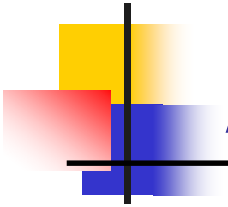
Digging into the Web with a Tiger: Lower-level Data Manipulation with J2SE 5.0 – Part 2

Neil Graham

Manager, XML Parser Development

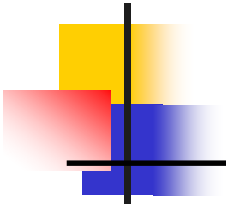
IBM Canada Ltd.

neilg@ca.ibm.com



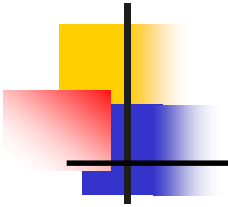
Agenda

- Evolution of XML Schema
- XML Schema to Java type mapping
- JAXP 1.3 validation API
- JAXP 1.3 XPath API



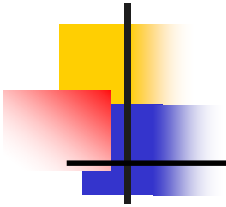
Evolution of XML Schema

- XML Schema 1.0, 1st Edition: 2nd May, 2001
- Endorsed by JAXP since 1.2
- 2nd Edition: Entered proposed edited Recommendation Review March 18, 2004
- JAXP endorses XML Schema 1.0 with stable errata



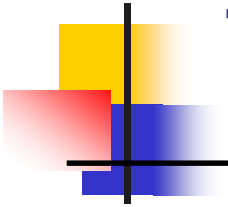
Significant XML Schema 1.0 Errata

- xs:gMonth: formerly, "—01—" was valid; now only "—01" will be permitted
- xs:lang: the pattern has been changed to align with XML 1.0 2nd edition
- Xs:anyType: Originally, was specified to have processContents="strict"; now processContents "skip" for derivation, "lax" for validation
  - No one actually implemented xs:anyType as originally specified; a significant change, but only at an abstract level



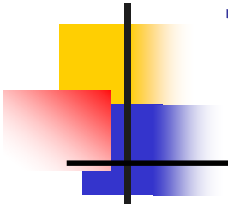
Java to XML Schema Mapping

- Schema datatypes: Widely accepted — even part of Relax NG
- Also highly useful in Web services, JAXRPC, JAXB, and XPath 2.0
- Was time to fix a mapping between datatypes defined by XML Schema and Java classes
- For most types, this is straightforward



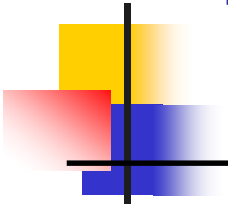
Java to XML Schema Mapping — Simple Cases

- Java BigDecimal <-> xs:decimal
- Java BigInteger: xs:integer, xs:positiveInteger, xs:negativeInteger, xs:nonPositiveInteger, xs:nonNegativeInteger
- Java double/float <-> xs:double/xs:float
- Java String: xs:anySimpleType, xs:string, xs:normalizedString, xs:token, xs:tokens, xs:name, …



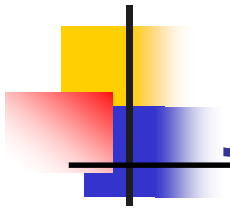
Java to XML Schema Mapping — Simple Cases *(Continued)*

- Java long: xs:long, xs:unsignedLong, xs:unsignedInt
- Java int: xs:int, xs:unsignedShort
- Java short: xs:short, xs:unsignedByte
- Java byte: xs:byte
- Java byte[]: xs:base64Binary, xs:hexBinary
- Java boolean: xs:boolean



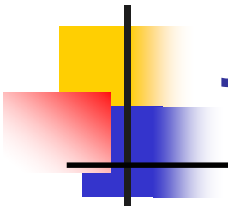
Java to XML Schema Mapping: Involved Cases

- javax.xml.namespace.QName: maps to both xs:qname and xs:notation
- javax.xml.datatype.XMLGregorianCalendar: xs:gDay, xs:gMonth, xs:gMonthDay, xs:gYear, xs:gYearMonth, xs:time, xs:dateTime, xs:date
- javax.xml.datatype.Duration: xs:duration
- Duration is also specified to map to the XQuery 1.0/XPath 2.0 xdt:dayTimeDuration and xdt:yearMonthDuration datatypes



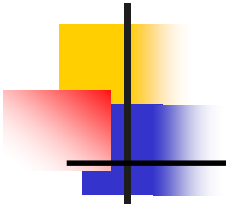
javax.xml.datatype Overview

- Like other JAXP packages, has a factory to enable implementation pluggability
- DatatypeFactory: an abstract class with a static newInstance() method to enable correct concrete implementation to be invoked
- DatatypeConfigurationException will be thrown if this fails



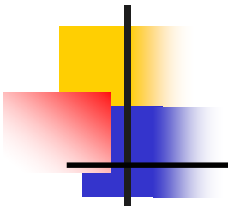
javax.xml.datatype Overview *(Continued)*

- Duration and XMLGregorianCalendar objects may then be created in various ways
- DatatypeConstants: defines constants for months, comparison results, fields for Duration and QNames for Schema types mapped



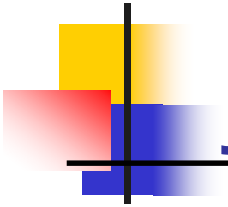
Producing Duration Objects

- Duration objects may be created from
  - Lexical representations
  - Totals in milliseconds (long)
  - Sequences of a boolean indicating positive or negative, years, months, days, hours, minutes, seconds
  - This last may have totals specified as BigInteger or int
- Durations are immutable



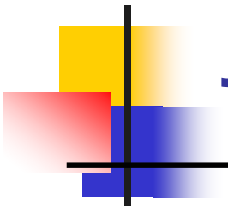
Producing Duration Objects
(Continued)

- DatatypeFactory also contains sets of newDurationDayTime and newDurationYearMonth methods
- These allow creation of XQuery/XPath-compatible Durations for xdt:durationYearMonth and xdt:durationDayTime
- Implemented as wrappers for newDuration methods; irrelevant parameters eliminated
- *e.g.*, newDurationYearMonth(boolean isPositive, int year, int month): Duration



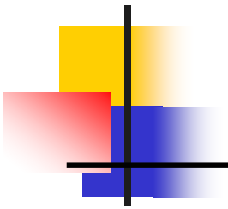
javax.xml.datatype.Duration

- Duration.Equals: carefully defined to align with definition in XML Schema
- Methods provided to determine if one Duration is shorter than another, and to compare two to one another
- Methods also provided to add or subtract two Durations, and to multiply a Duration by some value



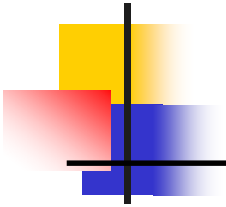
javax.xml.datatype.Duration
(Continued)

- Durations may also be added to Calendars or Dates
- Nuration#normalizeWith(Calendar): produces a new Duration with same length with its days field normalized with respect to the parameter



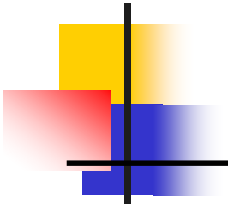
The XMLGregorianCalendar Class

- XMLGregorianCalendar fields (timezone, year, months, … second, fractionalSecond) may be set at will
- Methods provided to normalize and validate the object
- May convert them to GregorianCalendar objects
- May also add a Duration to them



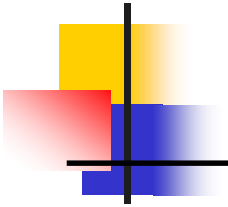
Datatypes Example

```
DatatypeFactory df =
  DatatypeFactory.newInstance();
// my work number in milliseconds:
Duration myPhone = df.newDuration(90541335191);
// my approximate lifespan up to the conference
Duration myLife = df.newDuration(
    true, 29, 2, 15, 13, 45, 0);
int compareVal = myPhone.compare(myLife);
switch (compareVal) {
case DatatypeConstants.LESSER:
    // uninteresting switch on comparison value
}
```



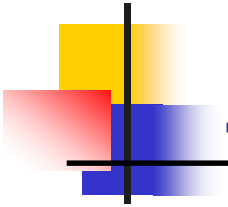
Datatypes Example *(Continued)*

```
XMLGregorianCalendar xgc =
  df.newXMLGregorianCalendar();
xgc.setYear(1975);
xgc.setMonth(DatatypeConstants.AUGUST);
xgc.setDay(11);
xgc.setHour(6);
xgc.setMinute(44);
xgc.setSecond(0);
xgc.setMillisecond(0);
xgc.setTimezone(5);
xgc.add(myPhone);
// print where phone number ends
```



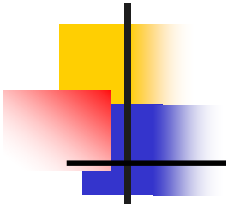
Datatypes Example Output

```
There are fewer milliseconds in my phone
  number
than my lifespan.
The approximate end of the number of
  milliseconds in
my phone number was
1975-11-24T01:46:13.519+00:05
```



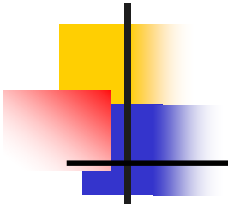
JAXP 1.3: Validation API

- Grammars tend to be expensive to parse
- One wants to be able to reuse them for multiple documents
- Would also be nice to validate a generic DOM level 2 tree or a stream of SAX events — say dynamically generated from an XSL transformation
- This API allows for all of this



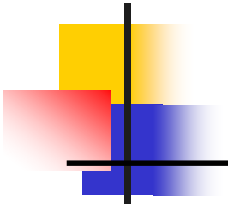
The Schema Abstract Class

- Represents the result of processing a particular grammar (*e.g.*, set of XML Schema documents)
- Immutable (and therefore thread-safe)
- Used for creating Validators (and ValidatorHandlers) to validate documents
- No facilities provided for introspecting the grammar (yet!)



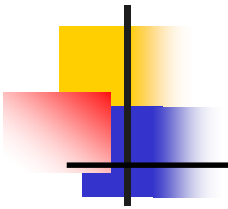
SchemaFactory

- A factory for creating Schema objects
- static newInstance(String schemaLanguage): parameter describes the schema language for which a factory is to be created
- JAXP 1.3 defines the URI to be used for Relax NG, but only XML Schema must be supported
- To support multiple schema languages, factory lookup mechanism is more complicated than for DOM/SAX



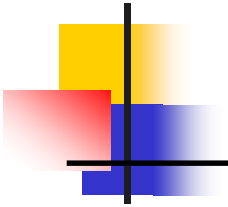
SchemaFactory Lookup Mechanism

- All class loading uses context class loader
- Attempt to load class referred to by system property "javax.xml.validation.SchemaFactory:schemaLanguage"
- schemaLanguage is same as in argument to newInstance; for XML Schema, the relevant property would be "javax.xml.validation.SchemaFactory:http://www.w3.org/2001/XMLSchema"



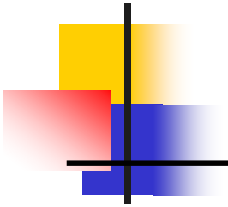
SchemaFactory Lookup Mechanism *(Continued)*

- If not found/not successful, file $JAVA_HOME/lib/jaxp.properties will be read and the value of a key identical to the above system property will be treated as a class name, if such a key is present
- All jars on the class loader's CLASSPATH will have their META-INF/services directories inspected for a file named "javax.xml.validation.SchemaFactory"



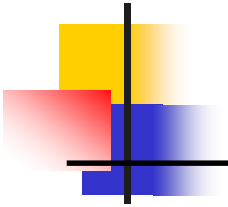
SchemaFactory Lookup Mechanism *(Continued)*

- If one is found, its contents will be interpreted as a class name
- A new instance of this class will be returned only if its isSchemaLanguageSupported(String) method returns true for this schemaLanguage
- If all else fails, a system default will be returned, if one is defined for this schemaLanguage



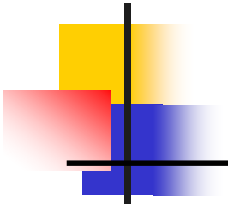
SchemaFactory *(Continued)*

- It's also possible to set features and properties on SchemaFactory objects (*e.g.*, to enable Xerces-J's schema-full-checking feature)
- An LSResourceResolver can also be attached (*e.g.*, to help with schema imports)
- A SAX ErrorHandler can be set to pick up schema construction errors



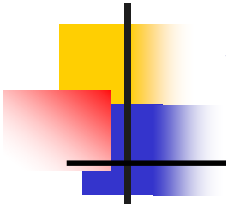
SchemaFactory *(Continued)*

- **N.B.: if no ErrorHandler is attached, any error will cause an exception to be thrown!**
- Schemas can be created from Source, File, or URL objects, or for arrays of Sources (*e.g.*, to forestall necessity of registering an LSResourceResolver)



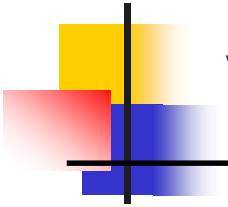
SchemaFactory *(Continued)*

- newSchema(): creates a special Schema; for XML Schemas, this simply follows schemaLocation hints in documents
  - Previously compiled grammars will be used when a schemaLocation is encountered that's identical to one previously found
  - (In Xerces-J, this is called "passive" grammar caching)



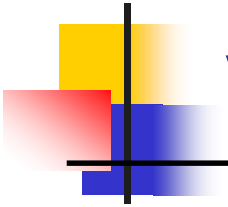
Validator

- Validates some (SAX or DOM) Source, optionally producing a (SAX or DOM, respectively) Result with infoset augmentations
- *e.g.*, default attributes will be filled in
- These objects are not thread-safe
- As with most JAXP objects, features and properties may be set/queried



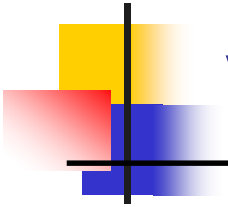
Validator *(Continued)*

- An LSResourceResolver may be attached (to deal with schemaLocation hints referring to schemas outside the knowledge of the Validator, for instance)
- An ErrorHandler should also be set
- As with SchemaFactory, if no ErrorHandler is set, all errors will result in SAXParseExceptions being thrown
- In deference to the parser/transform packages, StreamSources and StreamResults are not handled



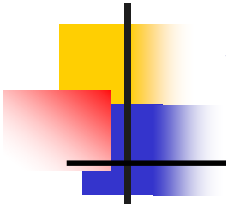
ValidatorHandler

- Validates a stream of SAX2 events by acting as a ContentHandler
- A ContentHandler implementation can be set, in which case it acts as a filter, augmenting events appropriately
- An LSResourceResolver may be set, an ErrorHandler should be set



ValidatorHandler *(Continued)*

- When acting as a filter, most events (*e.g.*, startElement, endElement) will not be buffered, and exceptions will be rethrown — guarantees minimal latency
- A TypeInfoProvider implementation may be queried from a ValidatorHandler

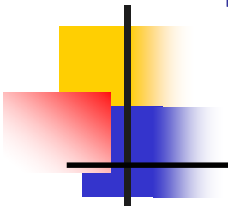


ValidatorHandler *(Continued)*

- TypeInfoProviders are immutable, long-lived objects implementing DOM l3 TypeInfo interface for the in-scope element or a specified attribute
- Enables a fully-conformant DOM level 3 Core implementation to be based on a ValidatorHandler
- TypeInfoProviders also afford a means to query if an attribute is of type ID, or whether it was faulted in from the schema

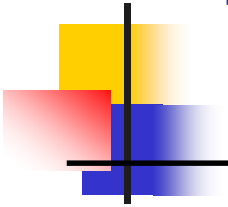
JAXP Validation Example: Simple Instance

```
<inventory xmlns="http://www.booze.com">
  <product sku="b01" amount="4000"
     price="0.25" cost="0.10">
    bad beer
  </product>
  <product sku="b03" amount="250" price="1.00"
     cost="0.50">
    good beer
  </product>
```



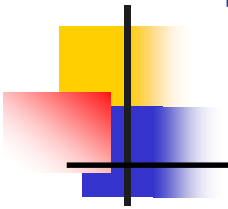
JAXP Validation Example: Simple Instance *(Continued)*

```
<product sku="s05" amount="100"
     price="20.00" cost="12.00">
    half-decent Scotch
  </product>
 <assets cash="5400.00" inventory="3250.00"/>
</inventory>
```



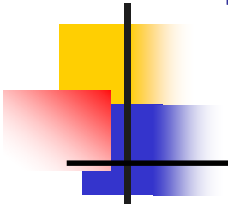
JAXP Validation Example: Schema

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.booze.com"
  xmlns="http://www.booze.com">
<xsd:simpleType name="moneyType">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="0.00"/>
    <xsd:fractionDigits value="2"/>
  </xsd:restriction>
</xsd:simpleType>
```



JAXP Validation Example: Schema *(Continued)*

```
<xsd:element name="inventory">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="product"
          maxOccurs="unbounded"/>
      <xsd:element ref="assets"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

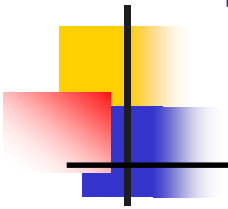


JAXP Validation Example: Schema *(Continued)*

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
    <xsd:attribute name="cost" type="moneyType"/>
    <xsd:attribute name="price"
                  type="moneyType"/>
    <xsd:attribute name="sku" type="xsd:ID"/>
    <xsd:attribute name="amount"
                  type="xsd:nonNegativeInteger"/>
      </xsd:extension>
```

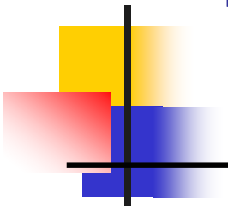
JAXP Validation Example: Schema *(Continued)*

```
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="assets">
    <xsd:complexType>
      <xsd:attribute name="cash"
                     type="moneyType"/>
      <xsd:attribute name="inventory"
                     type="moneyType"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



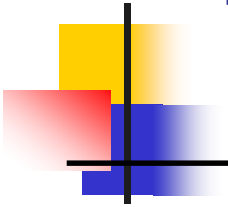
JAXP Validation Example: Code

```
public class ValidationTest implements
   ErrorHandler {
 public static void main (String []
  args) throws Exception {
   ValidationTest test = new ValidationTest();
   // create a SchemaFactory
  System.out.println(
     "creating SchemaFactory instance");
 SchemaFactory sf = SchemaFactory.newInstance(
      XMLConstants.W3C_XML_SCHEMA_NS_URI);
 // in case there are any errors in the schema
 sf.setErrorHandler(test);
```



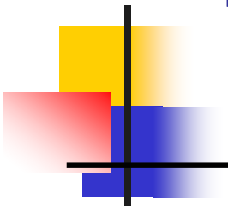
JAXP Validation Example: Code *(Continued)*

```
 // and create a schema from args[1]
System.out.println(
    "creating Schema instance from "
     + args[1]);
Schema s = sf.newSchema(new File(args[1]));
// create a DOM parser factory
DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
dbf.setSchema(s);
DocumentBuilder db = dbf.newDocumentBuilder();
db.setErrorHandler(test);
```



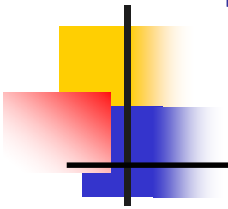
JAXP Validation Example: Code *(Continued)*

```
System.out.println("parsing " + args[0]);
  Document inventory = db.parse(args[0]);
  test.printStatus(inventory);
  // set ourselves up to do transactions
  DOMSource docSource = new
   DOMSource(inventory);
  Validator val = s.newValidator();
  val.setErrorHandler(test);
```



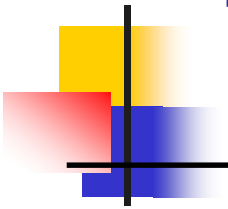
JAXP Validation Example: Code *(Continued)*

```
System.out.println(
 "transaction #1:  buy 1000 bad beer");
 if(test.transact(inventory, docSource, val,
   true, "b01", 1000)) {
  System.out.println(
   "transaction failed!  Bailing...\n");
  test.printStatus(inventory);
  System.exit(0);
 }
 test.printStatus(inventory);
// a second transaction omitted…
```



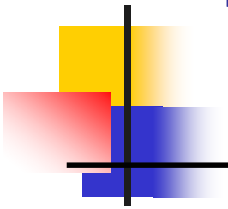
JAXP Validation Example: Code *(Continued)*

```
    System.out.println("Let's buy expensive
scotch 'til we go broke...");
 while(true) {
    if(test.transact(inventory, docSource, val,
        true, "s05", 400)) {
      System.out.println(
       "transaction failed!  Bailing...\n");
      test.printStatus(inventory);
      System.exit(0);
    }
    test.printStatus(inventory);
 }
```



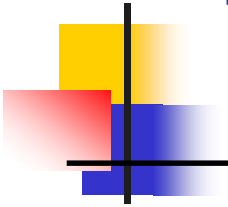
JAXP Validation Example: Code *(Continued)*

```
private boolean fError;
 public ValidationTest() {
   fError = false;
 };
 // SAX ErrorHandler methods just print
 // appropriate messages and set fError
 // printStatus is not interesting; a few DOM
 // l2 methods to traverse the tree
```



JAXP Validation Example: Code *(Continued)*

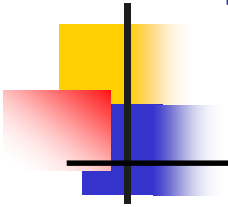
```
public boolean transact(Document doc,
        Source docSource,
        Validator validator,
        boolean weAreBuying,
        String sku, int amount) {
   Element target = doc.getElementById(sku);
   if(target == null )
     return false;
   Attr targetAmount =
    target.getAttributeNodeNS(null, "amount");
```



JAXP Validation Example: Code *(Continued)*

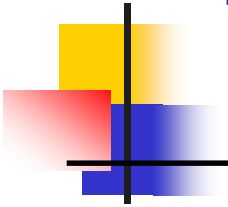
```
Attr targetPrice =
  target.getAttributeNodeNS(null, "price");
Attr targetCost =
  target.getAttributeNodeNS(null, "cost");

// use some knowledge of doc structure here...
Element assets =
  (Element)doc.getElementsByTagNameNS
  ("http://www.booze.com", "assets").item(0);
Attr inventoryValue =
  assets.getAttributeNodeNS(null,
  "inventory");
```



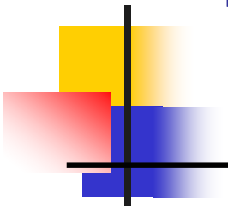
JAXP Validation Example: Code *(Continued)*

```
Attr cash = assets.getAttributeNodeNS(null,
   "cash");
if(weAreBuying) {
    BigDecimal totalCost = (new BigDecimal(
        targetCost.getValue())).multiply(
        new BigDecimal(amount));
   BigDecimal totalPrice = (new BigDecimal(
        targetPrice.getValue())).multiply(
        new BigDecimal(amount));
```



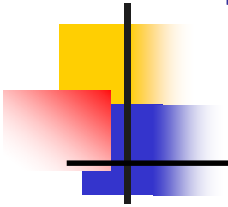
JAXP Validation Example: Code *(Continued)*

```
targetAmount.setValue(Integer.toString(
      Integer.parseInt(targetAmount.getValue())
      + amount));
      inventoryValue.setValue((new BigDecimal(
      inventoryValue.getValue())).add(
      totalPrice).toString());
cash.setValue((new BigDecimal(
      cash.getValue())).subtract(
      totalCost).toString());
```



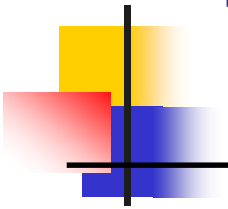
JAXP Validation Example: Code *(Continued)*

```
// similar for when selling; now validate:
try {
      validator.validate(docSource);
}
catch(SAXException e) {
      return false;
}
catch(IOException e) {
      return false;
}
return fError;
```



JAXP Validation Example: Output

```
creating SchemaFactory instance
creating Schema instance from ex6.xsd
parsing ex6.xml
Product SKU  Amount    Total Cost    Total Price
b01             4000        400.00        1000.00
b03              250         125.00         250.00
s05              100        1200.00        2000.00
Cash on hand:  5400.00; total value of inventory:
   3250.00
```

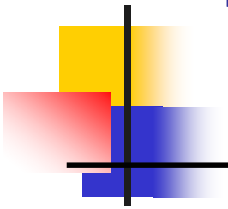


JAXP Validation Example: Output *(Continued)*

transaction #1: buy 1000 bad beer

Product SKU	Amount	Total Cost	Total Price
b01	5000	500.00	1250.00
b03	250	125.00	250.00
s05	100	1200.00	2000.00

Cash on hand: 5300.00; total value of inventory: 3500.00

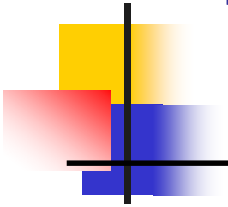


JAXP Validation Example: Output *(Continued)*

Let's buy expensive scotch 'til we go broke...

Product SKU	Amount	Total Cost	Total Price
b01	5000	500.00	1250.00
b03	0	0.00	0.00
s05	500	6000.00	10000.00

Cash on hand: 750.00; total value of inventory: 11250.00

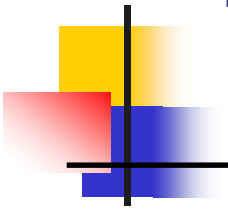


JAXP Validation Example: Output *(Continued)*

Error: http://www.w3.org/TR/xml-schema-1#cvc-minInclusive-valid?-4050.00&0.0&moneyType

Error: http://www.w3.org/TR/xml-schema-1#cvc-attribute.3?assets&cash&-4050.00&moneyType

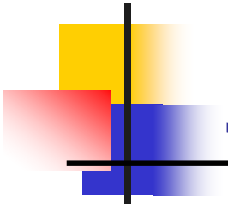
transaction failed! Bailing...



JAXP Validation Example: Output *(Continued)*

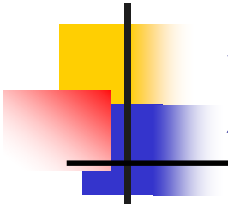
Product SKU	Amount	Total Cost	Total Price
b01	5000	500.00	1250.00
b03	0	0.00	0.00
s05	900	10800.00	18000.00

Cash on hand: -4050.00; total value of inventory: 19250.00



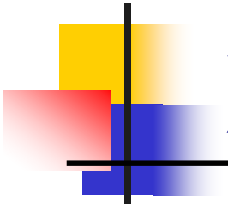
JAXP XPath API

- Designed to allow XPath 1.0 expressions to be evaluated on any data model (that has a defined mapping to XPath 1.0)
- Various other XPath API's are either bound to a particular data model (*e.g.*, DOM level 3) or somewhat tied to an implementation (*e.g.*, Jaxen)
- Factory look-up mechanism therefore works exactly like validation API look-up mechanism
- Base (javax.xml.xpath.XPathFactory) to which the data model's URI is appended



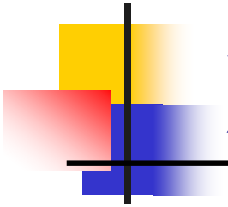
XPathFactory *(Continued)*

- Only data model required to be supported is DOM
- XPathFactory#newInstance(): creates an XPathFactory to be used with DOM
- XPathFactory#newInstance(String): provided so that a URI identifying another data model can be passed in
- Also has an isObjectModelSupported(String) method, for the same purpose as SchemaFactory#isSchemaLanguageSupported (String)



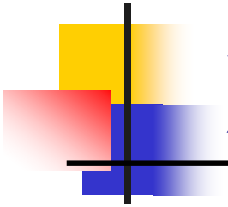
XPathFactory *(Continued)*

- Can get and query features (but not properties)
- XPathFunctionResolver and XPathVariableResolver can be set, and will be used by all manufacutred XPath objects
- The newXPath() method will create new XPath objects appropriate for the data model selected



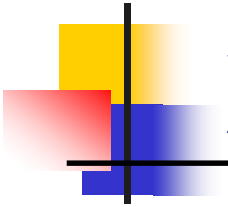
XPathVariableResolver

- Implemented by the application, this is used to provide values for variables referenced in XPath expressions
- Will be called by the XPath processor when it encounters the variable
- Returns an Object; that Object must be appropriate for the data model in use
- *e.g.*, for DOM, would most often be some type of Node



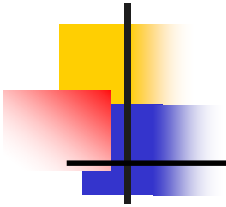
XPathFunctionResolver

- Like XPathVariableResolver, implemented by the application and invoked when an unknown XPath function is encountered
- Takes a QName object — the function's name — and an int corresponding to the number of the function's arguments
- Returns an XPathFunction object
- The processor will call the evaluate method on this object
- evaluate takes a List with the appropriate number of elements; returns an Object which must be appropriate to the data model



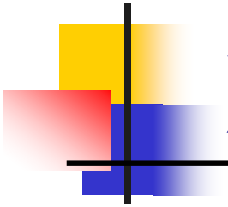
XPath

- XPath objects can be reset; they are neither immutable nor thread-safe
- An XPathVariableResolver and XPathFunctionResolver can be attached/queried
- A NamespaceContext can also be set/queried
- This latter is required when an XPathExpression is being compiled without reference to a context node



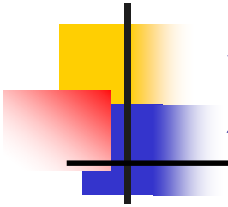
XPath *(Continued)*

- This kind of static compilation is very useful when particular XPath expressions will be used often or against various documents
  - With many processors, the XPath expression will be compiled into Java bytecode
- XPath objects may also be used for evaluation; then a String representing the XPath expression is interpreted directly
- In this case, the namespaces in scope given the context node are used to interpret the XPath expression



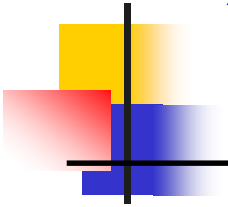
XPath Evaluation

- Both XPath and XPathExpression have 4 evaluate methods
- Only difference is that XPath's take a String for the XPath expression, while XPathExpression embodies the expression
- XPathExpression#evaluate(Object item, QName returnType): Object — item is the context node, will be of a type defined by the underlying data model
- QName defines which of XPath 1.0's data types should be returned (defined in XPathConstants)



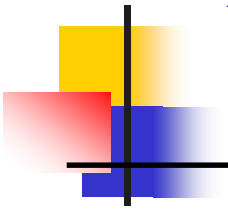
XPath Evaluation *(Continued)*

- XPathExpression#Evaluate(Object item): String — convenience method to be used when the expected return type is a String
- XPathExpression#evaluate(InputSource , QName returnType): Object — will cause the InputSource to be parsed by the implementation
- XPathExpression#evaluate(InputSource): String — convenience method to be used when the expected return type is a String
- The Document node will be used as the context node in evaluating the expression in both the preceding two methods



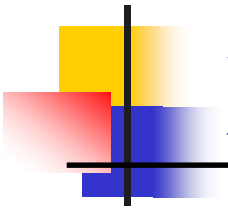
XPath 1.0 Data Model to Java Mapping

- XPath number type: maps to Java Double; called out with the NUMBER field of XPathConstants
- XPath string type: maps to Java String; called out with the STRING field of XPathConstants
- XPath boolean type: maps to Java Boolean; called out with the BOOLEAN field of XPathConstants
- XPath node list type: depends on data model; in DOM, maps to NodeList; called out with the NODESET field of XPathConstants



XPath Example: Instance Document

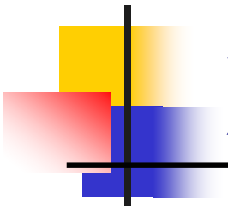
```
<?xml version="1.0" encoding="ASCII"?>
<purchaseOrder xmlns="http://stockings.com">
    <order id="12365">
        <product name="ratty green socks"
            id="rgs09" quantity="22"/>
        <shipTo name="Pete the Pauper"
                address="a warm grate"/>
    </order>
    <billTo name="Pete the Pauper" address="a
          warm grate“ totalPrice="22"/>
</purchaseOrder>
```



XPath Example: Code

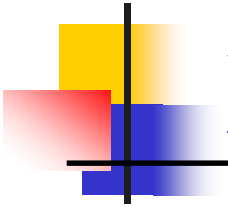
```
private static final String[] BAD_CUSTOMERS =
  {"Shifty Shamis", "Pete the Pauper"};

// in the main method
String xpathFilter =
  "string(s:purchaseOrder/s:billTo/@name)";
String totalPriceExpr =
  "number(s:purchaseOrder/s:billTo/@totalPrice)";
String destinationsExpr =
  "s:purchaseOrder//s:shipTo";
```



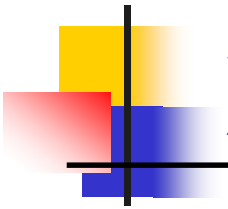
XPath Example: Code *(Continued)*

```
try {
   DocumentBuilderFactory dbf =
      DocumentBuilderFactory.newInstance();
   dbf.setNamespaceAware(true);
   dbf.setValidating(false);
   DocumentBuilder db =
      dbf.newDocumentBuilder();
   /* get an XPathFactory for the DOM */
   XPathFactory xf =
      XPathFactory.newInstance();
   XPath xpath = xf.newXPath();
```



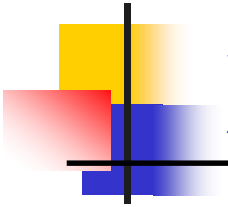
XPath Example: Code *(Continued)*

```
// we have to make a NamespaceContext since
// we're going to precompile our filter:
NamespaceContext nsc = new
   SimpleNamespaceContext();
xpath.setNamespaceContext(nsc);
XPathExpression filterExpression =
   xpath.compile(xpathFilter);
InputSource fileSource = new InputSource();
```



XPath Example: Code *(Continued)*

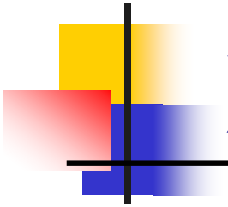
```
for(int i=0; i<args.length; i++) {
   fileSource.setSystemId(args[i]);
   String customerName =
 filterExpression.evaluate(fileSource);
   if(customerName == null ||
         customerName.length() == 0) {
     System.out.println(
         "not processing file " + args[i]
         + "; no customer.");
     continue;
   }
   boolean processCustomer = true;
```



XPath Example: Code *(Continued)*

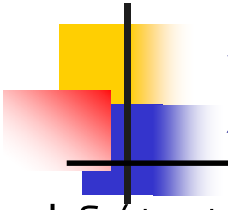
```
for(int j=0; j<BAD_CUSTOMERS.length; j++) {
  if(customerName.equalsIgnoreCase(
       BAD_CUSTOMERS[j])) {
    System.out.println(
      "Considering order from bad
customer...");
    // lost the bet; have to process this

    db.reset();
    Document doc = db.parse(fileSource);
```



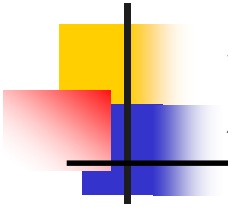
XPath Example: Code *(Continued)*

```
Double totalPrice =
     (Double)xpath.evaluate(
     totalPriceExpr,
     doc, XPathConstants.NUMBER);
NodeList orders =
     (NodeList)xpath.evaluate(
     destinationsExpr,
     doc, XPathConstants.NODESET);
```



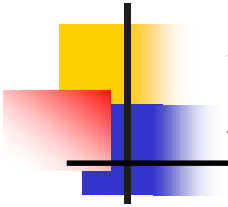
XPath Example: Code *(Continued)*

```
if(totalPrice.doubleValue() > 25.0) {
      processCustomer = false;
      System.out.println(
      "Declining to process order for bad customer”
        + customerName + " because they ordered $"
        + totalPrice + ", which is more than $25");
}
if(orders.getLength() > 1) {
        processCustomer = false;
        System.out.println(
      "Declining to process order for bad customer”
      + customerName
      + " because their order involved "
      + orders.getLength() + " destinations.");
}
```



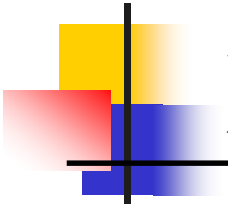
XPath Example: Code *(Continued)*

```
    }
    if(processCustomer) {
        System.out.println("order from customer”
            + customerName
            + " will be processed.");
    }
}
} catch (Exception e) {
  System.out.println(
     "Oh-oh; something went badly wrong...");
}
```



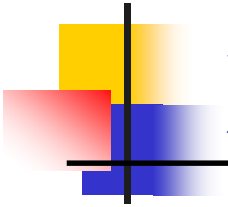
XPath Example: Code *(Continued)*

```
private static class SimpleNamespaceContext
  implements NamespaceContext {
  private HashMap<String, String>
     fPrefix2URIMap;
  private HashMap<String, String>
     fURI2PrefixMap;
  SimpleNamespaceContext () {
     fPrefix2URIMap = new HashMap<String,
          String>();
     fURI2PrefixMap = new HashMap<String,
          String>();
```



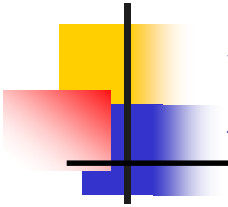
XPath Example: Code *(Continued)*

```
 // set up our namespaces
 fURI2PrefixMap.put("http://stockings.com", "s");
 fPrefix2URIMap.put("s", "http://stockings.com");
}
public String getNamespaceURI(String prefix) {
  String retVal = fPrefix2URIMap.get(prefix);
  return (retVal == null)
     ?XMLConstants.NULL_NS_URI
     :retVal;
}
```



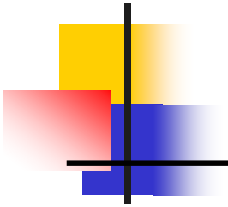
XPath Example: Code *(Continued)*

```
public String getPrefix(String uri) {
   String retVal = fURI2PrefixMap.get(uri);
   return (retVal ==      null)
      ?XMLConstants.NULL_NS_URI:retVal;
}
public Iterator getPrefixes(String uri) {
    String retStr = fPrefix2URIMap.get(uri);
    Vector<String> retVal = new Vector<String>();
    if(retStr != null)
     retVal.addElement(retStr);
    return retVal.iterator();
}
```



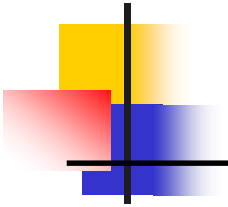
XPath Example: Output

```
Considering order from bad customer...
Declining to process order for bad customer
  Shifty Shamis because they ordered $839.0,
  which is more than $25
Declining to process order for bad customer
  Shifty Shamis because their order involved 2
  destinations.
order from customer Reliable Rick will be
  processed.
Considering order from bad customer...
order from customer Pete the Pauper will be
  processed.
```



References — XML Core

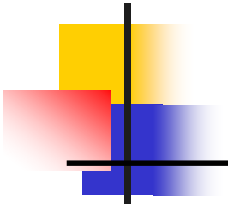
- Extensible Markup Language (XML) 1.1): http://www.w3.org/TR/2004/REC-xml11-20040204/
- Extensible Markup Language (XML) 1.0 (Third Edition): http://www.w3.org/TR/2004/REC-xml-20040204/



References — XML Core

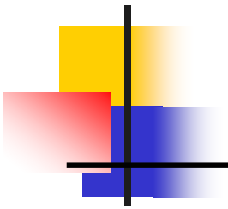
(Continued)

- XML Inclusions (XInclude) Version 1.0, Candidate Recommendation: http://www.w3.org/TR/2004/PR-xinclude-20040930/
- XML Path Language (XPath) Version 1.0: http://www.w3.org/TR/1999/REC-xpath-19991116



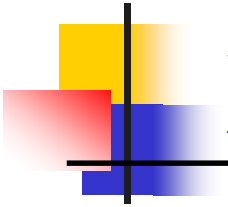
References — XML Schema

- XML Schema Part 0: Primer:
  http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/
- XML Schema Part 1: Structures:
  http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/



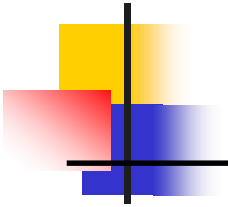
References — XML Schema *(Continued)*

- XML Schema Part 2: Datatypes: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
- XQuery 1.0 and XPath 2.0 Data Model (Working Draft): http://www.w3.org/TR/2004/WD-xpath-datamodel-20040723/



References — XML Schema 2nd Edition

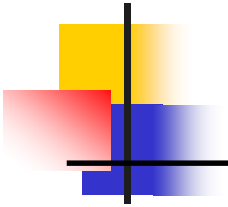
- As of this writing, these are Proposed Edited Recommendations
- XML Schema Part 0: Primer, Second Edition: http://www.w3.org/TR/2004/PER-xmlschema-0-20040318/
- XML Schema Part 1: Structures, Second Edition: http://www.w3.org/TR/2004/PER-xmlschema-1-20040318/
- XML Schema Part 2: Datatypes, Second Edition: http://www.w3.org/TR/2004/PER-xmlschema-2-20040318/



References — APIs

- Document Object Model (DOM) Level 3 Core Specification:
  http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/
- Document Object Model (DOM) Level 3 Load and Save Specification:

  http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407



References — APIs

- SAX home page:
  http://www.saxproject.org/
- JAXP 1.1/1.2 (Final):
  http://www.jcp.org/en/jsr/detail?id=63
- JAXP 1.3 (proposed final draft as of now):
  http://www.jcp.org/en/jsr/detail?id=206