Basics of programming 3

XML handling in Java

XML intro

- e**X**tensible **M**arkup **L**anguage
- Goal: standard, human readable data format
  - storing
  - transmitting
- backed by the success of HTML
- text-based
  - cf. `toString()`
- Today: „Whatever the question, XML is the answer„
  - or JSON…

XML history

- GML
  - generalized markup language
  - IBM, '60
  - document markup

```
:h1.Chapter 1:  Introduction
   :p.GML supported hierarchical containers, such as
   :ol
   :li.Ordered lists (like this one),
   :eol.
   as well as simple structures.
   :p.Markup minimization allowed the end-tags to be
omitted for the "h1" and "p" elements.
```

XML history

- SGML
  - standard generalized markup language
  - 1986
  - encyclopedias, dictionaries (OED), databases
  - XML-like
  - DTD is invented

XML history

- HTML
  - hypertext markup language
  - 1991
  - web pages
    - breakthrough
    - earlier gopher, etc.
  - not flexible enough
    - compatibility issues
    - *"this page is optimized for ObscureBrowser 11.3a"*

XML features

- Tree structure: hierarchy
  - tags, attributes and text
- W3C standard
  - syntax
  - parsing rules
  - well-formedness and validity
- Meta-structure, can be extended

XML well-formedness

- Syntax rules
  - optional header

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

  - each tag has a closing counterpart

```
<p>Hello <img src=„kitty.png"/> </p>
```

  - there is a root
  - tags sequentially or embedded

XML syntax

- comment

```
<!- this is a comment -->
```

- tag attribute

```
<img src="cat.png"/>
```

  - always use quotation marks

- special signs

XML	text
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

XML validity

- Semantic rules
- Constraints can be given as…
  - schema
  - DTD (document type definition)
- Specifies the structure of a document
  - allowed elements
  - allowed hierarchy
  - allowed attributes
  - ...

XML semantics specification

- DTD
  - old
  - missing features
  - deprecated
- XML schema (XSD)
  - new
  - structured: elements and connections
  - namespaces
  - in XML itself

DTD example (XML file)

```
<?xml version="1.0" encoding="UTF-8"?>
<people_list>
  <person>
    <name>Neil Armstrong</name>
    <birthdate>1930-08-05</birthdate>
    <gender>Male</gender>
  </person>
  <person>
    <name>Buzz Aldrin</name>
    <birthdate>1930-01-20</birthdate>
    <gender>Male</gender>
    <neptuncode>MOON02</neptuncode>
  </person>
</people_list>
```

DTD example (DTD description)

```
<!ELEMENT people_list (person*)>
<!ELEMENT person (name, birthdate, gender?,
       neptuncode?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT neptuncode(#PCDATA)>
```

XSD example

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="people_list">
    <xs:complexType><xs:sequence>
        <xs:element name="person" maxOccurs="unbounded">
          <xs:complexType><xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="birthdate" type="xs:string"/>
            <xs:element name="gender" type="xs:string"
                minOccurs="0"/>
            <xs:element name=„neptuncode" type="xs:string"
                minOccurs="0"/>
          </xs:sequence></xs:complexType>
        </xs:element>
      </xs:sequence></xs:complexType>
  </xs:element>
</xs:schema>
```

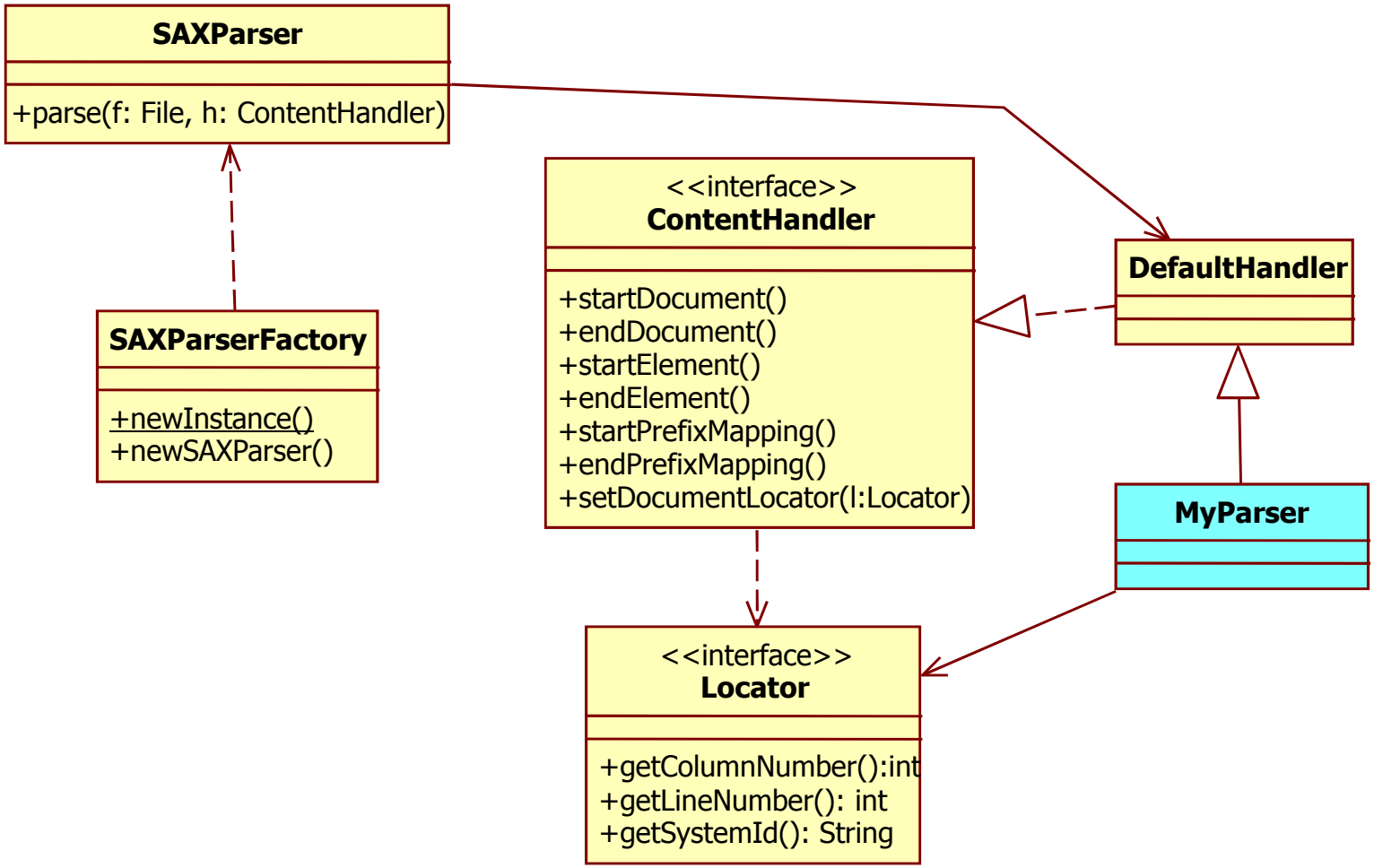
XML handling

- manual
  - simply don’t
- SAX *(simple API for XML)*
  - event based
  - sequential
- DOM *(Document Object Model)*
  - dumb object structure
- JDOM *(Java DOM)*
  - smart object structure

SAX parser usage

- Event handling
  - start/end of document
  - start/end of tag
  - start/end of prefixMapping
  - characters (text)
  - whitespace
  - skipped entities
  - processing instructions

SAX parser classes



ContentHandler interface

- `org.xml.sax.ContentHandler`
- This is to be implemented
- Callback-based event handling
  - for each event a method is provided
- Empty implementation
  - `org.xml.sax.helpers.DefaultHandler`
  - all methods are empty

ContentHandler

- `void startDocument()`
- `void endDocument()`

- `void startElement(String uri, String localName, String qName, Attributes atts)`
- `void endElement(String uri, String localName, String qName)`

- `void startPrefixMapping(String prefix, String uri)`
- `void endPrefixMapping(String prefix)`

ContentHandler

- `void characters(char[] ch, int start, int length)`
- `void ignorableWhitespace(char[] ch, int start, int length)`
- `void processingInstruction(String target, String data)`
- `void skippedEntity(String name)`
- `void setDocumentLocator(Locator locator)`
  - locator can get access to further processing data

ContentHandler example

- Problem:
  - Let's create a simple Java application
  - that prints out the XML tree
    - attributes are omitted
    - texts are omitted

ContentHandler example

- Solution:
  - implement interface *ContentHandler*
    - using class *DefaultHandler*
  - register handler
  - parse the XML file

ContentHandler example

```
public class MyParser extends DefaultHandler {

    public static void main(String[] args) {
        DefaultHandler h = new MyParser();
        SAXParserFactory factory =
            SAXParserFactory.newInstance();
        try {
            SAXParser p = factory.newSAXParser();
            p.parse(new java.io.File(args[0]), h);
        } catch (Exception e) {e.printStackTrace();}
    }

    ...
```

ContentHandler example

```
...
int tab=0;
public void println(String s) {
    for (int i = 0; i < tab; i++) {
        System.out.print("  ");
    }
    System.out.println(s);
}
public void startDocument() throws SAXException {
    println("Start document");
}
public void endDocument() throws SAXException {
    println("End document");
}
...
```

ContentHandler example

```
    ...
    public void startElement(String namespaceURI,
       String sName, String qName, Attributes attrs)
       throws SAXException {
         tab++;
         println("start element: "+qName);
    }

    public void endElement(String namespaceURI,
       String sName, String qName)
       throws SAXException {
         println("end element: "+qName);
         tab--;
    }
}
```

ContentHandler example input

```
<!-- test.xml -->
<level1>
        <level2>
                <level3 attr1="test1">
                </level3>
                <level3 attr1="test2" attr2="second">
                </level3>
                <level3 attr1="test3">
                </level3>
        </level2>
</level1>
```

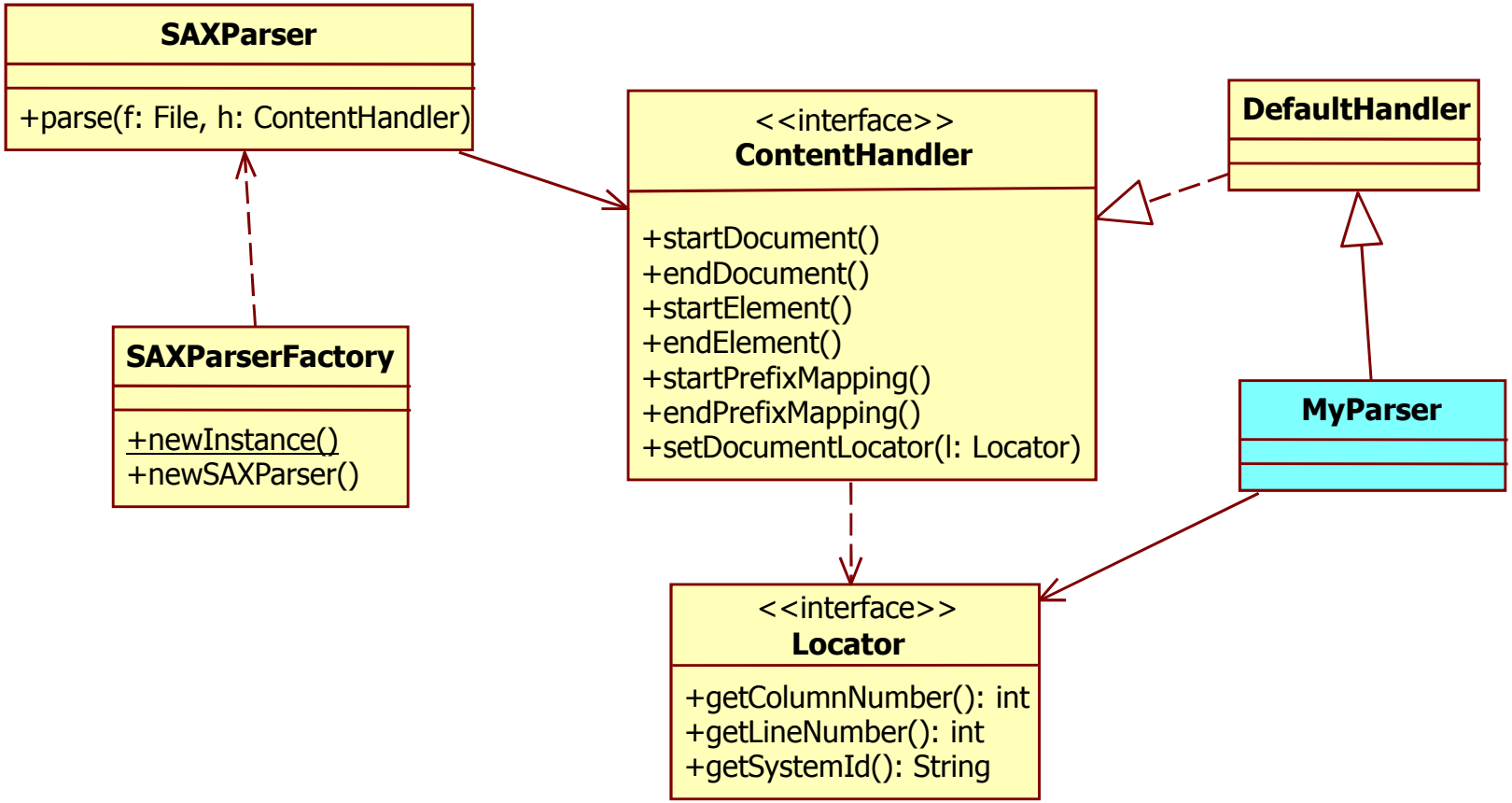
ContentHandler example output

```
$ java MyParser test.xml
Start document
  start element: level1
    start element: level2
      start element: level3
      end element: level3
      start element: level3
      end element: level3
      start element: level3
      end element: level3
    end element: level2
  end element: level1
End document
```

Locator

- Provides information about processed file
- `void setDocumentLocator(Locator l)`
  - `int getColumnNumber()`
    - index of character in current line the handler is processing
  - `int getLineNumber()`
    - index of line being processed
  - `String getSystemId()`
    - name of document (e.g. filename) as a URL

Locator classes



Locator example

```
Locator loc = null;
public void setDocumentLocator(Locator l) {
    println("LOCATOR");
    loc = l;
}
```

```
public void startElement(String namespaceURI,
   String sName, String qName, Attributes attrs)
   throws SAXException {
     tab++;
     println("start element: "+qName);
     println("  Locator: ("+loc.getLineNumber()
       +","+loc.getColumnNumber()+") "
       +loc.getPublicId()+", "+loc.getSystemId());
     ...
}
```

Locator example input

```
<!-- test.xml -->
<level1>
        <level2>
                <level3 attr1="test1">
                </level3>
                <level3 attr1="test2"
attr2="second">
                </level3>
                <level3 attr1="test3">
                </level3>
        </level2>
</level1>
```

Locator example output

```
$ java MyParser test.xml
LOCATOR
Start document
  start element: level1
    Locator: (1,9) null,
file:/home/balage/sax-example/example4/test.xml
    start element: level2
      Locator: (3,10) null,
file:/home/balage/sax-example/example4/test.xml
      start element: level3
        Locator: (4,25) null,
file:/home/balage/sax-example/example4/test.xml
        attr0: attr1=test1
      end element: level3
...
```

Document validity

- Let's add validation

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true);
factory.setNamespaceAware(true);
```

```
SAXParser p = factory.newSAXParser();
String JAXP_SCHEMA_LANGUAGE =
 "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
String W3C_XML_SCHEMA =
 "http://www.w3.org/2001/XMLSchema";
p.setProperty(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
```

XML error handling

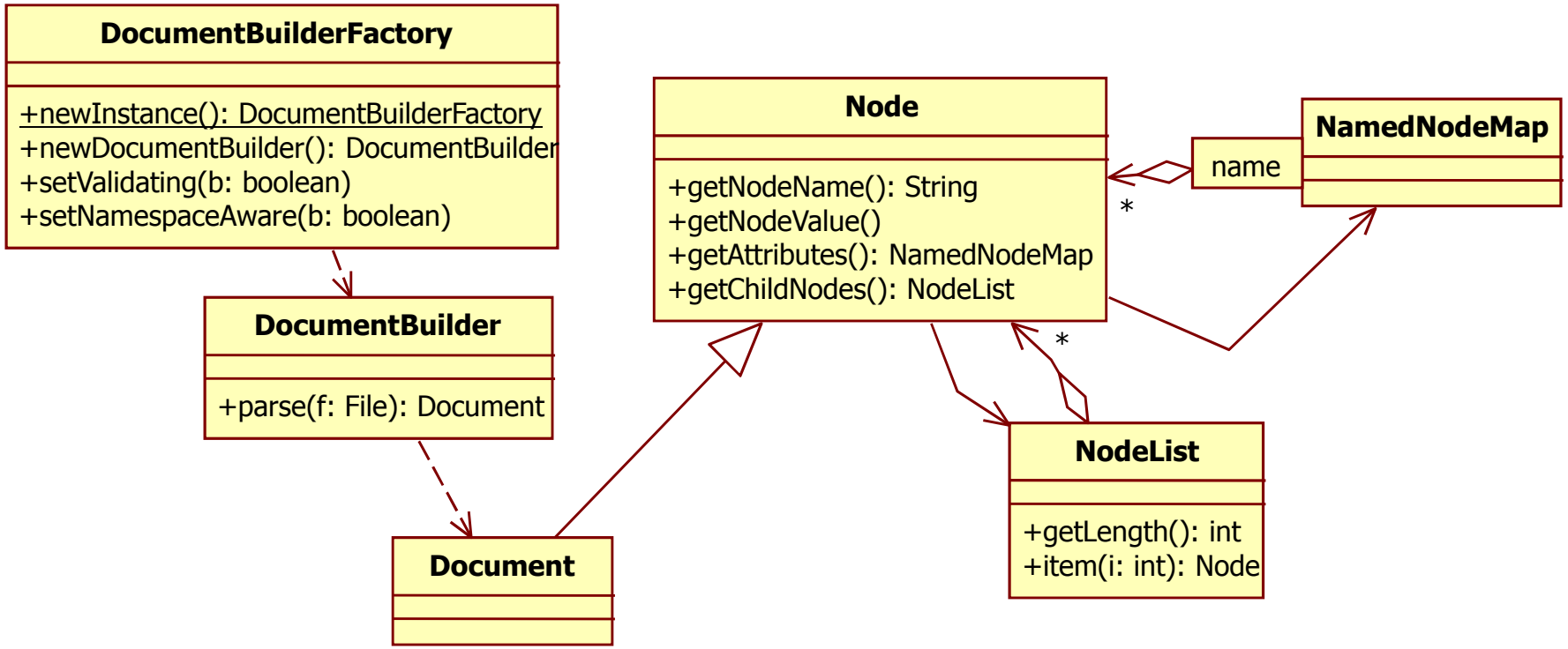
- Error types
  - fatal error
    - document is not well formed
  - error
    - document is not valid
  - warning
    - small error, e.g. same type declared twice

Document Object Model

DOM

- *Document Object Model*
  - builds an object model based on the XML content
- Object model can be modified
  - and printed out as XML
- Validation included

DOM classes


DocumentBuilderFactory
+newInstance(): DocumentBuilderFactory
+newDocumentBuilder(): DocumentBuilder
+setValidating(b: boolean)
+setNamespaceAware(b: boolean)
DocumentBuilder
+parse(f: File): Document
Document
Node
+getNodeName(): String
+getNodeValue()
+getAttributes(): NamedNodeMap
+getChildNodes(): NodeList
name
NamedNodeMap
*
*
NodeList
+getLength(): int
+item(i: int): Node

DOM introductory example

```
try {
  DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
  factory.setValidating(true);
  factory.setNamespaceAware(true);

  DocumentBuilder builder =
    factory.newDocumentBuilder();

  Document document =
    builder.parse(new java.io.File(args[0]));
  ...
} catch (Exception e) {
  e.printStackTrace();
}
```

Document

- Represents the document
  - this is not the root element!
- Base data can be accessed
  - doctype, XML version, etc
- Can create elements
  - attribute, element, comment, etc.
- Is itself a `Node`

Node

- An element of the tree
  - element, attribute, comment, text, etc.
- Its data can be accessed and modified
  - name, type, value, etc.
- Is navigable
  - up: parent and document
  - down: children
  - sideways: siblings
- Children can be modified (add, delete)

DOM helper classes

- `NodeList`
  - when accessing the children of a `Node`
    - `NodeList Node.getChildNodes()`
  - `int getLength()`
    - length of the list
  - `Node item(int index)`
    - returns element at index *index*

DOM helper classes

- `NamedNodeMap`
  - for accessing the attributes of a `Node`
    - `NamedNodeMap Node.getAttributes()`
  - `int getLength()`
    - size of the set
  - `Node item(int index)`
    - Node at index
  - `Node getNamedItem(String name)`
  - `Node getNamedItemNS(String namespaceURI, String localName)`
    - Node with the given name

DOM helper classes

- `NamedNodeMap` (cont.)
  - `Node removeNamedItem(String name)`
  - `Node removeNamedItemNS(String namespaceURI, String localName)`
    - removes the element from the set and the `Node`
    - swapped for default value if specified
  - `Node setNamedItem(Node arg)`
  - `Node setNamedItemNS(Node arg)`
    - modifies the value of the item
    - stores according to attribute `nodeName`

Example: simple printout

```
static void print(Node n, String tab) {
  System.out.println(tab+"("+n.getNodeName()+") \""
    +n.getNodeValue()+"\"");
  if (n.hasAttributes()) {
    NamedNodeMap map = n.getAttributes();
    for (int i = 0; i < map.getLength(); i++) {
      Node n1 = map.item(i);
      print(n1, tab+"attr: ");
    }
  }
  NodeList nl = n.getChildNodes();
  for (int i = 0; i < nl.getLength(); i++) {
    Node n1 = nl.item(i);
    print(n1, tab+"  ");
  }
}
```

Recursion

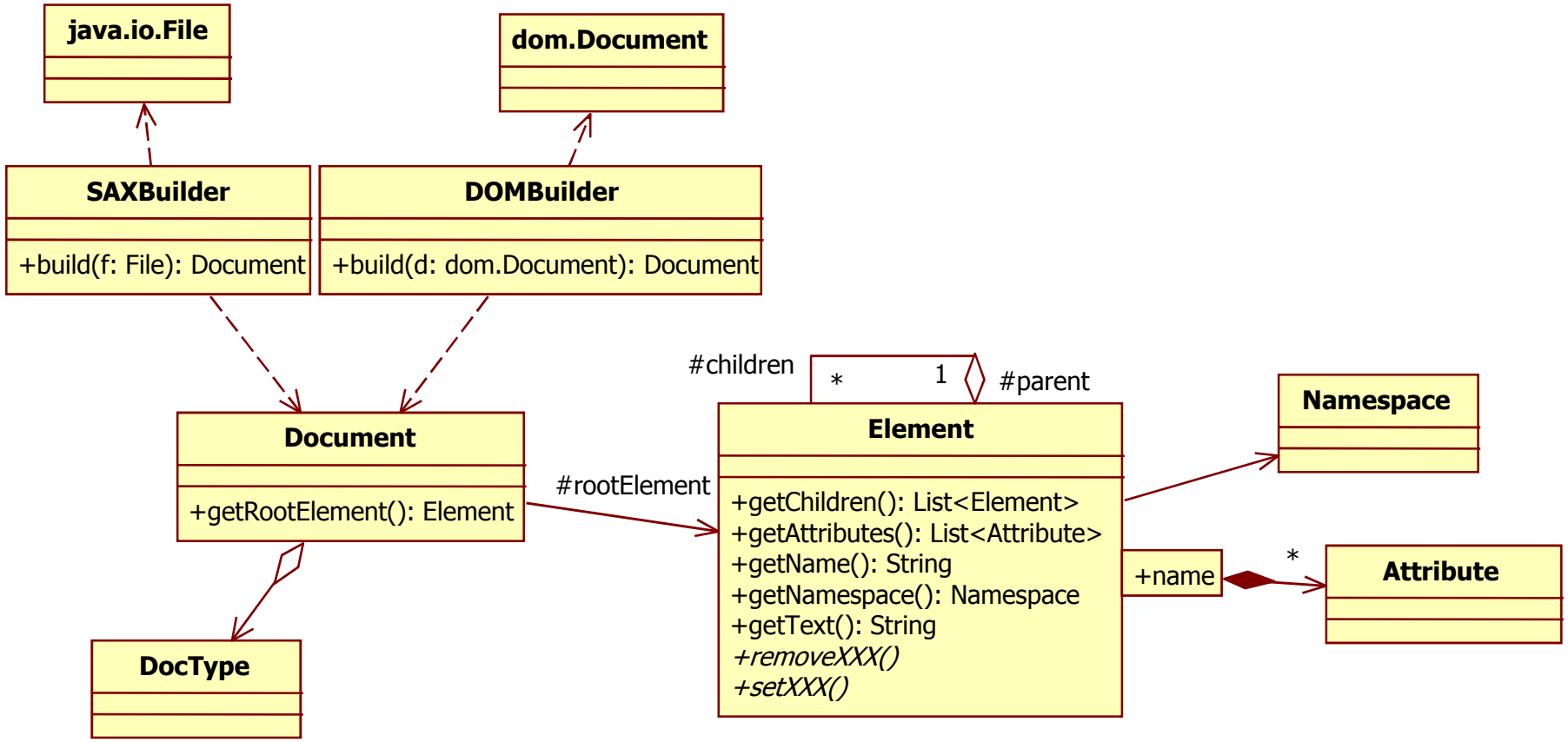
DOM specialities

- Name of XML *tag* is the name of the `Node`
  - `getNodeName()`
- Value of XML *tag* is value of a child of the `Node`
  - child called `#text`
  - `getNodeValue()`
- Name and value of XML attribute is the name and value of the `Node`
  - also has a child with name `#text` that stores the value

JDOM

- Object model closer to XML
  - `Attribute, CDATA, Comment, Content, DefaultJDOMFactory, DocType, Document, Element, EntityRef, Namespace, ProcessingInstruction, Text`
- Attributes, children are easier to access, modify
  - no `NamedNodeMap`, `NodeList`
  - uses `java.util.List`

JDOM classes (partial)



JDOM

- Filters can be specified for searching
  - `org.jdom.filter.Filter`
    - `boolean matches(java.lang.Object obj)`
- Supports XSL transformations
  - `org.jdom.transform.XSLTranformer`
- Supports XPATH searches
  - `org.jdom.xpath.XPath`

JDOM

- Parsing is out-sourced
  - `DOMBuilder`
  - `SAXBuilder`
- Document can be saved
  - as XML document
  - as a DOM model
  - using SAX event-generator

JDOM Example: simple printout

```
public class JDOMParse {
  static void print(Element n, String tab) ...
  public static void main(String[] args) {
      SAXBuilder b = new SAXBuilder();
      File f = new File("test.xml");
          try {
              Document doc = (Document)b.build(f);
              Element r = doc.getRootElement();
              print(r, "");
          } catch (IOException io) {
              System.out.println(io.getMessage());
          } catch (JDOMException je) {
              System.out.println(je.getMessage());
          }
    }
}
```

JDOM Example: simple printout

```
static void print(Element n, String tab) {
    System.out.println(tab+"("+n.getName()
       +") \""+n.getValue()+"\"");
    if (n.hasAttributes()) {
        List<Attribute> list = n.getAttributes();
        for (Attribute a : list) {
            System.out.println(tab+"attr: "+a);
        }
    }
    List<Element> nl = n.getChildren();
    for (Element e : nl) {
        print(e, tab+"  ");
  }
}
```

Recursion