create WSDL & XSD from a Java implementation of a Web Service
  - Automatic on deployment / `wsgen`
- Create skeleton code to support implementation of a Web Service in Java starting from WSDL & XSD
  - `wsimport`

Creating A New WebService In Java

```
@WebService public class SmartRemark {
  private String [] remarks = {
    "Imagination is more important than knowledge.",
    "Quidquid Latine dictum sit altum videtur!"
  };

  @WebMethod public String getRemark() {
    int idx = (int)(Math.random() * remarks.length);
    return remarks[idx];
  }
}
```

Publishing A Web Service



- Options include:
  - In a Web Container
  - From a Stateless Session Bean in an EJB container
  - In Java SE directly
- Benefits of Web and EJB containers:
  - Declarative security control
  - Management/monitoring features
  - Capacity/efficiency

Publishing A WebService In Java SE

```
public class Publisher {
  public static void main(String[] args) {
    String theURL =
      "http://localhost:8888/ws/server";

    Endpoint.publish(theURL, new SmartRemark());
    System.out.println("Service is published!");
  }
}
```

Accessing Deployed WSDL

- Following deployment, WSDL is automatically generated and published:
  - In this example:

```
http://localhost:8888/ws/server?wsdl
```

Creating A Java WS Client

- Generate supporting artefacts from WSDL:

```
wsimport –keep –p <package> <wsdl-url>
```

- Resulting classes provide:
  - Java `interface` defining the service methods
  - Factory class for creating the port / stub
  - JAX-B annotated classes for arguments, returns, exceptions (faults), and exception details (as JavaBeans)
  - `ObjectFactory` for creating objects of supporting types

Working With Eclipse

- Eclipse doesn't like "foreign" classes
  - Run `wsimport` with the `–keep` option, in a *different directory tree* entirely
  - Delete all the `.class` files that it generates
  - Create the destination package in the project in Eclipse
  - Right-click on the package in Eclipse, then select Import
  - In the wizard, open the "General" folder and choose File System, hit "Next"
  - Browse to & select the generated package directory
  - Select the checkbox for that directory & hit "Finish"

Creating A Java WS Client

- Find the class that offers the method:
  - **get**Xxxx**Port**()
- Create the port and call methods on it:

```
public class RemarkClient {
  public static void main(String[] args) {
    SmartRemark remark =
      new SmartRemarkService().getSmartRemarkPort();
    System.out.println("Smart Remark is “
      + remark.getRemark());
  }
}
```

WS Arguments & Return Types

- JAX-WS permits complex arguments & returns
  - They must be XSD compatible (aka JAX-B compatible)
- These do not need to be JAX-B annotated; `wsgen`/`wsimport` creates the JAX-B types as needed
- `wsimport` will represent `List` or array elements using a mutable `List`
  - Only a `List<?> getXxxx()` method will be provided, expect to modify the provided list, not replace it

Exceptions From Web Methods

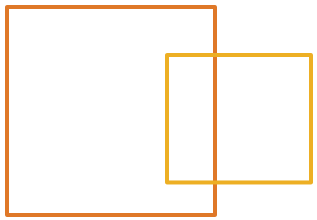
- Exceptions may be thrown from web methods
- Exception classes should be XSD compatible
- wsgen/wsimport create exception types that use a JavaBean to represent the exception data
- This JavaBean property is called “faultInfo”, generally has a property “message”
  - `exception.getFaultInfo().getMessage()`

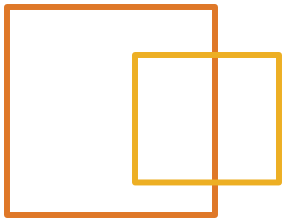
Summary

In this module, we covered:

- Manipulate XML data using:
  - SAX
  - DOM
  - JAX-B
- Create a REST Web Service using JAX-RS
- Write a client for a REST Web Service
- Write a SOAP Web Service using JAX-WS
- Write a SOAP WS Client using JAX-WS

Lab 12





- JAX-WS Web Services
  - Convert the File Information Server from the last lab to expose the same functionality as a JAXW-WS web service.
  - Convert the File Information Client to access the service as a JAX-WS client.
  - Solution: JAX-WSFileLab