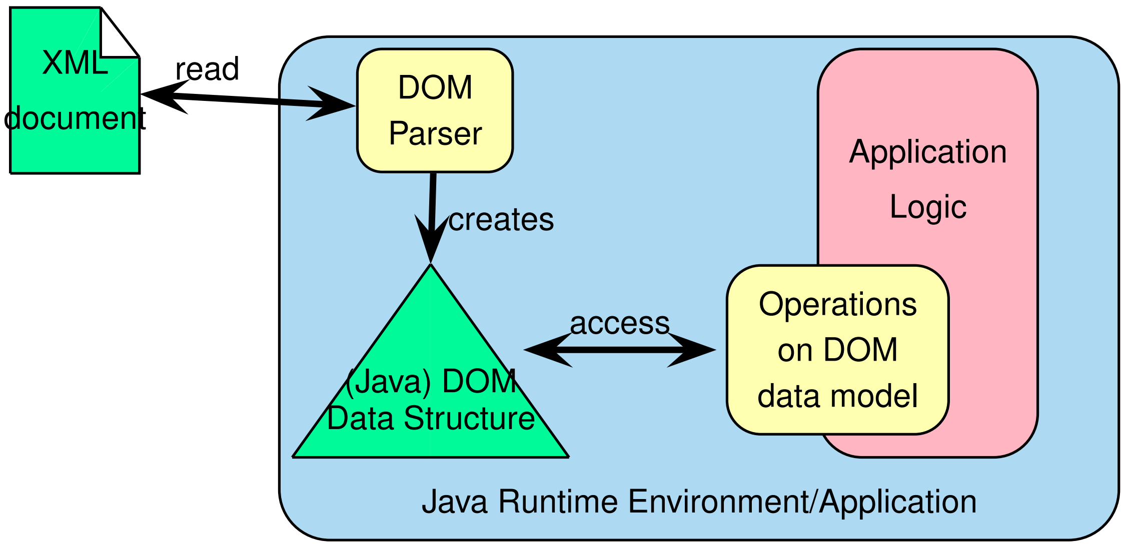
# Chapter 11
# Algorithms and APIs

- XML as a data structure:
  - *abstract datatype* with API: DOM
  - (mainly main-memory) implementations; used e.g. in Java applications
  - low-level API with variable-based access
- Databases?
  - high-level API: XPath, XQuery
  - mapping to relational model (Oracle, IBM DB2) or ObjectTypes (Oracle, DB2)
  - “Native” storage: Software AG-Tamino
  - database functionality: multiuser, transactions, recovery
- Stream Processing:
  - XML data transfer as sequence of events
  - SAX (Simple Application Interface for XML), StAX (Streaming API for XML)



## 11.1 DOM

- DOM (Document Object Model) defines a platform- and language-independent object-oriented *interface* (i.e., an *abstract datatype*) for generating, processing and manipulating XML data.



## DOM

- DOM is a specification of an interface/abstract datatype, *not a* data model and *not a* programming language!
- implementations in Java, C++, etc; usually main-memory-based;
  specialized Java interface definition: JDOM
- language base of the DOM specification: OMG-IDL
- Main-memory-based: only for small application programs
  (most of the "lightweight"-tools used in the course are internally based on DOM)



## DOM: PRINCIPLES

- only one document in a DOM
- step-by-step-access to the data:
  based on variable assignments in the surrounding imperative/object-oriented programming language and on iterators (cf. proceeding in the network data model):
  - document: represents the complete document,
    - Query-Methods, e.g. NodeList getElementsByName(string)
  - class "Node": getNodeType(), getChildren(), getFirstChild(), getNextSibling(), getParentNode(), ...
  - class "Element": getName(), getAttributes(), getContent(), ...
  - class "Attribute": getName(), getValue(), ...
  - corresponding methods for generating and changing nodes.
- additionally, XPath and XSLT can be applied to instances of Document and Element;
- based on DOM, XPath and XQuery can be implemented (cf. Apache Xerces (XML/DOM)/Xalan (XSLT)/Xindice (DB))
- often inefficient (no indexes, query optimization)

## DOM – sample code fragment: Stepwise access

(taken from LanguageElement.java from MARS)

```
// given: Element element;

protected Set<InputVariableDefinition> getInputVariableDefinitions(
          boolean includeJoinVariables)
{ Set<InputVariableDefinition> definitions = new HashSet<InputVariableDefinition>();

  @SuppressWarnings("unchecked")
  List<Element> elements = element.getChildren();
  for (Element e : elements)
  { String elementName = e.getName();
    if (!elementName.equals("Opaque"))
    { String name = e.getAttributeValue("name", "");
      InputVariableDefinition variable = null;
      if (elementName.equals("has-input-variable"))
        variable = new InputVariableDefinition(name, InputVariableDefinition.INPUT);
      else if (elementName.equals("uses-variable") && includeJoinVariables)
        variable = new InputVariableDefinition(name, InputVariableDefinition.USE);
      if (variable != null)
      { String use = e.getAttributeValue("use", "");
        if (use.length() > 0) variable.setUse(use);
        definitions.add(variable);
      }
    }
  }
  return definitions;
}
```



## DOM – sample code fragment: XPath

(taken from ServiceRegistry.java from MARS)

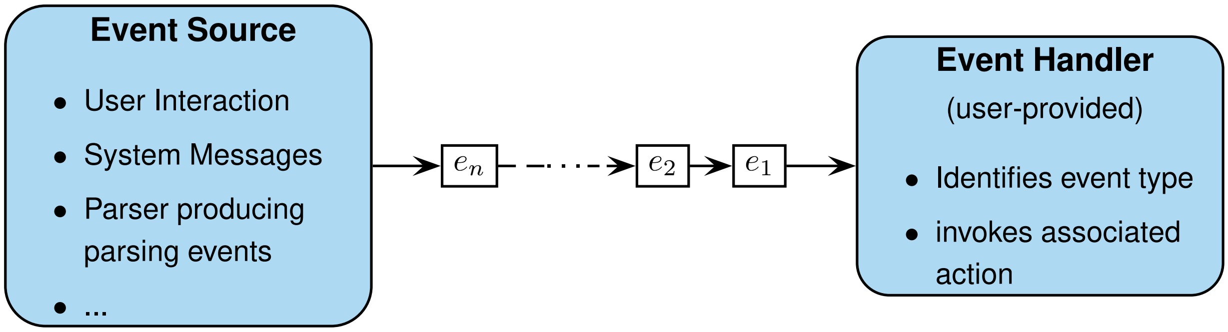
- similar to the JDBC statement concept for SQL in Java:

```
public Element getTaskDescr(Element serviceDescr, String task)
{
  Element taskDescr = null;
  try
  {  XPath xpath = XPath.newInstance(
         "./lsr:has-task-description/lsr:TaskDescription[" +
         "contains(lsr:describes-task/@rdf:resource,$task)]");
     xpath.addNamespace(Namespaces.RDF_NS);
     xpath.addNamespace(Namespaces.MARS_NS);
     xpath.addNamespace(Namespaces.LSR_NS);
     xpath.setVariable("task", task);
     taskDescr = (Element) xpath.selectSingleNode(serviceDescr);
  }
  catch (Exception e) {...}
}
```

# 11.2 SAX: Event-Based XML Processing

- SAX (The Simple API for XML) is an *event-based interface/model*

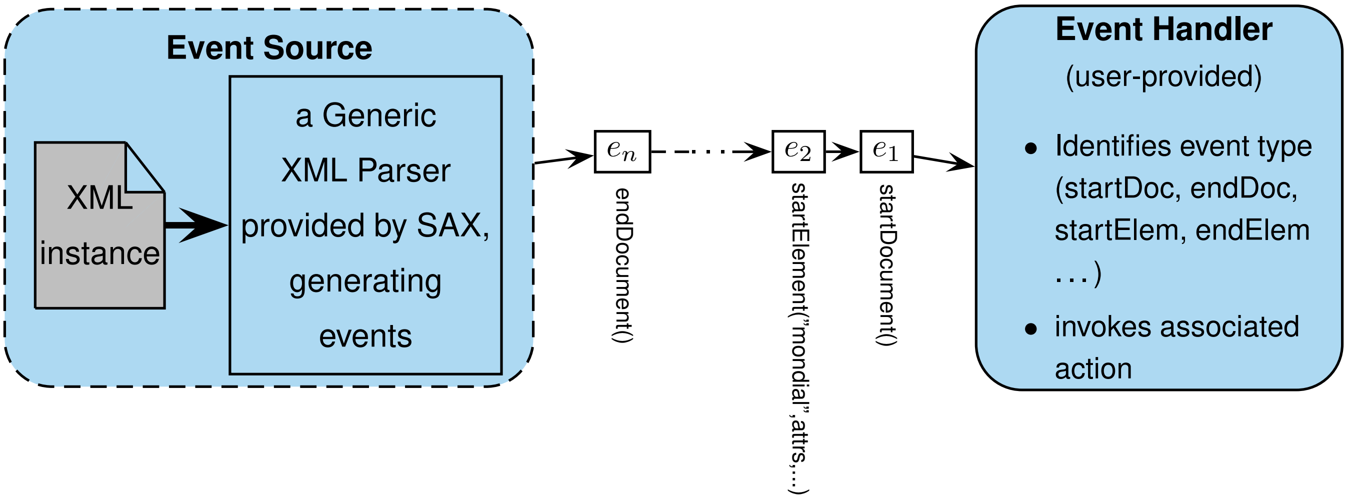
Event-Based Processing as a *general Design Pattern*




- The application programmer provides the Event Handler implementation, containing actions for each type of event;
- kind of *rule-based*;
- user is *not* in charge of the control flow.



# Event-based XML Parsing with SAX




Represents/processes an XML document as a sequence of events (depth-first traversal), e.g.

- startDocument(), endDocument()
- startElement(Name, attributesList) – attributes not split
- endElement(Name)
- characters(string, int)

## XML PARSING WITH SAX

- a generic XML Parser is parameterized with a *Content Handler* (plus *Error Handler*, *DTD Handler*, and *EntityResolver*) implementation.
- The most trivial Content Handler is the *DefaultHandler* that does nothing: the document is parsed, events are detected, but no action is performed.
- Normally, the user-provided Content Handler extends the DefaultHandler, overwriting (some of) its Event Methods.
- even with only the unextended DefaultHandler: DTD / XML Schema validation is still performed (if switched on).
- With the content handler implementation, the user provides "actions" in form of Java code, associated with specific events (and even dependent on context information).
- if during parsing of the XML document, a specific event occurs, the code of the associated action from the content handler is invoked (*"callback"*).



## SAX: APPLICATIONS

Only events are signaled: linear processing based on incoming sequence of events.

- ... among many other things, one can generate a DOM tree structure,
- validation according to a DTD (using the automaton as given on Slide 175) in linear time,
- stream-processing of XML input
  - start processing already when input document is not yet complete
  - filtering for elements that are relevant for a given application
  - linear search for something, e.g., names of countries
    (Exercise: sketch the behavior of the event handler on relevant events)
- if necessary: application needs to maintain context
  - usually: maintain a stack that contains all ancestor elements of the element that is currently processed
    (Exercise: describe the event-handler that maintains the stack)

## SAX Example Code

Consider a very simple application that

- detects all elements with attributes
- for each element, output the element's name
- for each element, output the name-value pairs of its attributes

```
>java PrintAttributes mondial.xml > bla.out
>less bla.out
element: country
 - attribute: 'car_code' value: 'AL' type: 'ID'
 - attribute: 'area' value: '28750' type: 'CDATA'
 - attribute: 'capital' value: 'cty-cid-cia-Albania-Tirane' type: 'IDREF'
 - attribute: 'memberships' value: 'org-BSEC org-CE org-CCC org-ECE org-EBRD org-EU ...' type: 'IDREFS'
element: encompassed
 - attribute: 'continent' value: 'europe' type: 'IDREF'
 - attribute: 'percentage' value: '100' type: 'CDATA'
element: ethnicgroups
 - attribute: 'percentage' value: '3' type: 'CDATA'
element: ethnicgroups
 - attribute: 'percentage' value: '95' type: 'CDATA'
element: religions
 - attribute: 'percentage' value: '70' type: 'CDATA'
...
```



Class "PrintAttributes.java":

```
import java.io.IOException;
import javax.xml.parsers.*;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class PrintAttributes {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("usage: PrintAttributes <url>");
            System.exit(1);
        }
        String url = args[0];    // ... prepare a contentHandler:
        DefaultHandler handler = new ContentHandlerPrintAttributes(
                "printing attributes of document at url '" + url + "'");
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            SAXParser parser = factory.newSAXParser();
            parser.parse(url, handler);  // <<<<<<<<< and now it runs ...
        } catch (IOException e1) {
            e1.printStackTrace();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        }
    }
}
```

[see java/SAX/PrintAttributes.java]

Class “ContentHandlerPrintAttributes.java”:

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ContentHandlerPrintAttributes extends DefaultHandler {
    public ContentHandlerPrintAttributes(String message) {
        System.out.println(message);
    }

    // react on opening elements:
    public void startElement(String url, String localName, String qName,
            Attributes attrs) throws SAXException {
        if (attrs.getLength() > 0) {
            String elementName;
            if(qName == null || qName.equals("")) elementName = localName;
            else elementName = qName;
            System.out.println("element: " + elementName);
            for (int i = 0; i < attrs.getLength(); i++) {
                System.out.println(" - attribute: '" + attrs.getQName(i)
                        + "' value: '" + attrs.getValue(i) + "' type: '"
                        + attrs.getType(i)+"'");
            }
            System.out.println();
        }
    }

    // all other "parsing events" are ignored in this case
}
```

[see java/SAX/ContentHandlerPrintAttributes.java]



## SAX: Applications to XPath Query Answering

### Forward queries

XPath-queries like //country[@car_code='D']/population can be answered very (time- and memory-)efficient,

- use the sequence of events (linear)
- maintain some context (often logspace/additional logtime sufficient)

... works only for queries, that contain only forward steps,

### General queries

which XPath expressions can be *transformed* in equivalent forward-expressions (and with what efforts)?

- “XPath: Looking forward”; F. Bry et al ; 2002; LMU München
- theory: complexity, connections to linear temporal logic
  For every linear temporal logic formula that uses past and future operators, there is an equivalent formula that uses only future operators
  ... but in general of exponential size.

## 11.3 StAX - The Streaming API for XML

Recall *SAX - Event-based parsing of XML documents*:

- Pro's: performant and simple.
- Con's:
  - not very flexible, serial access only
  - control flow is passed to the parser (a so-called *push*-API)

### Processing of Streamed XML Data with *StAX*

SAX appears somewhat sturdy to many software developers, since passing the control flow to a subordinate parser is not a common design pattern.

- Idea: on-the fly processing, no in-memory representation for good performance, and thus enabling for processing of XML streams (also *SAX*-like), *BUT:*
- control flow is not passed to the parser (**unlike** *SAX*), but relevant XML items are accessed using a *cursor/iterator*, controlled by the Java program using the StAX API (*Pull-API*)



## *StAX* EXAMPLE: EXAM REGISTRATION

Assume the administration of exams in a student's office ("Prüfungsamt"):

- The *subject* (e.g., "Semi-structured Data and XML") and ID of lectures/exams,
- whether the exam is *written* or *oral*,
- for written exams, the date of the exam,
- for oral exams, a number of dates is given when the single exams are held.
- the registration period *starts* when receiving an incoming XML message `start-registration`
- the registration period *ends* when receiving an incoming XML message `end-registration`
- for all students that did (`register`) correctly, the student's relevant details are extracted and written to some output stream (`valid-register`; in the example, we use `STDOUT`.)
- students that register before beginning or after the end of registration, are not accounted for the exam; an XML message `invalid-register` goes to `STDOUT`,

## *StAX* Example: Exam Registration (Cont'd)

- the registration data of the students comes in via a continuous input stream;
- the program should allow the management of registrations for multiple exams at one time (all incoming over the same input stream).

Example stream:

… → start-registration(lecture="XML", id="08154711", mode="written", date="7/11/2008") →
register(student="0001", exam="08154711") → register(student="0009", exam="08154711") →
register(student="0004", exam="08154711") → end-registration(id="08154711") →
register(student="0007", exam="08154711") → …

⇒ XML Processor ⇒

… → registered-exam(lecture="XML", id="08154711", mode="written", date="7/11/2008") →
valid-register(student="0001", exam="08154711") → valid-register(student="0009", exam="08154711") →
valid-register(student="0004", exam="08154711") → invalid-register(student="0007", exam="08154711") → …



## *StAX* Example Cont'd:

Consider the following XML sequence as input stream:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<stream>
<register student="0007" exam="08154711"/>
<start-registration id="08154711" mode="written">
  <subject>Semistructured Data and XML</subject>
  <date>07/11/2008</date>
</start-registration>
<register student="0001" exam="08154711"/>
<register student="0009" exam="08154711"/>
<start-registration id="12345678" mode="oral">
  <subject>Dental Hygiene</subject>
  <dates>
    <date>17/9/2008</date>
    <date>18/9/2008</date>
  </dates>
</start-registration>
<end-registration id="12345678"/>
<register student="0004" exam="08154711"/>
<register student="0004" exam="12345678"/>
<register student="0007" exam="12345678"/>
<end-registration id="08154711"/>
<register student="0007" exam="08154711"/>
</stream>
```

[Filename: StAX/exam.xml]

## StAX Example Cont'd (2):

Here is the code for the `Exam` bean, containing the exam's properties and some constants):

```
import java.util.ArrayList;
import java.util.List;

public class Exam {
  public static final String DATE = "date";
  public static final String SUBJECT = "subject";
  public static final String ID = "id";
  public static final String MODE = "mode";
  public static final String DATES = "dates";
  public static final String STARTOFREG = "start-of-registering";
  public static final String ENDOFREG = "end-of-registering";

  private String id;
  private boolean oral;
  private String subject;
  private String date;
  private List<String> dates;
  private boolean registeringClosed = false;
  private String startOfReg;
  private String endOfReg;

  public Exam(String id, String mode) {
    this.id = id;
    this.oral = "oral".equals(mode);
    this.dates = new ArrayList();
  }
```



```
  public String getId() { return id; }
  public void setDate(String date) { this.date = date; }
  public String getDate() { return date; }
  public void setDates(List<String> dates) {this.dates = dates; }
  public List<String> getDates() { return dates; }
  public void setSubject(String subject) { this.subject = subject; }
  public String getSubject() { return subject; }
  public boolean isOral() { return oral; }
  public boolean isWritten() { return (!oral); }

  public String getMode() {
    if (oral) return "oral";
    return "written";
  }
  public boolean isRegisteringClosed() {
    return registeringClosed;
  }
  public void setRegisteringClosed(boolean registeringClosed) {
    this.registeringClosed = registeringClosed;
  }
  public String getEndOfReg() {
    return endOfReg;
  }
  public String getStartOfReg() {
    return startOfReg;
  }
  public void setStartOfReg(String startOfReg) {
    this.startOfReg = startOfReg;
  }
  public void setEndOfReg(String endOfReg) {
    this.endOfReg = endOfReg;
  }
}
```

[Filename: StAX/Exam.java]

# *StAX* Example Cont'd (3):

Here the code for the main parser class, containing the `main` method:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.OutputStream;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamWriter;


public class ExamStreamParser {

  FileInputStream inputStream;
  OutputStream outputStream;

  public ExamStreamParser(FileInputStream in, OutputStream out) {
    this.inputStream = in;
    this.outputStream = out;
  }


  public void startParsing() {
    try {
      XMLInputFactory inputFactory = XMLInputFactory.newInstance();
      XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
      XMLStreamReader parser = inputFactory.createXMLStreamReader(inputStream);
      XMLStreamWriter writer = outputFactory.createXMLStreamWriter(outputStream);
      Exam currentExam = null;
      Map<String,Exam> exams = new HashMap<String,Exam>();
          boolean goOn = true;
```



```
while (goOn) {
int event = parser.next();
switch(event) {
case XMLStreamConstants.END_DOCUMENT:
   parser.close();
   writer.flush();
   writer.close();
   goOn = false;
   break;
case XMLStreamConstants.START_ELEMENT:
  // start-registration and its subelements
  if("start-registration".equals(parser.getLocalName())) {
      currentExam = new Exam(parser.getAttributeValue(null, Exam.ID),parser.getAttributeValue(null, Exam.MOD
      currentExam.setStartOfReg(getDate());
      break;
    }
  if(Exam.SUBJECT.equals(parser.getLocalName())) {
      currentExam.setSubject(parser.getElementText()); break;
    }
  if(Exam.DATE.equals(parser.getLocalName())) {
      if(currentExam.isWritten()) currentExam.setDate(parser.getElementText());
      else currentExam.getDates().add(parser.getElementText());
      break;
    }
  if("end-registration".equals(parser.getLocalName())) {
      String examId = parser.getAttributeValue(null,Exam.ID);
    Exam exam = exams.get(examId);
    if(exam == null) {
      System.err.println("no such exam with id '"+examId+"' open for registration!");
      break;
    }
      exam.setEndOfReg(getDate());
      exam.setRegisteringClosed(true);
      break;   // no output is generated.
    }
  // register and its subelements
  if("register".equals(parser.getLocalName())) {
      String studentId = parser.getAttributeValue(null, "student");
      String examId = parser.getAttributeValue(null, "exam");
      if(exams.containsKey(examId)) {
        Exam exam = exams.get(examId);
        if(! exam.isRegisteringClosed()) {
          writer.writeStartElement("valid-register");
        writer.writeAttribute("student", studentId);
        writer.writeAttribute("exam", examId);
        writer.writeEndElement();
      } else {
          writer.writeStartElement("invalid-register");
          writer.writeAttribute("student", studentId);
        writer.writeAttribute("exam", examId);
        writer.writeStartElement("message");
        writer.writeCharacters("invalid registration! registration for exam '" + exam.getId()
          + "' (" + exam.getSubject() + ") has ended on " + exam.getEndOfReg());
        writer.writeEndElement();
      }
      } else {
        writer.writeStartElement("invalid-register");
        writer.writeAttribute("student", studentId);
      writer.writeAttribute("exam", examId);
      writer.writeStartElement("message");
      writer.writeCharacters("invalid registration! exam '"+examId+"' is not (yet?) open for registration.")
      writer.writeEndElement();writer.writeEndElement();
      }
      writer.writeCharacters("\n");
      break;
    }
    break;
```

```
        case XMLStreamConstants.END_ELEMENT:
          if("start-registration".equals(parser.getLocalName())) {
            exams.put(currentExam.getId(),currentExam);
            writer.writeStartElement("registered-exam");
            writer.writeAttribute(Exam.ID, currentExam.getId());
            writer.writeAttribute(Exam.MODE, currentExam.getMode());
            writer.writeCharacters("\n  ");
              writer.writeStartElement(Exam.SUBJECT);
            writer.writeCharacters(currentExam.getSubject());
            writer.writeEndElement();
            writer.writeCharacters("\n  ");
              writer.writeStartElement(Exam.STARTOFREG);
            writer.writeCharacters(currentExam.getStartOfReg());
            writer.writeEndElement();
            writer.writeCharacters("\n  ");
              if(currentExam.isWritten()) {
              writer.writeStartElement(Exam.DATE);
              writer.writeCharacters(currentExam.getDate());
              writer.writeEndElement();
            } else {
              writer.writeStartElement(Exam.DATES);
              for(Iterator<String> i=currentExam.getDates().iterator();i.hasNext();) {
                writer.writeStartElement(Exam.DATE);
                writer.writeCharacters(i.next());
                writer.writeEndElement();
              }
              writer.writeEndElement();
            }
              writer.writeCharacters("\n");
              writer.writeEndElement();writer.writeCharacters("\n");
            currentExam = null; // it's better to provoke
            // a nullpointer exception than to edit the wrong exam object
            break;
          }
        }
      }
    } catch (XMLStreamException e) {
      e.printStackTrace();
    }
  }

  private String getDate() {
    DateFormat format = new SimpleDateFormat();
    Date date = new Date();
    return format.format(date);
  }

  public static void main(String[] args) {
    try {
      ExamStreamParser examStreamParser = new ExamStreamParser(new FileInputStream(args[0]), System.out);
      examStreamParser.startParsing();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```



## StAX Pro's and Con's

- Pro's:
  - Quick and performant like SAX
  - can handle XML Streams
  - can produce XML Output in a simple way without fiddling around with Strings, StringBuffers and angle brackets
  - is a "Pull API".
- Con's:
  - Common pattern: huge `switch` command performing a case differentiation over all relevant events
    
    $\Rightarrow$ look and feel is not much different from writing a SAX event handler
    
    $\Rightarrow$ Push vs Pull: *so what?*
  - can easily produce XML output via `XMLStreamWriter` $\Rightarrow$ same feature can be provided by a SAX eventhandler as well $\Rightarrow$ *so what?*

## *StAX* SUMMARY

- seems to connect the flexibility of SAX with the programmer friendliness of SAX
- The along-coming XMLStreamWriter class is nice, but neither ground-breaking nor necessarily inherent to the StAX approach
- the "delta" to plain SAX seems relatively narrow



## SAX AND STAX: APPLICATIONS

Stream-based processing can be applied to XML data on multiple levels:

- low-level applications:
  SAX is often used for building a DOM from ASCII XML input: "opening tag with attributes", "text", "closing tag" can immediately be translated into the DOM constructors.
- low-level streaming of an XML instance:
  answering XPath (forward-axes only) queries; optionally maintaining some context (e.g., stack).
- higher level "application-level events":
  the XML stream is not seen as the traversal of a large instance, but as a sequence of (independent) events (cf. the StAX example)
  (batch processing of a script-like sequence).
  relax [RFID applications, time series of stock quotes, RSS feeds]

## 11.4 JAXB - The Java API for XML Binding

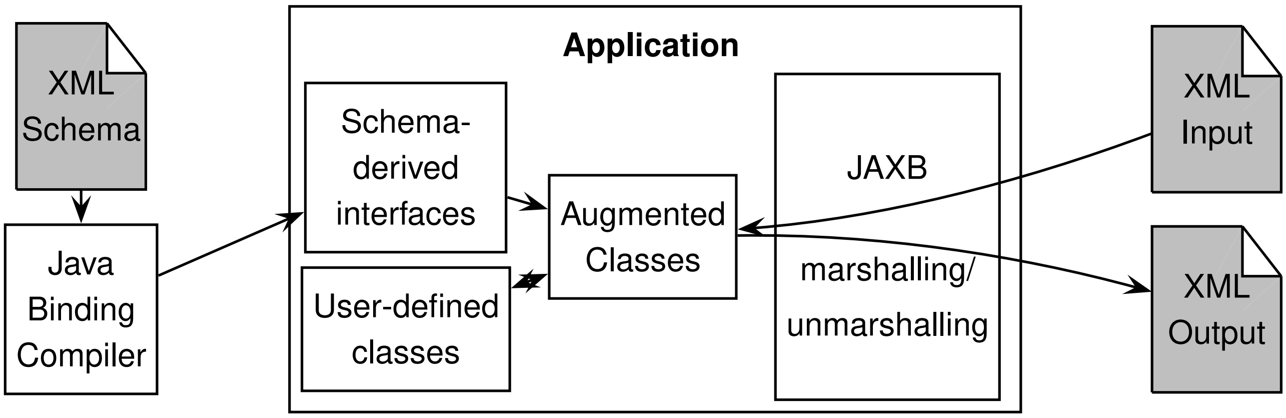
- XML elements describe objects with properties,
- correspond to classes of an application,
- derive interface with setX/getX methods (= Java Beans) as skeletons for these classes (automatically generated from XSD),
- user derives classes from these interfaces by adding behavior,
- application logics implemened by using these classes,
- import/export of XML instances of these classes via generic mappings (derived from the XSD).



## JAXB Architecture

- map XML Schemas to Java interfaces (get/set methods),
- methods for *unmarshalling* XML instance documents into Java objects.
- methods for *marshalling* Java objects back into XML instance documents.

### Architecture



# JAXB - Use Case

Software configuration management via XML files

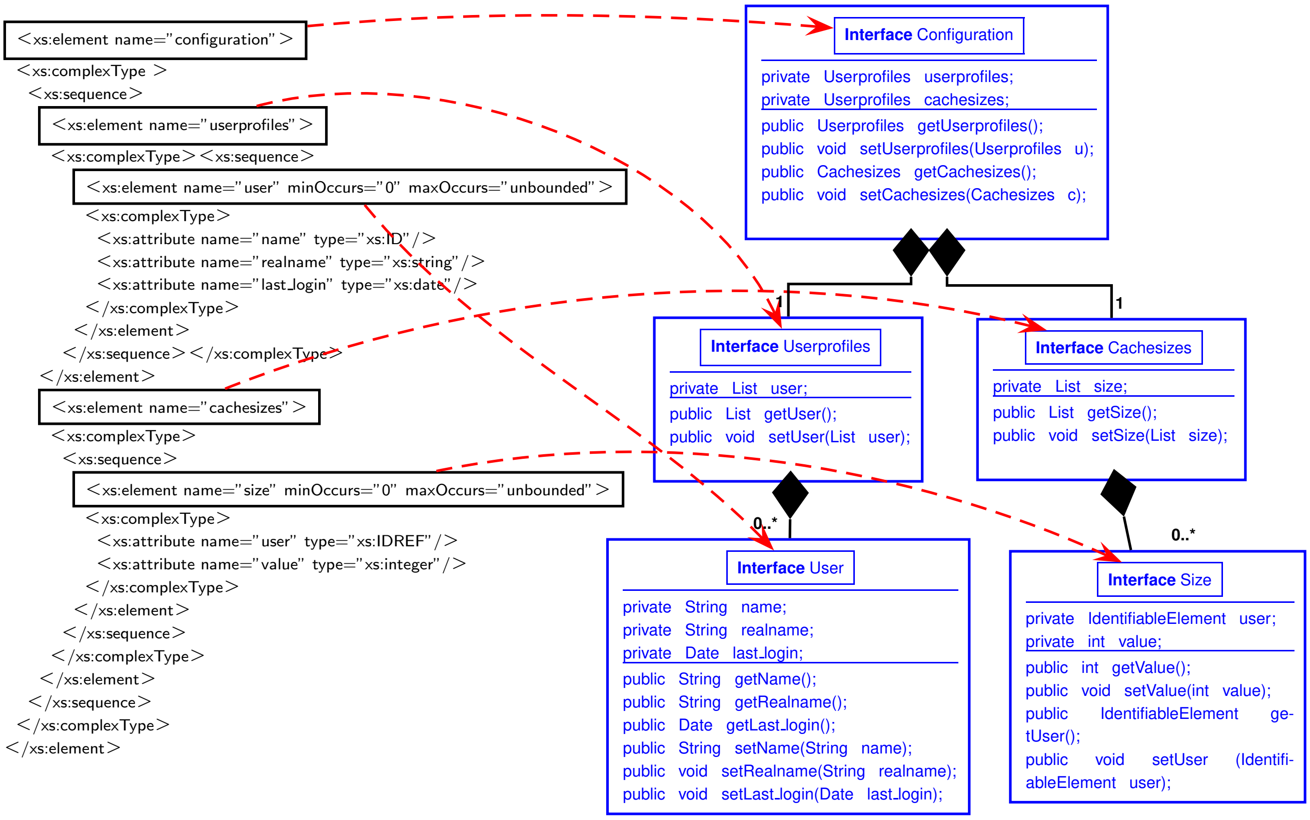
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="config.xsd">
  <userprofiles>
    <user name="wmay" realname="Wolfgang May" last_login="19-09-2005T17:41.02"/>
    <user name="ofritzen" realname="Oliver Fritzen" last_login="21-09-2005T13:04:27"/>
      ...
  </userprofiles>
  <cachesizes>
    <size user="wmay" value="0"/>
    <size user="ofritzen" value="65535"/>
      ...
  </cachesizes>
</configuration>
```

- software with different users
- each user has properties (name, last login etc)
- each user has cache size setting

Imagine the case you want to update wmay's settings for last login:



# JAXB: Binding XML Schema to Java Classes / Interfaces



# JAXB: Example Code Piece

```
:
  try {
      JAXBContext jc = JAXBContext.newInstance("config.binding");
      ObjectFactory objFactory = new ObjectFactory();
      Unmarshaller u = jc.createUnmarshaller();
      Configuration conf = (Configuration) u.unmarshal(new FileInputStream(infilename));

      List users = conf.getUserProfiles().getUsers();
      User user = null;
      for(int i=0; i<users.getLength(); i++) {
          user = (User) users.item(i);
          if(user.getName().equals("wmay")) break;
      }
      if(user != null) user.lastLogin(timer.getDateTime());

      Marshaller m = jc.createMarshaller();
      // put result into DOM tree
      DOMResult domResult = new DOMResult();
      m.marshal(s, domResult);
      Document doc = (Document) domResult.getNode();
      // transformer stuff is only for writing DOM tree to file/stdout
      TransformerFactory factory = TransformerFactory.newInstance();
      Source docSource = new DOMSource(doc);
      StreamResult result = new StreamResult(outfilename);
      Transformer transformer = factory.newTransformer();
      transformer.transform(docSource, result);
  } catch (TransformerConfigurationException e) { e.printStackTrace();
  } catch (TransformerException e) { e.printStackTrace();
  } catch (FileNotFoundException e) { e.printStackTrace();
  } catch (JAXBException e) { e.printStackTrace();
  }
:
```



# JAXB: pro's and con's

Plus:

- allows for easy and lightweighted unmarshaling, bean-based manipulation and marshalling of XML data,
- integrates well with XML files and with DOM trees,
- higher level of abstraction from XML representation, compared with DOM and SAX.

Minus:

- still wasting much time fiddling around with lists of lists of lists ... of items
  (But: seems more like a problem inherent to Java than to JAXB technology)

## JAXB FACTS

- Part of the Java Web Services Developer Pack
- Strongly recommended for studying: SUN's Java WebService Tutorial
  http://java.sun.com/webservices/docs/1.5/tutorial/doc/index.html
- at /afs/informatik.uni-goettingen.de/course/xml-prakt/xml/JAXB-README
  you find an explanation on how to install and use JAXB in the CIP Pool.