Net-Based Applications

Chapter 10: XML Programming

Holger Schwarz
Universität Stuttgart

Winter Term 2016/2017

Overview

- Overview
- DOM
- JAXB
- SAX
- StAX

Overview XML programming

- XML is an application independent format
- DTD/XML Schema: application independent definition of application specific constraints
- application independent XML parsers are possible
- Standardized parser interfaces ensure complete decoupling
  - Available for Java
  - and also for C, C++, Python, Ada, ...
- Four different processing models
  - DOM: application gets parse tree
  - JAXB: parser generates schema-specific object structure
  - SAX: event-based, parser calls call-back methods
  - StAX: event-based, application pulls events from parser

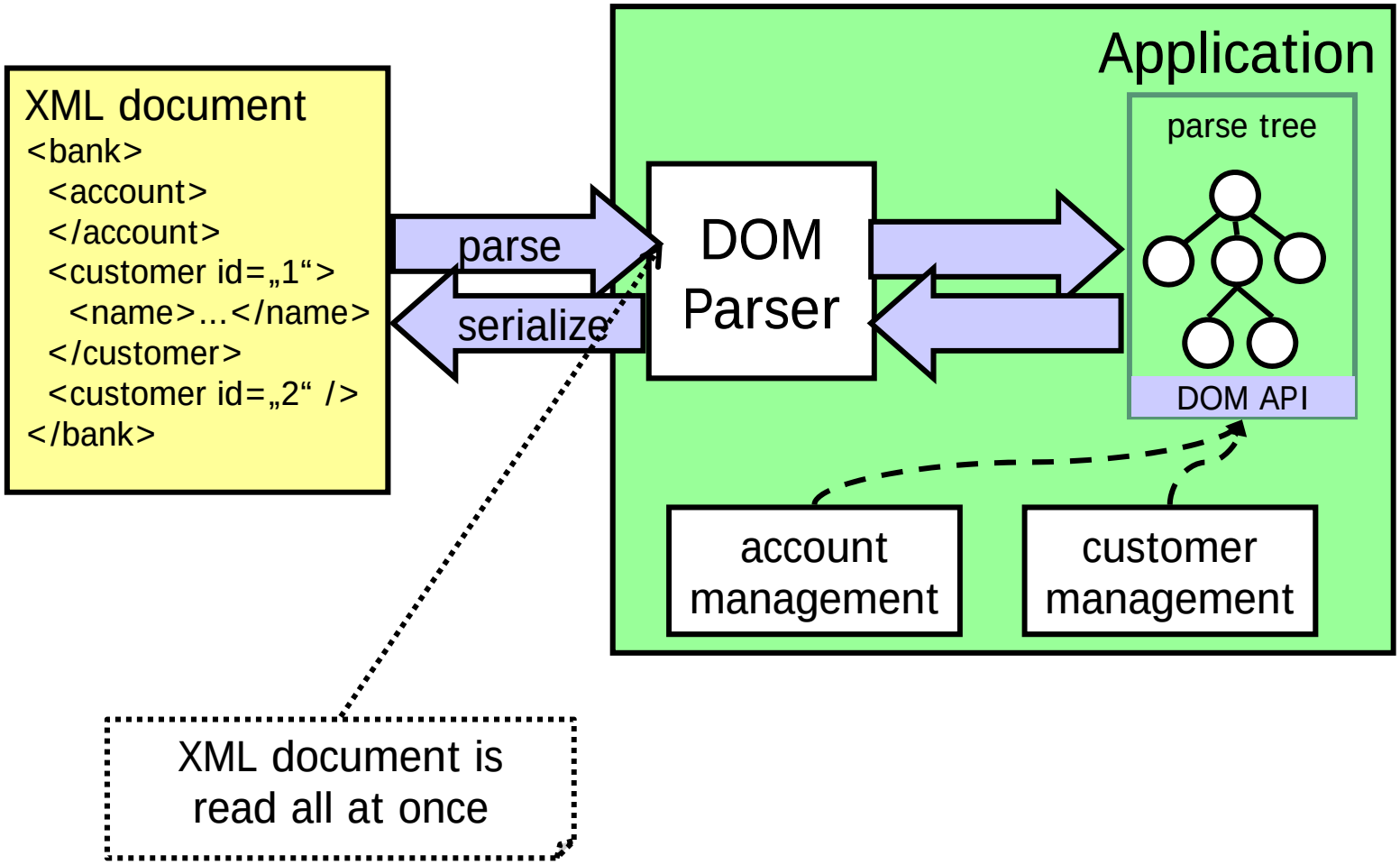
Properties

- API type
  - Parse document in some memory structure
  - Stream of events
- Arbitrary access or sequential read only?
- Memory efficiency: Is the entire document read at once?
- Writing XML documents possible?
- Validation of XML documents supported?
- State management

Overview

- Overview
- DOM - Document Object Model
- JAXB
- SAX
- StAX

Document Object Model (Java DOM)



DOM Overview

- W3C standard
  - Platform- and language-neutral
  - http://www.w3.org/DOM
- Language bindings
  - For Java and JavaScript defined by the W3C
  - Bindings for C++, Python, PHP, ... available
  - Java binding does not use classes from java.util package
- Java specific variant: JDOM
  - Not part of Java or W3C standards
  - http://www.jdom.org

DOM specification levels

- Level 0:
  - all the vendor-specific DOMs that existed before creation of DOM Level 1, e.g. document.images, document.forms, document.layers, and document.all.
  - not a formal specification published by the W3C but rather a reference to what existed before the standardisation process.
- Level 1:
  - Navigation of DOM (HTML and XML) document (tree structure) and manipulation of content (includes adding elements)
  - also: HTML-specific elements
- Level 2:
  - XML namespace support, filtered views and events.
- Level 3:
  - Load and save, Validation, XPath, Views and Formatting

Common Support for XML Parsers

- Java API for XML Parsing (JAXP)
  - Creating parsers
  - Configuring generic features of parsers: validation, namespaces, whitespace processing, XInclude, ...
  - Java standard (not part of the DOM specification)
    - Package javax.xml.parsers
    - Uses org.xml.sax.SAXException
- Validation (XML Schema, RELAX NG, ...)
  - Not relevant for DTD
  - Package javax.xml.validation
- Ignorable whitespace / element content whitespace
  - Parsers can remove whitespace used for formatting XML documents
  - Schema or DTD required

Parsing Documents with a DTD

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

DocumentBuilderFactory parserFactory =
   DocumentBuilderFactory.newInstance();
parserFactory.setNamespaceAware(false);
parserFactory.setValidating(true);
parserFactory.setIgnoringElementContentWhitespace(true);

DocumentBuilder parser = parserFactory.newDocumentBuilder();

Document doc = parser.parse(new File("bank_dtd.xml"));
```

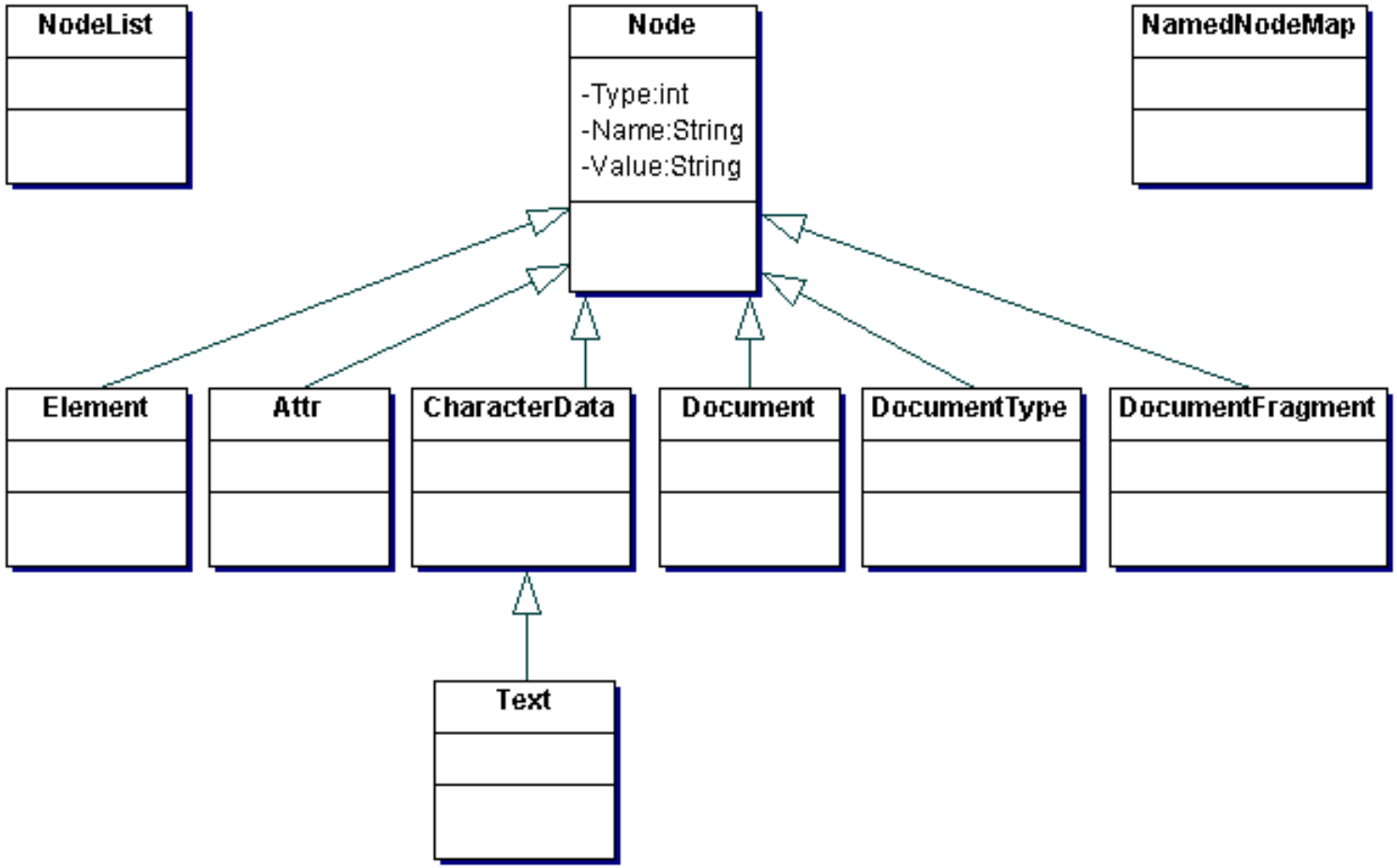
Parsing Documents with an XML Schema

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;

import org.w3c.dom.Document;

SchemaFactory schemaFactory =
   SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
Schema schema = schemaFactory.newSchema(      );

DocumentBuilderFactory parserFactory =
   DocumentBuilderFactory.newInstance();
parserFactory.setNamespaceAware(true);
parserFactory.setValidating(false);
parserFactory.setSchema(schema);
parserFactory.setIgnoringElementContentWhitespace(true);

DocumentBuilder parser = parserFactory.newDocumentBuilder();

Document doc = parser.parse(new File("bank.xml"));
```

schema must be referenced in the document OR provide schema file or URL here

UML model of most important DOM classes

- Node interface represents a single node in the document tree



Reading Node Content

- Type
  - getNodeType(): short
  - get the type of a node as a constant, e.g. ATTRIBUTE_NODE, COMMENT_NODE, DOCUMENT_NODE, ELEMENT_NODE, …
- Name
  - getNodeName(): String
  - name of the Node (if applicable – see Node type interfaces table)
- Values
  - getNodeValue(): String
  - setNodeValue(String)
- Attribute
  - getAttributes(): NamedNodeMap

NamedNodeMap

- Methods for access:
  - getLength(): int
  - item(int): Node
  - getNamedItem(String): Node
- Methods to modify:
  - setNamedItem(Node)
    - If it exists, node will be replaced
  - removeNamedItem(String)
- Namespace-aware variants of these methods with “NS” suffix
- NamedNodeMaps are “live”

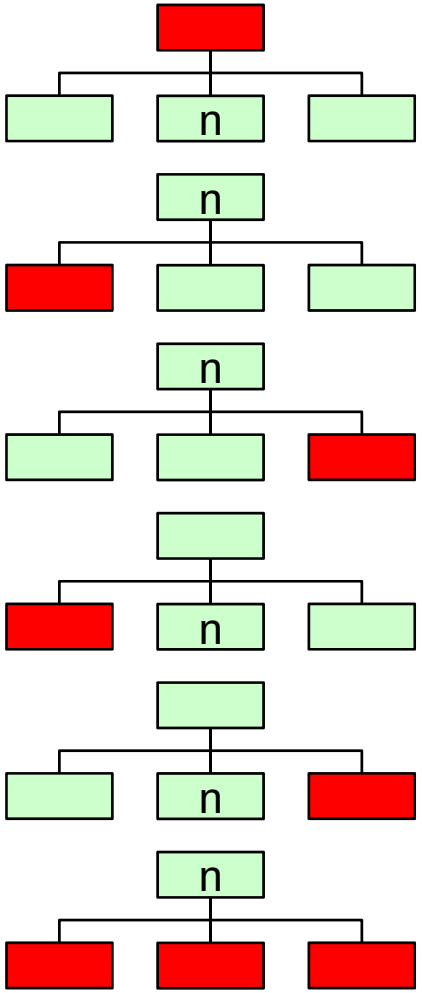
Node type interfaces

Interface	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	"#cdata-section"	content of the CDATA Section	null
Comment	"#comment"	content of the comment	null
Document	"#document"	null	null
DocumentFragment	"#document-fragment"	null	null
DocumentType	document type name	null	null
Element	tag name	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	"#text"	content of the text node	null

Node Interface (Structure)

Given a node n:

- n.getParentNode(): Node
  - Returns the parent node
- n.getFirstChild(): Node
  - Returns the first child node
- n.getLastChild(): Node
  - Returns the last child node
- n.getPreviousSibling(): Node
  - Returns the previous sibling node
- n.getNextSibling(): Node
  - Returns the next sibling node
- n.getChildNodes(): NodeList
  - Returns a node list with all children

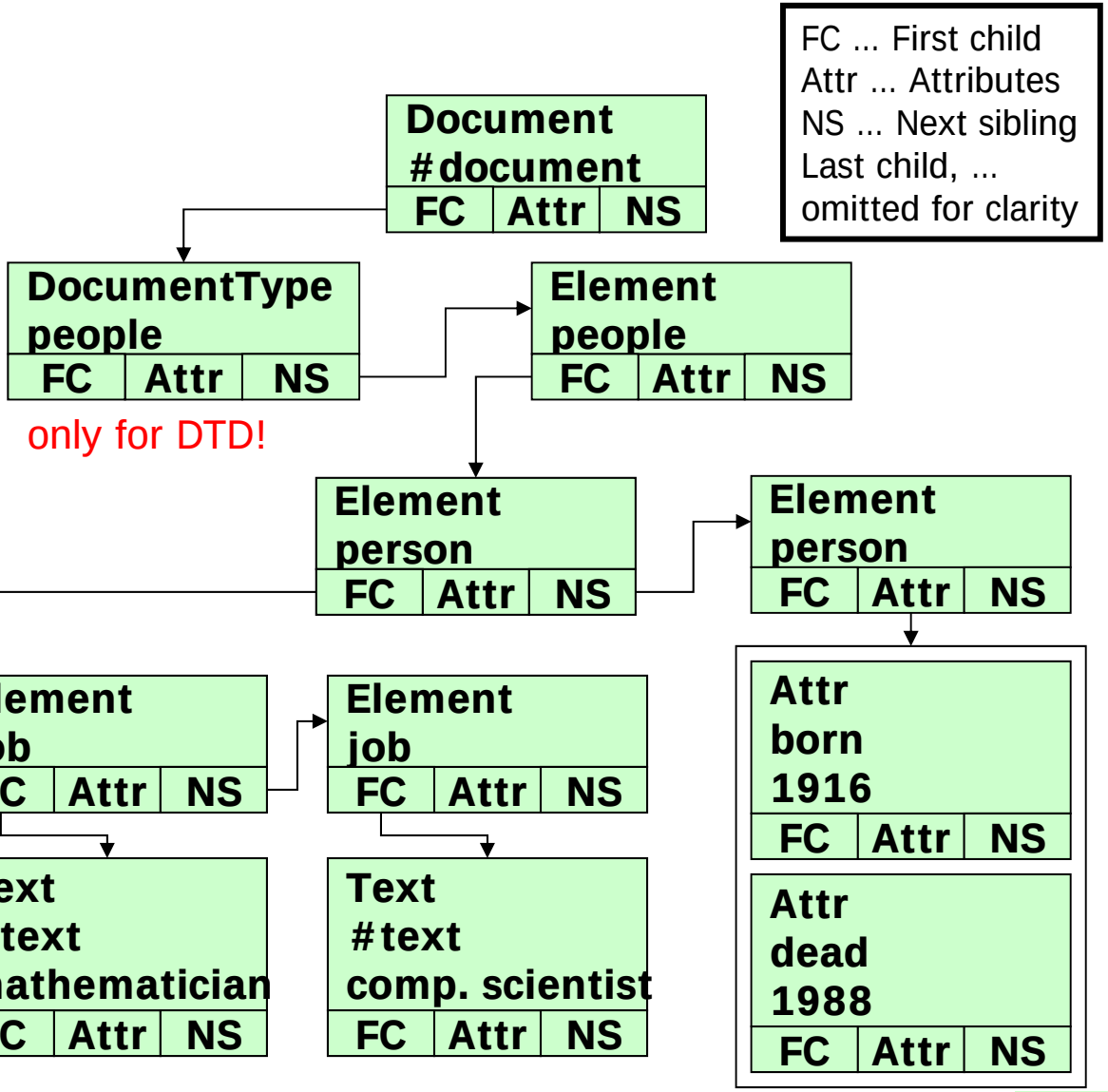


NodeList

- Methods:
  - getLength(): int
  - item(int): Node
- Not modifiable via NodeList interface
- NodeLists are “live”
  - Changes of the DOM are automatically reflected by the list.

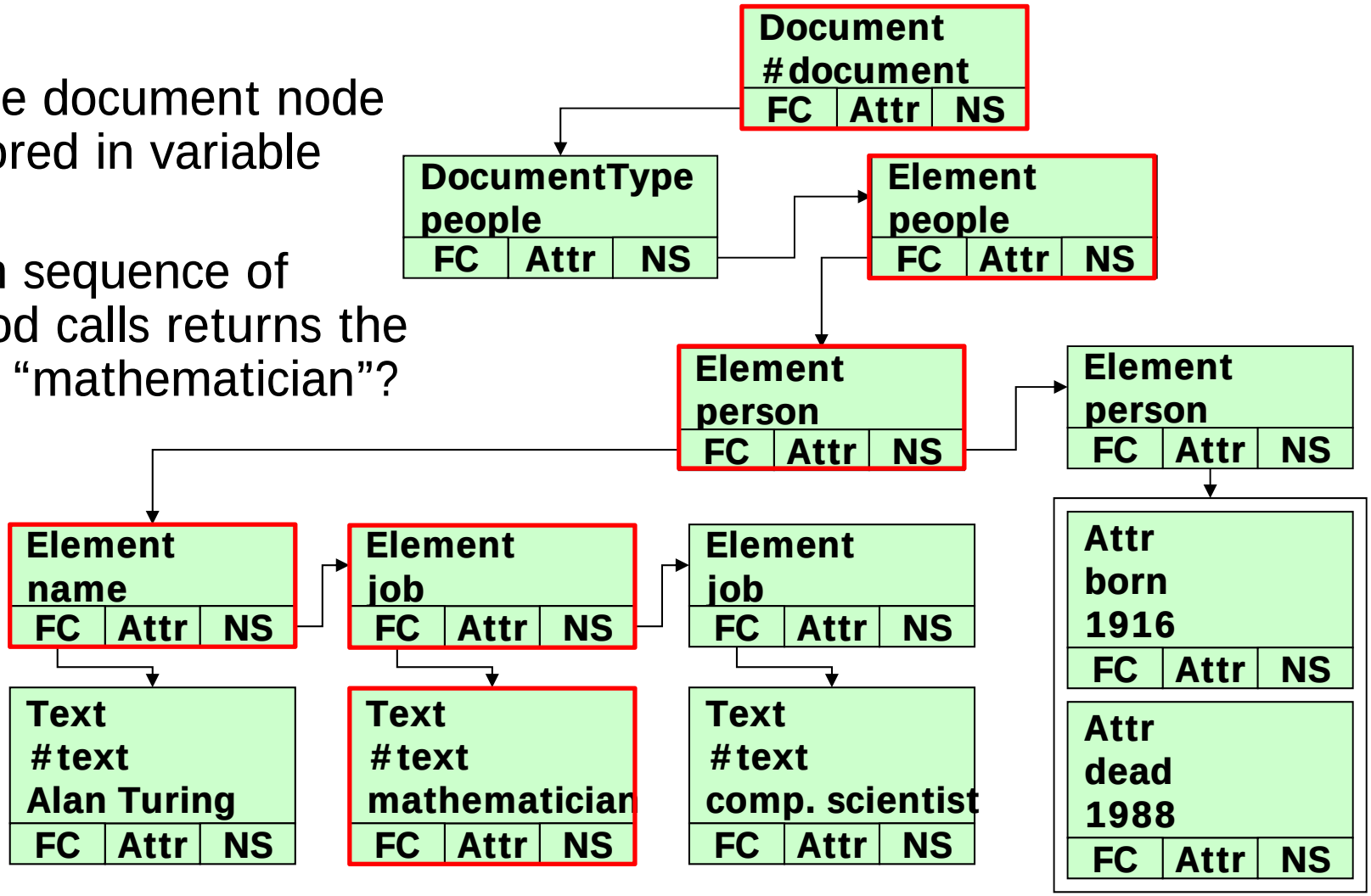
Example of a DOM Tree (1)

```
<?xml version="1.0"?>
<!DOCTYPE people
 SYSTEM "people.dtd">
<people>
 <person>
  <name>Alan Turing</name>
  <job>mathematician</job>
  <job>comp. scientist</job>
 </person>
 <person born="1916"
         dead="1988"/>
</people>
```



Example of a DOM Tree (2)

- Let the document node be stored in variable “doc”.
- Which sequence of method calls returns the string “mathematician”?




```
doc.getLastChild().getFirstChild().getFirstChild().getNextSibling().getFirstChild().getNodeValue()
```

Modifying the node content

- Value
  - setNodeValue(String)
    - only for nodes with reasonable value
- Name
  - Name of a node cannot be changed

Modifying the Document Structure (1)

- Methods of Node interface
  - New child nodes:
    - appendChild(Node): Node
    - insertBefore(Node newChild, Node refChild): Node
  - Delete child nodes:
    - removeChild(Node): Node
  - Replace child nodes:
    - replaceChild(Node newChild, Node oldChild): Node
- Nodes are removed from the tree before being inserted

Modifying the Document Structure (2)

- Nodes cannot exist without a document
  - getOwnerDocument(): Document
- Methods of Document interface
  - Creating new nodes:
    - createElement(String name): Element
    - createAttribute(String name): Attribute
    - createTextNode(String content): Text
    - Namespace-aware methods: createElementNS, createAttributeNS: (String namespaceURI, String prefixedName)
  - Importing nodes from other documents:
    - importNode(Node node, boolean deep): Node
    - Creates a copy of the node
    - Optional: deep copy
  - After copying, nodes can be inserted into the target document (using one of the node methods).

Example for Adding Nodes

- Adding Alan Turing's new job "cryptographer" to the DOM

```
/* parse document into doc */

Text newJobText = doc.createTextNode("cryptographer");
Element newJobElement = doc.createElement("job");
newJobElement.appendChild(newJobText);
Node turingNode = doc.getLastChild().getFirstChild();
turingNode.appendChild(newJobElement);
```

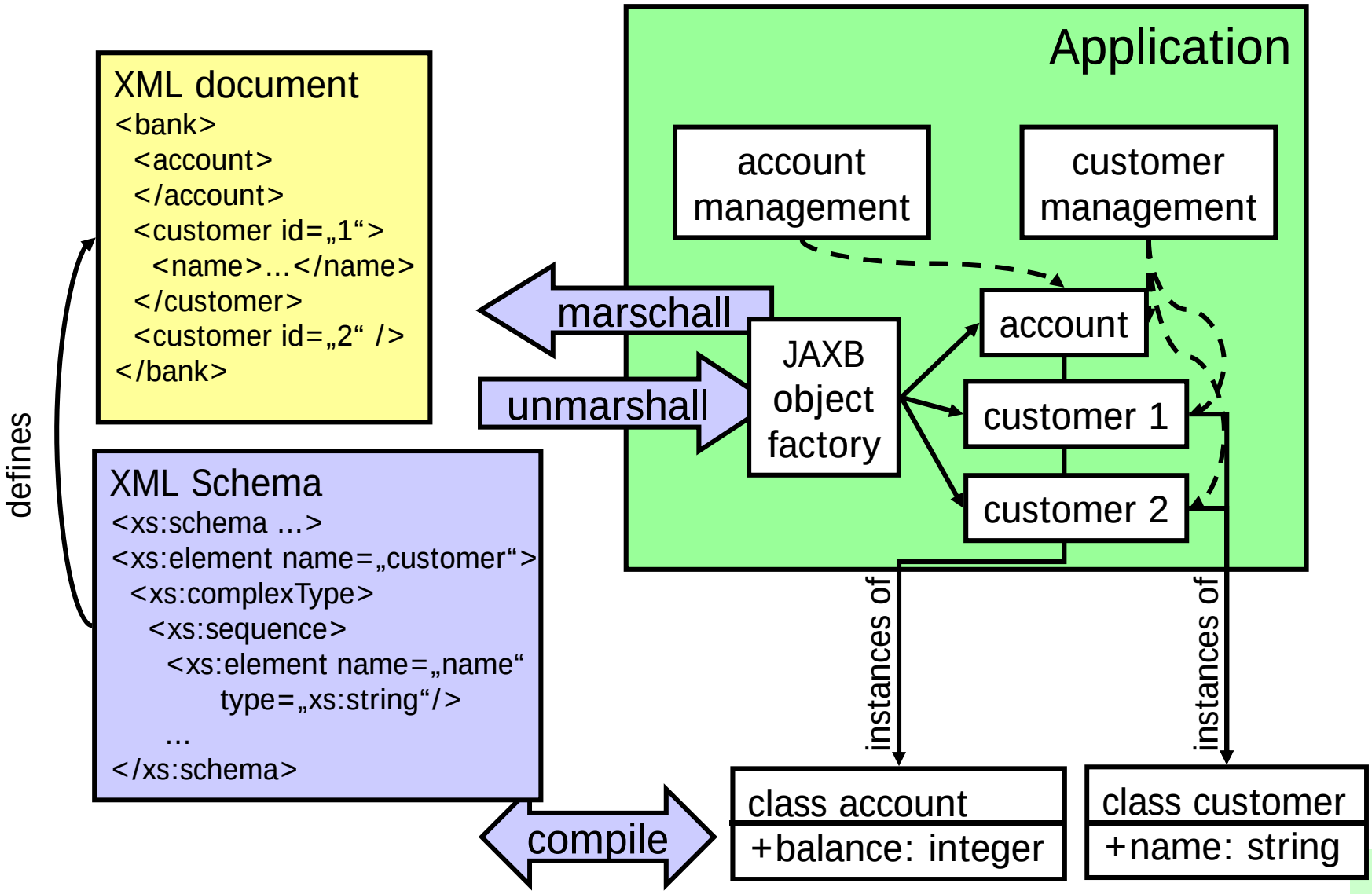
Simplifications

- Attributes
  - Methods of element nodes
    - getAttribute(String name): String
    - setAttributeNode(Attr)
    - setAttribute(String name, String value)
    - removeAttribute(String name)
- Search for elements
  - Methods of element and document nodes
    - getElementsByTagName(String name): NodeList
- Namespace-aware variants of these methods with “NS” suffix
- Access text
  - Method of all nodes
    - getTextContent(): String
  - Basically concatenates values of descendant text nodes
- Get the root element
  - Method of document nodes
    - getDocumentElement(): Element
    - getDoctype(): DocumentType

Overview

- Overview
- DOM
- JAXB – Java Architecture für XML Binding
- SAX
- StAX

Java Architecture for XML Binding (JAXB)



Data Binding – Motivation

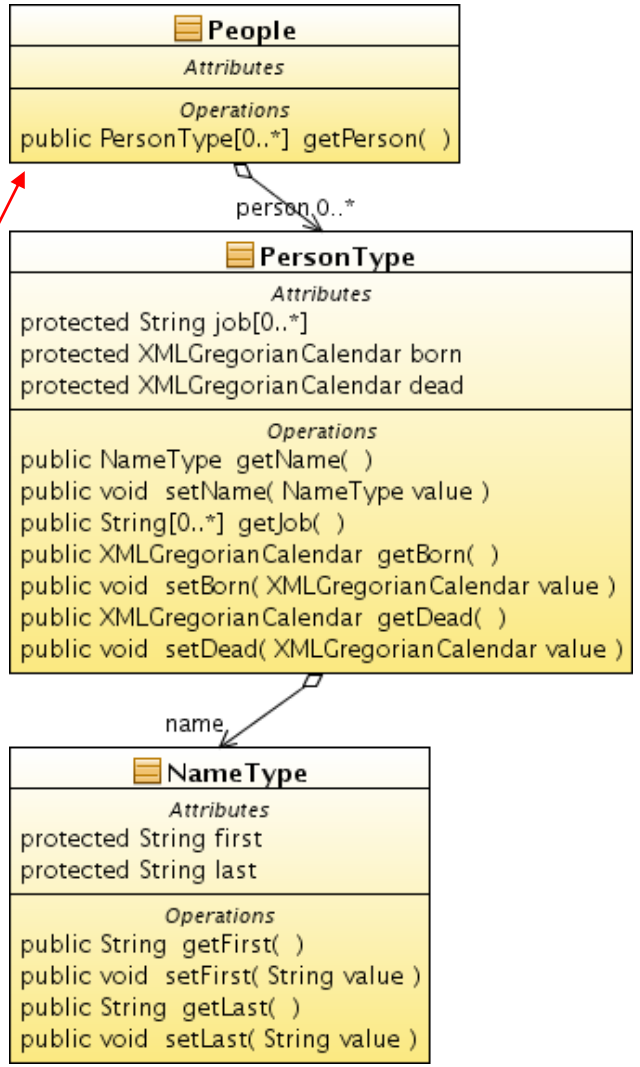
- XML data binding
  - Representation of the XML document in memory (like DOM)
  - Provide more abstraction
  - Reduce implementation overhead by automatic code generation
- Features of JAXB
  - Provides generic marshaller and unmarshaller
  - Generates Java classes from XML schemas
  - Generates XML schemas from Java classes
  - Both generation processes can be customized
  - JAXB 2.1 in Java 6
  - http://jaxb.java.net/

Generating Java Classes (1)

- Compiler generates Java classes from XML schemas
  - xjc -d <target directory> -p <package name> <schema file>
  - Generated Java classes are less strict than the XML schema.
- Bindings for simple types
  - Built-in types are mapped to corresponding Java classes or types.
  - Limited support for derivations
    - Most facets are not represented by Java code
    - XML schema enumerations are represented by Java enumerations
- Bindings for complex types
  - Represented by generated Java classes with the same name
  - XML child elements and XML attributes are represented as Java fields with the same name.
    - No distinction between different compositors
    - Lists where maxOccurs > 1
  - Simple content in field “value”
  - Mixed content in List<Serializable>
  - Derived complex types represented by subclasses

Example for Generating Java Classes

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:p="http://example.org/people"
           targetNamespace="http://example.org/people"
           elementFormDefault="qualified">
  <xs:element name="people">
    <xs:complexType>
      <xs:sequence>
          <xs:element name="person" type="p:personType"
                      minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="personType">
    <xs:sequence>
      <xs:element name="name" type="p:nameType"/>
      <xs:element name="job" type="xs:string"
                  minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="born" type="xs:gYear"/>
    <xs:attribute name="dead" type="xs:gYear"/>
  </xs:complexType>
  <xs:complexType name="nameType">
    <xs:sequence>
      <xs:element name="first" type="xs:string"/>
      <xs:element name="last" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```



Generating Java Classes (2)

- Class JAXBElement<T>
  - Basically a pair of a QName and a value of class T
  - Used in cases where applications cannot (easily) determine the element's name
    - Root elements with simple content
    - xs:choice with maxOccurs > 1
    - Mixed content
    - Substitution groups (JAXBElement<? extends T>)

```
<xs:complexType name="aType">
  <xs:choice maxOccurs="unbounded">
    <xs:element name="b" type="xs:string"/>
    <xs:element name="c" type="xs:string"/>
  </xs:choice>
</xs:complexType>
```

- Schema compiler also creates ObjectFactory
  - Can create instances of all created classes
  - Most useful for JAXBElement
- Annotations in generated classes provide hints for marshalling

Processing Documents with JAXB

name of package of generated classes

```
JAXBContext jc = JAXBContext.newInstance("org.example.people1");
Unmarshaller u = jc.createUnmarshaller();
People p = (People) u.unmarshal(new File("people.xml"));
PersonType turing = p.getPerson().get(0);
System.out.println(turing.getName().getLast());
turing.getJob().add("cryptographer");
Marshaller m = jc.createMarshaller();
m.marshal(p, System.out);
m.marshal(new People(), System.out);
```

- Validation provided by JAXP
  - Create a javax.xml.validation.Schema
  - Turn on validation with (Unmarshaller|Marshaller).setSchema(schema)
- JAXB can be used on top of DOM and StAX parsers
  - Unmarshal methods for org.w3c.Node and javax.xml.stream.XMLStreamReader

Customizing XML Schemas

- Setting the package name
- Replacing types
- …
- Can be specified
  - inline in the schema
  - or in a bindings file.

```
<xs:schema
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
  jaxb:version="2.1"
  …>
  <xs:annotation>
    <xs:appinfo>
      <jaxb:schemaBindings>
          <jaxb:package name="org.example.people1"/>
      </jaxb:schemaBindings>
    </xs:appinfo>
  </xs:annotation>
  …
  <xs:complexType name="personType">
    …
    <xs:attribute name="born" type="p:yearType"/>
    <xs:attribute name="dead" type="p:yearType"/>
  </xs:complexType>
  …
  <xs:simpleType name="yearType">
    <xs:annotation>
      <xs:appinfo>
          <jaxb:javaType name="short"
            parseMethod="java.lang.Short.parseShort"
            printMethod="java.lang.Short.toString"/>
      </xs:appinfo>
    </xs:annotation>
    <xs:restriction base="xs:gYear"/>
  </xs:simpleType>
</xs:schema>
```

Generating XML Schemas

- Compiler generates XML schemas from Java classes
  - schemagen -d <target directory> -cp <class path> <class name...>
- Additional requirements / restrictions
  - Fields must be public or have public getter / setter methods
  - At least annotation for the class of the root element
  - ObjectFactory or jaxb.index file in the package's class directory
- Important features of the schema are arbitrarily chosen.

Example for Generating XML Schemas (1)

```
package org.example.people2;

…
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class People {

    private List<Person> persons;

    public List<Person> getPersons() {
        return persons;
    }

    public void setPersons(List<Person> persons) {
        this.persons = persons;
    }
}
```

```
package org.example.people2;

…

public class Person {

    private String firstName;
    private String lastName;
    private short born;
    private short dead;
    private List<String> job;

    … getters and setters for fields…
}
```

jaxb.index

```
People
Person
```

Example for Generating XML Schemas (2)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="people">
    <xs:sequence>
      <xs:element name="persons" type="person" nillable="true"
                  minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="person">
    <xs:sequence>
      <xs:element name="born" type="xs:short"/>
      <xs:element name="dead" type="xs:short"/>
      <xs:element name="firstName" type="xs:string" minOccurs="0"/>
      <xs:element name="job" type="xs:string" nillable="true"
                  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="lastName" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
  <persons>
    <born>1912</born>
    <dead>1954</dead>
    <firstName>Alan</firstName>
    <job>mathematician</job>
    <job>comp. scientist</job>
    <lastName>Turing</lastName>
  </persons>
</people>
```

Customizing Java Classes

- Controlling order
- Changing names
- Using attributes instead of elements
- Managing namespaces
- …

```
<people xmlns="http://example.org/people">
  <person born="1912" dead="1954">
    <firstName>Alan</firstName>
    <lastName>Turing</lastName>
    <job>mathematician</job>
    <job>comp. scientist</job>
  </person>
</people>
```

```
package org.example.people2;
…
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement
public class People {

    @XmlElement(name = "person")
    private List<Person> persons;

    … getter and setter for field…
}
```

package-info.java

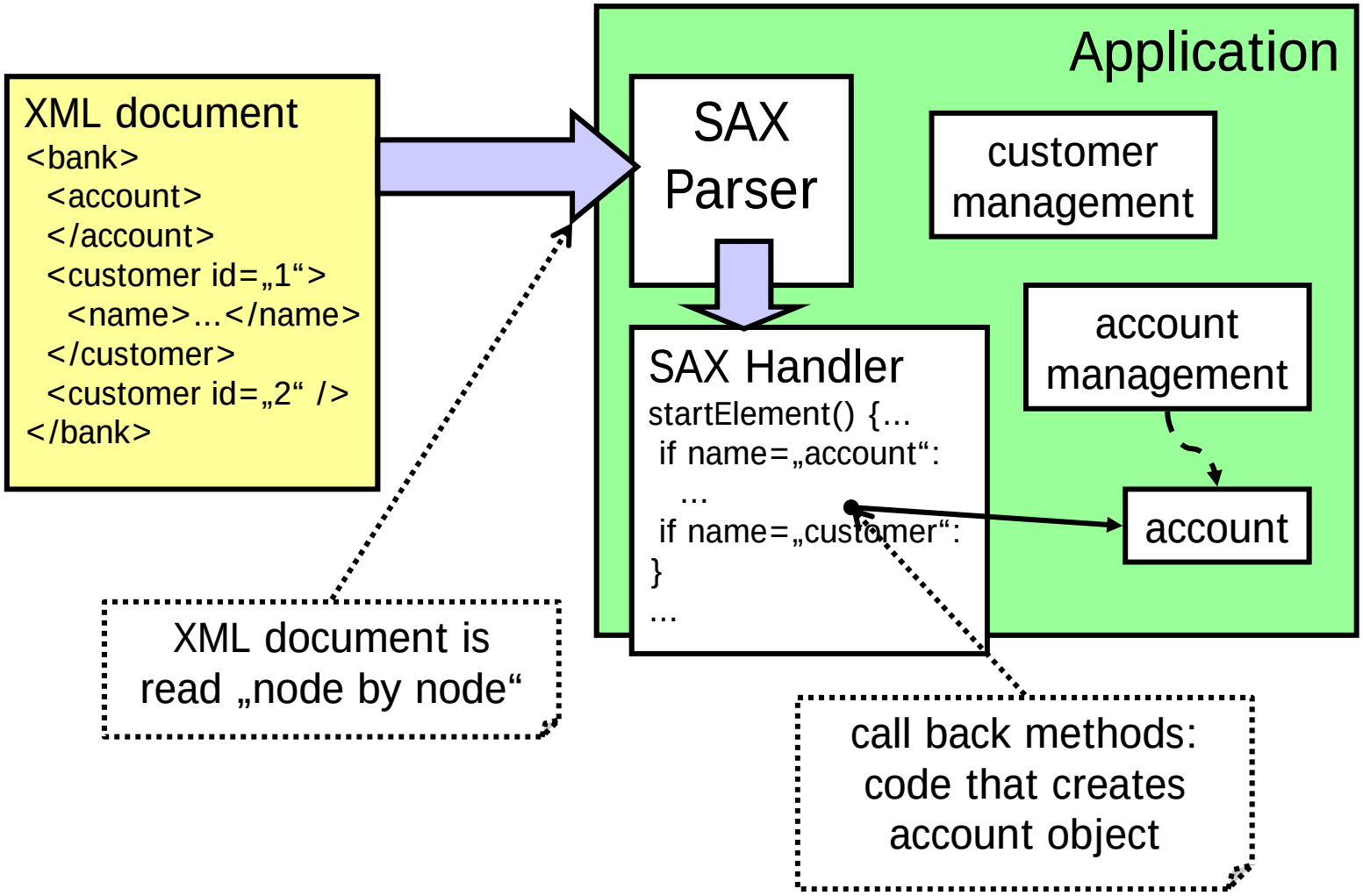
```
@XmlSchema(namespace = "http://example.org/people",
           elementFormDefault = XmlNsForm.QUALIFIED)
package org.example.people2;

import javax.xml.bind.annotation.XmlNsForm;
import javax.xml.bind.annotation.XmlSchema;
```

```
package org.example.people2;
…
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder =
    {"firstName", "lastName", "job"})
public class Person {

    private String firstName;
    private String lastName;
    @XmlAttribute
    private short born;
    @XmlAttribute
    private short dead;
    private List<String> job;

   … getters and setters for fields…
}
```

Overview

- Overview
- DOM
- JAXB
- SAX - Simple API for XML Parsing
- StAX

Simple API for XML (SAX)



SAX Overview

- Simple API for XML Parsing
- De-facto standard
  - Current major version: SAX2
  - Initially developed in Java, now also available for other languages (Perl, Pascal, Python, C++, ...)
  - http://www.saxproject.org/
- Data processing during parsing
  - Event-driven push-parser
- Package org.xml.sax

ContentHandler

- Interface with call-back methods
- Application implements call-backs
- Application registers implementing object at parser
- Invoked during parsing
- Convenience class: DefaultHandler
  - Implements a ContentHandler with empty methods
  - Application defines subclass
    - Small applications are often subclasses of ContentHandler
  - Package: org.xml.sax.helpers
  - Also implements ErrorHandler, DTDHandler and EntityResolver

Methods of ContentHandler (1)

- most important methods for parsing
- document start and end:
  - **startDocument**()
  - **endDocument**()
  - initialize application, ...
- opening and closing tags:
  - **startElement**(String nsURI, String lname, String qname, Attributes attrs)
  - **endElement**(String nsURI, String lname, String qname)
  - arguments are dependent on NamespaceAwareness
- text nodes
  - **characters**(char[] charray, int start, int length)
  - **ignorableWhitespace**(char[] charray, int start, int length)
  - eventually multiple callings for one text node

Methods of ContentHandler (2)

- scope of a namespace
  - **startPrefixMapping**(String prefix, String URI)
    - Called before startElement of declaring element
  - **endPrefixMapping**(String prefix)
    - Called after endElement of declaring element
- methods throw (only) SAXExceptions
  - use exception chaining if necessary

Attributes

- List with attributes of an element
- Length:
  - getLength(): int
- Access to names and values:
  - getURI(int index): String
  - getLocalName(int index): String
  - getQName(int index): String
  - getValue(int index): String
- Searching for attributes
  - getIndex(String qName): int
  - getIndex(String nsURI, String localName): int
  - getValue(String qName): String
  - getValue(String nsURI, String localName): String
- List does not contain attributes for declaring namespaces when parser is namespace aware (by default).

Simple SAX Example

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.helpers.DefaultHandler;

SAXParserFactory parserFactory = SAXParserFactory.newInstance();
parserFactory.setNamespaceAware(false);
parserFactory.setValidating(true);

SAXParser parser = parserFactory.newSAXParser();

DefaultHandler handler = new MyHandler();
parser.parse(new File("bank_dtd.xml"), handler);
```

```
public class MyHandler extends DefaultHandler {

  public void startElement(String uri, String lName, String qName, Attributes a) {
    if ("".equals(uri)) System.out.println("Start element " + qName);
    else System.out.println("Start element: {" + uri "} " + lName);
  }
  public void endDocument() {
    System.out.println("End document");
  }
}
```

Document Locator

- Reference to parser location in document
  - getLineNumber(): int
  - getColumnNumber(): int
- Provided to ContentHandler by parser
  - setDocumentLocator(Locator);
- Invocation by parser once before startDocument()
  - Valid only within call-backs

```
public class MyHandler extends DefaultHandler {
  private Locator loc;

  public void setDocumentLocator(Locator loc) {
    this.loc = loc;
  }
  public void startElement(String uri, String lName, String qName, Attributes a) {
    if ("".equals(uri) System.out.println("Start element " + qName);
    else System.out.println("Start element: {" + uri "} " + lName);
  }
  public void endDocument() {
    System.out.println("Document has " + loc.getLineNumber() + " lines");
  }
}
```

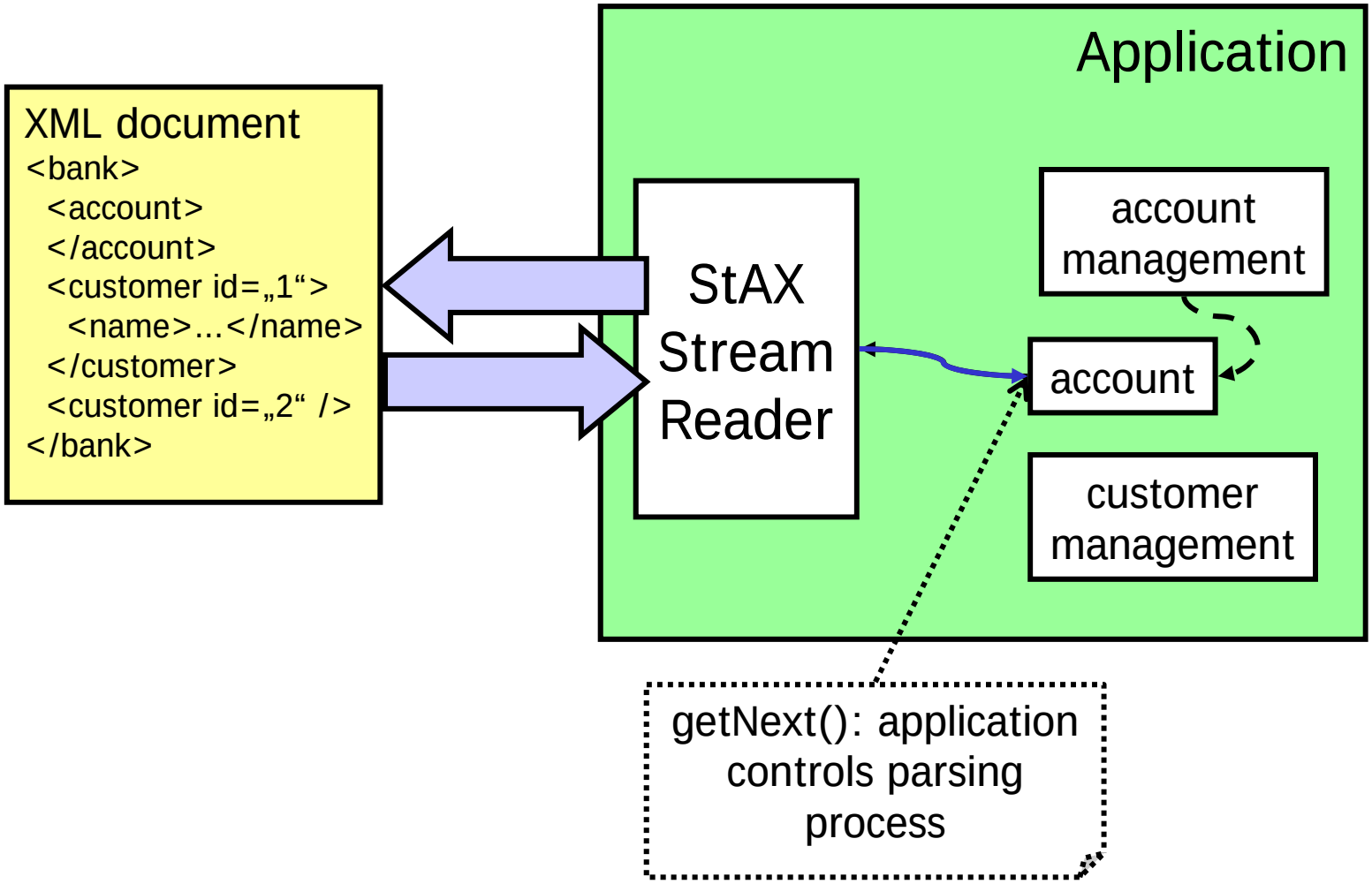
Creating SAX Parsers

- Using JAXP (see previous example)
  - Create a javax.xml.parsers.SAXParser
  - Similar to DOM parsers created with JAXP
- Using SAX helper class
  - Create a org.xml.sax.XMLReader
  - Factory class from org.xml.sax.helpers
  - XMLReader parser = XMLReaderFactory.createXMLReader()
  - XMLReader is configured directly
- SAXParser is a wrapper around an XMLReader
  - SAXParser.getXMLReader()

Overview

- Overview
- DOM
- JAXB
- SAX
- StAX - Streaming API for XML

Streaming API for XML (StAX)



StAX Overview

- Streaming API for XML
- http://stax.codehaus.org/
- Application controls parser
  - Event-driven pull-parser
  - Simplified application development
    - State can be represented by program structure
- Goal
  - As simple to use as DOM
  - As fast as SAX
  - Memory usage as low as with SAX

Two StAX APIs

- Cursor API
  - Lean and efficient
  - Get data directly from XMLStreamReader
- Iterator API
  - Compact and concise
  - Each part of document is an XMLEvent object
- Java Package: javax.xml.stream

StAX Factory Classes

- Create factories using static newInstance() methods
- Configure input and output factories via properties:
  - setProperty(String name, Object value)
- XMLInputFactory
  - Creates XMLStreamReader, XMLEventReader
  - Properties: javax.xml.stream.(isNamespaceAware | isValidating | isCoalescing | ...)
- XMLOutputFactory
  - Creates XMLStreamWriter, XMLEventWriter
  - Property: javax.xml.stream.isRepairingNamespaces
- XMLEventFactory
  - Creates XMLEvents
  - createStartElement(...), createEndElement(...), createAttribute(...), createCharacters(...), ...

Cursor API – Read (1)

- Methods of javax.xml.stream.XMLStreamReader
- Move cursor forward to the next event
  - next(): int
    - Returns the type of this event (constant from XMLStreamConstants): START_ELEMENT, END_ELEMENT, CHARACTERS, ATTRIBUTE, ...
    - Empty elements (<tag/>) also generate two events.
  - hasNext(): boolean
    - Checks if more parsing events are available
- Examine the current event
  - getEventType(): int
  - Name for current event
    - getLocalName() / getNamespaceURI() / getPrefix(): String
    - getName(): QName
  - Content for current event (except for attributes)
    - getText(): String
    - Content may be split into several chunks

XMLStreamReader Example (1)

```
import javax.xml.namespace.QName;

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

XMLInputFactory factory = XMLInputFactory.newInstance();
factory.setProperty("javax.xml.stream.isNamespaceAware", false);

XMLStreamReader reader =
    factory.createXMLStreamReader(new FileReader(new File("bank.xml")));

while(reader.hasNext()) {
  int event = reader.getEventType();
  switch (event) {
  case XMLStreamConstants.START_ELEMENT:
    System.out.println("start element " + reader.getLocalName());
    break;
  …
  }
  reader.next();
}
reader.close();
```

Initial event is START_DOCUMENT!

Cursor API – Read (2)

- ATTRIBUTE event apparently is not used by the cursor API
- Attributes for current event
  - getAttributeCount(): int
  - Access via indexes
    - getAttributeLocalName(int index) / getAttributeNamespace(int index) / getAttributePrefix(int index): String
    - getAttributeName(int index): QName
    - getAttributeValue(int index): String
    - getAttributeType(int index): String
      - Returns the (DTD) type of the attribute
  - Access via name
    - getAttributeValue(String namespaceURI, String localName): String
  - List does not contain attributes for declaring namespaces when parser is namespace aware.

Cursor API – Read (3)

- Namespace declarations
  - Treated as normal attributes when parser is not namespace aware
  - Available for
    - START_ELEMENT: beginning of scope of bindings
    - END_ELEMENT: end of scope of bindings
  - getNamespaceCount(): int
  - getNamespaceURI(int index): String
  - getNamespacePrefix(int index): String
- Methods may only be called for the corresponding events
  - java.lang.IllegalStateException is thrown otherwise
- Whitespace handling
  - isWhiteSpace(): CHARCTERS event contains only whitespace
  - Ignorable whitespace is reported as SPACE event (requires DTD)

Convenience Methods (XMLStreamReader)

- Text content of an element
  - getElementText(): String
  - Comments and processing instructions are discarded
  - Child elements are not allowed
  - Precondition: current event is START_ELEMENT
  - Postcondition: current event is the corresponding END_ELEMENT
- Skip uninteresting content
  - nextTag(): int
  - Moves cursor to next START_ELEMENT or END_ELEMENT
  - Skips all white space, comments, processing instructions
  - Other content is not allowed
- Ensure given state of parser
  - require(int type, String namespaceURI, String localName)
  - Throws XMLStreamException if current event does not match
  - namespaceURI and localName are not compared if null
- Check current event
  - isStartElement() / isEndElement() / isCharacters() / isWhitespace(): boolean

XMLStreamReader Example (2)

```
…

reader.require(XMLStreamConstants.START_DOCUMENT, null, null);
while (reader.next() != XMLStreamConstants.START_ELEMENT); // skip DTD
reader.require(XMLStreamConstants.START_ELEMENT, null, "people");
reader.nextTag();
reader.nextTag();
reader.getElementText(); // skip name element
reader.nextTag();
reader.require(XMLStreamConstants.START_ELEMENT, null, "job");
System.out.println(reader.getElementText());  // prints "mathematician"

…
```

```
<?xml version="1.0"?>
<!DOCTYPE people SYSTEM "people.dtd">
<people>
 <person>
  <name>Alan Turing</name>
  <job>mathematician</job>
  <job>comp. scientist</job>
 </person>
 <person born="1916"
         dead="1988"/>
</people>
```

Cursor API – Write (1)

- Methods of javax.xml.stream.XMLStreamWriter
- Begin and end of document
  - writeStartDocument()
  - writeEndDocument()
- Opening and closing tags
  - writeStartElement(...)
  - writeEndElement()
  - writeEmptyElement(...)
- Text content
  - writeCharacters(String text)
- Attributes
  - writeAttribute(..., String value)
- Methods may only be called for the corresponding events
  - java.lang.IllegalStateException is thrown otherwise

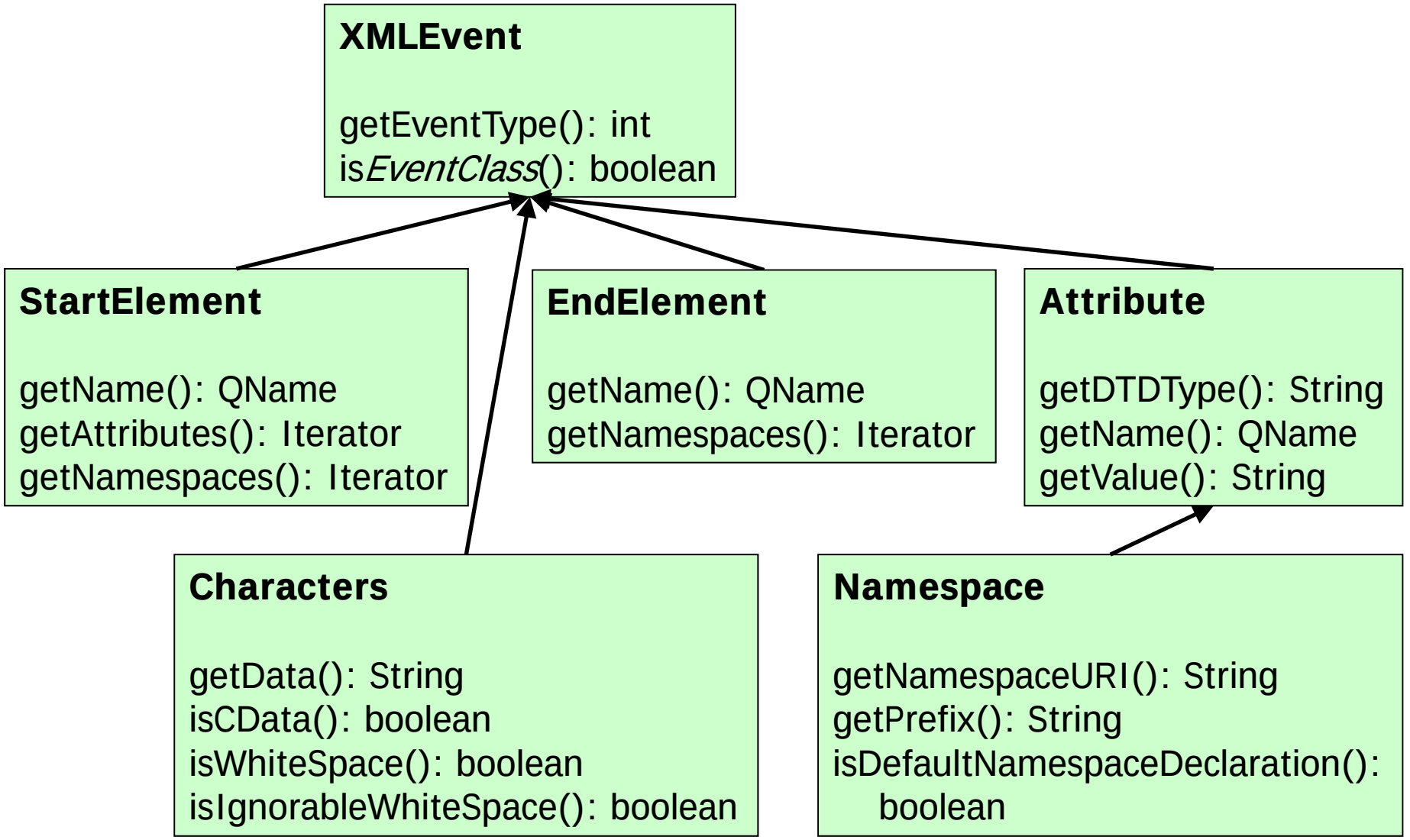
Cursor API – Write (2)

- Different signatures for methods require the specification of names
  - (String name): name without prefix
  - (String namespaceURI, String localName): localName with a prefix bound to namespaceURI
  - (String prefix, String namespaceURI, String localName): localName with prefix
    - Additionally binds prefix to namespaceURI in the XMLStreamWriter
- Binding namespaces
  - in the XMLStreamWriter
    - setPrefix(String prefix, String uri)
    - setDefaultNamespace(String uri)
  - in the XML document
    - writeNamespace(String prefix, String namespaceURI)
    - writeDefaultNamespace(String namespaceURI)
  - Order of method invocations is important
  - Automatic binding when isRepairingNamespaces property is set
- flush() and close()

Iterator API – Read

- Methods of javax.xml.stream.XMLEventReader
- XMLEventReader implements java.util.Iterator
  - hasNext(): boolean
  - next(): Object – actually an XMLEvent
- Additional methods
  - nextEvent(): XMLEvent
    - next() including class cast
  - peek(): XMLEvent
    - Return the next event without moving the iterator
- Methods for examining the current state are in XMLEvent and its subinterfaces.
- XMLEventReader does not generate Attribute or Namespace events

XMLEvent Interface Hierarchy



XMLEventReader Example

```
import javax.xml.namespace.QName;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.events.XMLEvent;
import javax.xml.stream.events.StartElement;

XMLInputFactory factory = XMLInputFactory.newInstance();
factory.setProperty("javax.xml.stream.isNamespaceAware", false);

XMLEventReader reader =
    factory.createXMLEventReader(new FileReader(new File("bank.xml")));
while(reader.hasNext()) {
  XMLEvent event = reader.nextEvent();
  if (event.isStartElement()) {
    StartElement startElement = (StartElement) event;
    System.out.println("start element " + startElement.getName());
  }
  …
}
reader.close();
```

Reader initially before START_DOCUMENT!

Convenience Methods (Iterator API)

- XMLEventReader
  - Same as XMLStreamReader: getElementText(), nextTag()
- XMLEvent
  - Class casts:
    - as*EventType*(): *EventType*
    - May throw ClassCastException
  - Type checks:
    - is*EventType*(): boolean
- StartEvent
  - getAttributeByName(QName name): Attribute

Iterator API – Write

- Methods of javax.xml.stream.XMLEventWriter
  - add(XMLEvent e)
- Creating XMLEvents using an instance of XMLEventFactory
  - create*EventType*(...): *EventType*
  - Different signatures for methods require the specification of names
    - (QName name)
    - (String prefix, String namespaceUri, String localName)
    - (String localName)
    - Not all combinations supported by all methods
  - Two ways for writing attributes (and namespace declarations)
    - Using XMLEventWriter directly: add(Attribute a)
    - Using an iterator over attributes when creating the StartElement event
- Strange features of the API
  - createEndElement(...) takes a namespace iterator as argument
  - Tags without namespaces can only be created using QName()
- flush() and close()

XMLEventWriter Example (1)

```
import javax.xml.namespace.QName;

import javax.xml.stream.XMLEventFactory;
import javax.xml.stream.XMLEventWriter;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;

XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
XMLEventFactory eventFactory = XMLEventFactory.newInstance();

XMLEventWriter writer = outputFactory.createXMLEventWriter(System.out);

writer.add(eventFactory.createStartDocument());
QName people = new QName("people");
writer.add(eventFactory.createStartElement(people, null, null));
QName person = new QName("person");
writer.add(eventFactory.createStartElement(person, null, null));
writer.add(eventFactory.createAttribute("born", "1916"));
writer.add(eventFactory.createAttribute("dead", "1988"));
writer.add(eventFactory.createEndElement(person, null));
writer.add(eventFactory.createEndElement(people, null));
writer.add(eventFactory.createEndDocument());

writer.close();
```

XMLEventWriter Example (2)

```
…
import java.util.HashSet;
import java.util.Set;
…
import javax.xml.stream.events.Attribute;
…

writer.add(eventFactory.createStartDocument());
QName people = new QName("people");
writer.add(eventFactory.createStartElement(people, null, null));
QName person = new QName("person");
Set<Attribute> attrs = new HashSet<Attribute>();
attrs.add(eventFactory.createAttribute("born", "1916"));
attrs.add(eventFactory.createAttribute("dead", "1988"));
writer.add(eventFactory.createStartElement(person, attrs.iterator(), null));
writer.add(eventFactory.createEndElement(person, null));
writer.add(eventFactory.createEndElement(people, null));
writer.add(eventFactory.createEndDocument());

writer.close();
```

Utility Classes for Namespaces

- Package javax.xml.namespace
- NamespaceContext
  - Accumulated results of namespace bindings
  - Retrieve current namespace context of XMLStreamReader, XMLStreamWriter or StartElement
    - getNamespaceContext(): NamespaceContext
    - Read-only and transient
  - getNamespaceURI(String prefix): String
  - getPrefix(String namespaceURI): String
    - If namespaceURI is bound to multiple prefixes, chooses one
  - getPrefixes(String namespaceURI): Iterator
- QName
  - Immutable triple of namespaceURI, localPart, prefix
  - Three costructors:
    - QName(String localPart)
    - QName(String namespaceURI, String localPart)
    - QName(String namespaceURI, String localPart, String prefix)

Cursor API vs. Iterator API

- Cursor API
  - XMLStreamReader, XMLStreamWriter
  - XML data stored directly inside reader or writer
  - Large number of methods which can only be called in certain states of the reader or writer
  - Creates only a few new objects during reading or writing, more efficient
- Iterator API
  - XMLEventReader, XMLEventWriter
  - XML data stored in XMLEvent objects
  - More object-oriented approach, improved usability
  - XMLEvents can easily be stored

Comparison

	DOM	JAXB	SAX	StaX
API type	parse tree	XML binding to Java objects	event, push	event, pull
Arbitrary access	yes	yes	no	no
Memory efficiency	bad	bad	good	good
Write XML documents	yes	yes	no	yes
Validation	yes	yes	yes	(yes)
State management	program	program	variables	program