<lecture>

BFST - First Year Project, Spring 2015

Danmarkskort: Visualisering, Navigation, Søgning og Ruteplanlægning

Lecture 5: XML, SAX, OSM og andre TBF’er

Troels Bjerre Sørensen

based on slides by Anders Møller and Michael Schwartzbach

Plan for today

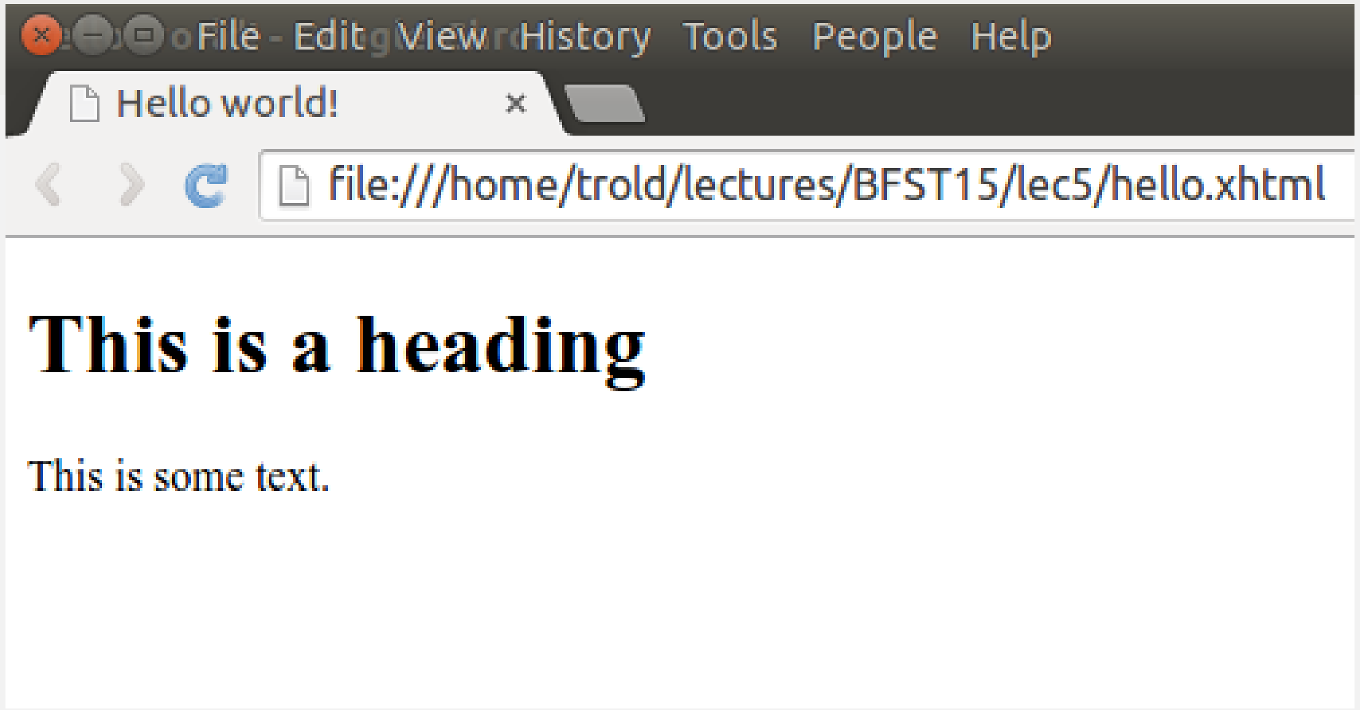
- Theory
  - Introduction to XML
  - The XML Data Model
  - The XML namespace mechanism
- Practice
  - SAX parsers in Java
  - The Open Street Maps format

What is XML?

- XML: ”Extensible Markup Language”
- A framework for defining markup languages
- Each language is targeted at its own application domain with its own markup tags
- There is a common set of generic tools for processing XML documents
- Inherently internationalized and platform independent (Unicode)
- Developed by W3C, standardized in 1998

Example: XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Hello world!</title></head>
  <body>
    <h1>This is a heading</h1>
    This is some text.
  </body>
</html>
```



This is a heading

This is some text.

XML for collecting recipes

- Define our own “Recipe Markup Language”

- Choose markup tags that correspond to concepts in this application domain:
  - recipe, ingredient, amount, ... ($\sim$ OO analysis)

- No canonical choices: ($\sim$ language design)
  - granularity of markup? ($\sim$ level of abstraction)
  - structuring?
  - elements or attributes?
  - ...

Example (1/2)

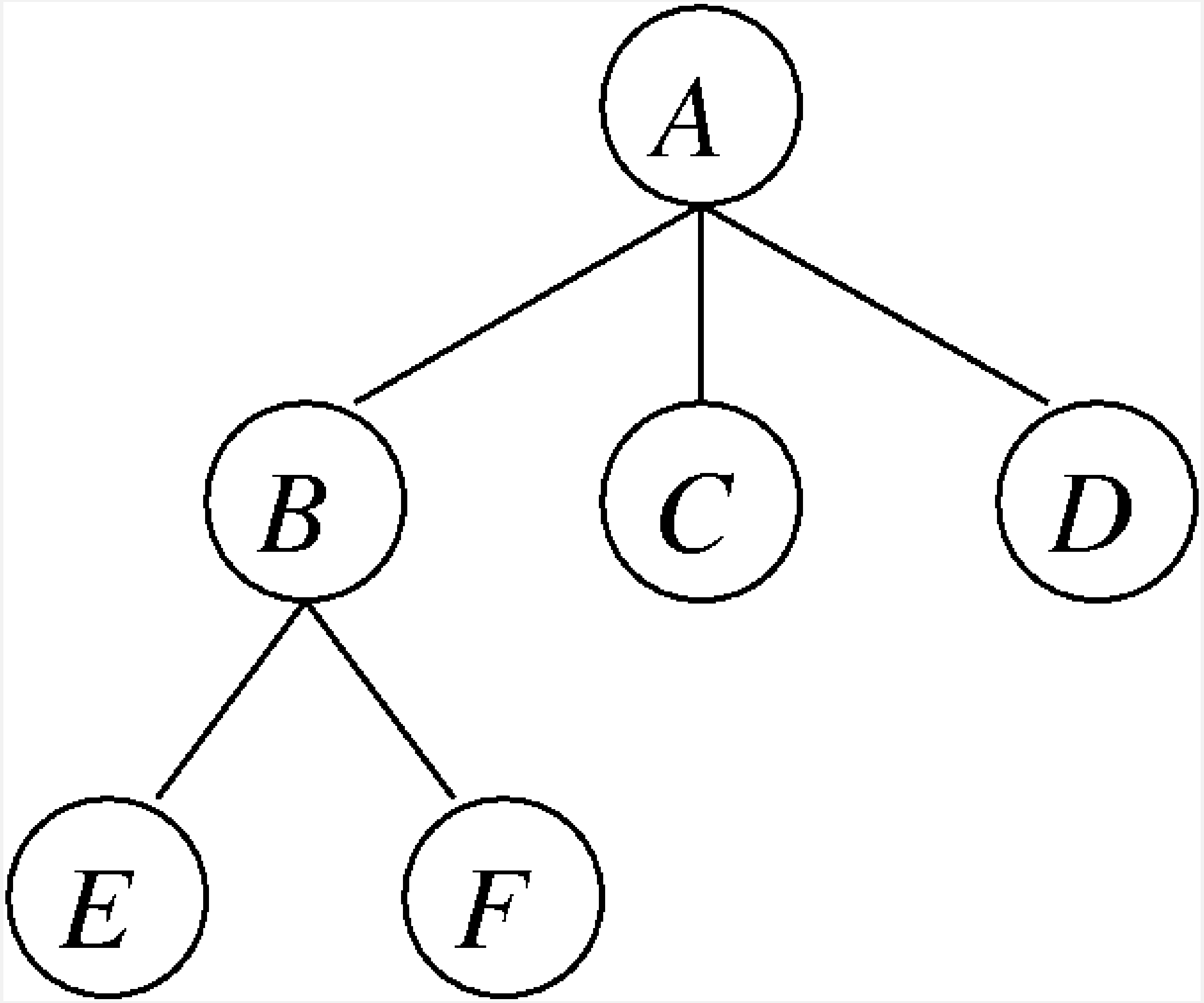
```
<collection>
  <description>Recipes suggested by Jane Dow</description>

  <recipe id="r117">
    <title>Rhubarb Cobbler</title>
    <date>Wed, 14 Jun 95</date>

    <ingredient name="diced rhubarb" amount="2.5" unit="cup"/>
    <ingredient name="sugar" amount="2" unit="tablespoon"/>
    <ingredient name="fairly ripe banana" amount="2"/>
    <ingredient name="cinnamon" amount="0.25" unit="teaspoon"/>
    <ingredient name="nutmeg" amount="1" unit="dash"/>

    <preparation>
      <step>
        Combine all and use as cobbler,
        pie, or crisp.
      </step>
  </preparation>
```

Example (2/2)

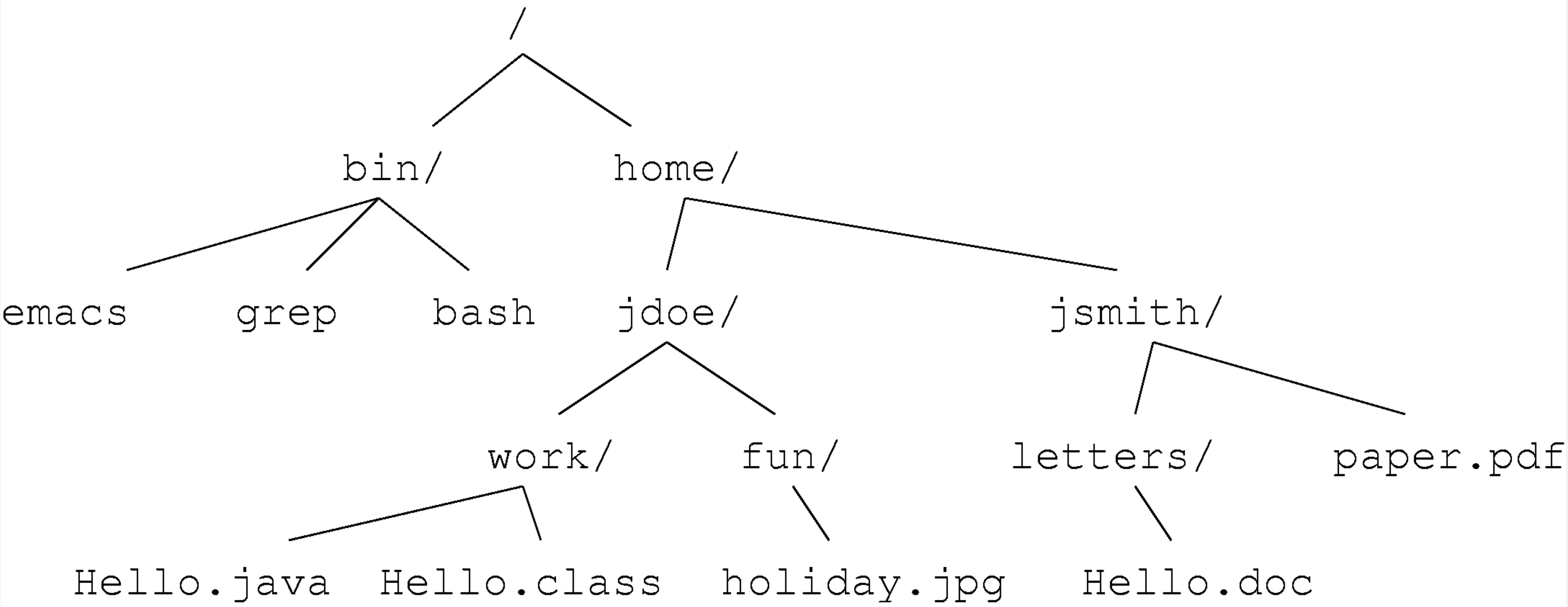
```xml
    <comment>
      Rhubarb Cobbler made with bananas as the main sweetener.
      It was delicious.
    </comment>

    <nutrition calories="170" fat="28%"
      carbohydrates="58%" protein="14%"/>
    <related ref="42">Garden Quiche is also yummy</related>
  </recipe>
</collection>
```

XML Trees

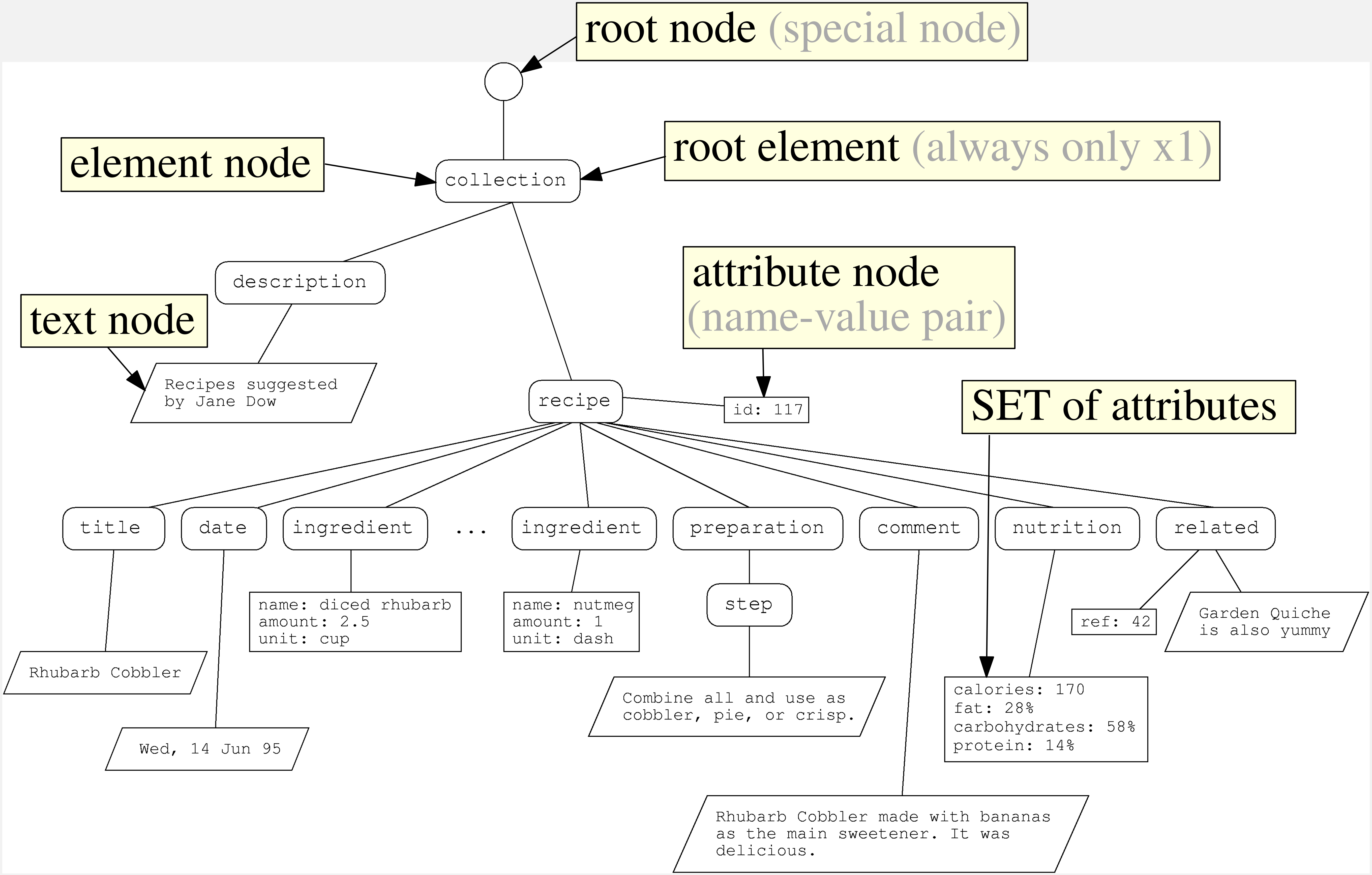
- Conceptually, an XML document is a tree structure
  - node, edge
  - root, leaf
  - child, parent
  - sibling (ordered),
  - ancestor,
  - descendant



An Analogy: File Systems



Tree View of the XML Recipes



root node (special node)

root element (always only x1)

element node

attribute node (name-value pair)

text node

SET of attributes

Nodes in XML Trees

- **Text nodes:**
  text data without explicit structure
- **Element nodes:**
  define hierarchical logical groupings of contents, each have a name
- **Attribute nodes:**
  unordered, each associated with an element node, has a name and a value
- **Comment nodes:**
  ignorable meta-information
- **Processing instructions:**
  instructions to specific processors, each have a target and a value
- **Root nodes:**
  every XML tree has one root node that represents the entire tree

Textual representation

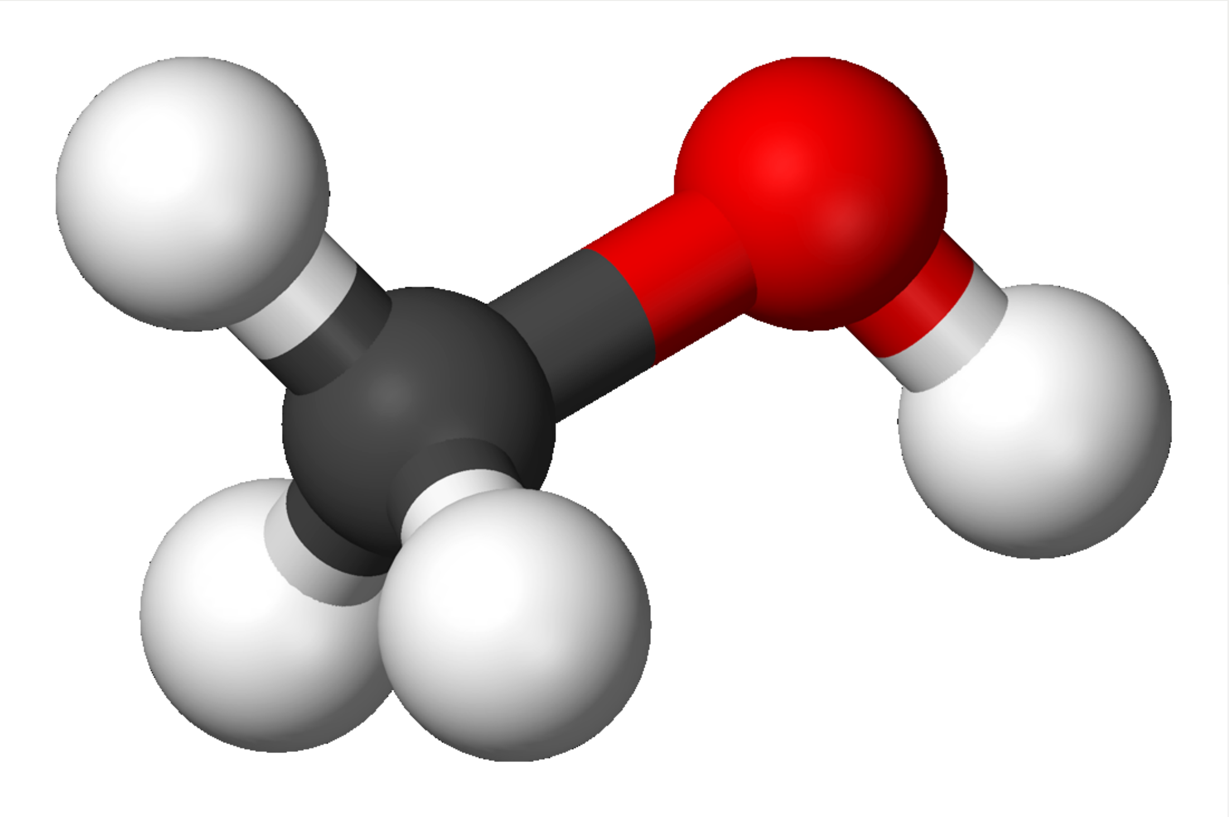
- **Text nodes:**
  written as the text they carry
- **Element nodes:** start-end tags
  - `<bla ...> ...  </bla>`
  - short-hand notation for empty elements: `<bla/>`
- **Attribute nodes:** `name="value"` in start tags
- **Comment nodes:** `<!-- bla -->`
- **Processing instructions:** `<?target value?>`
- **Root nodes:** implicit

Well-formedness

- Every XML document must be well-formed
  - start and end tags must match and nest properly
    - ✓ `<x><y></y></x>`
    - ~~`</z><x><y></x></y>`~~
  - exactly one root element
  - ...
- in other words, it defines a proper tree structure
- **XML parser:**
  given the textual XML document, extract the structure and data

Example: CML (Chemical Markup Language)

```
<molecule id="METHANOL">
 <atomArray>
  <stringArray builtin="id">a1 a2 a3 a4 a5 a6</stringArray>
  <stringArray builtin="elementType">C O H H H H</stringArray>
  <floatArray builtin="x3" units="pm">
    -0.748 0.558 ...
  </floatArray>
  <floatArray builtin="y3" units="pm">
    -0.015 0.420 ...
  </floatArray>
  <floatArray builtin="z3" units="pm">
    0.024 -0.278 ...
  </floatArray>
 </atomArray>
</molecule>
```



Example: ebXML (Electronic Business eXtensible Markup Language)

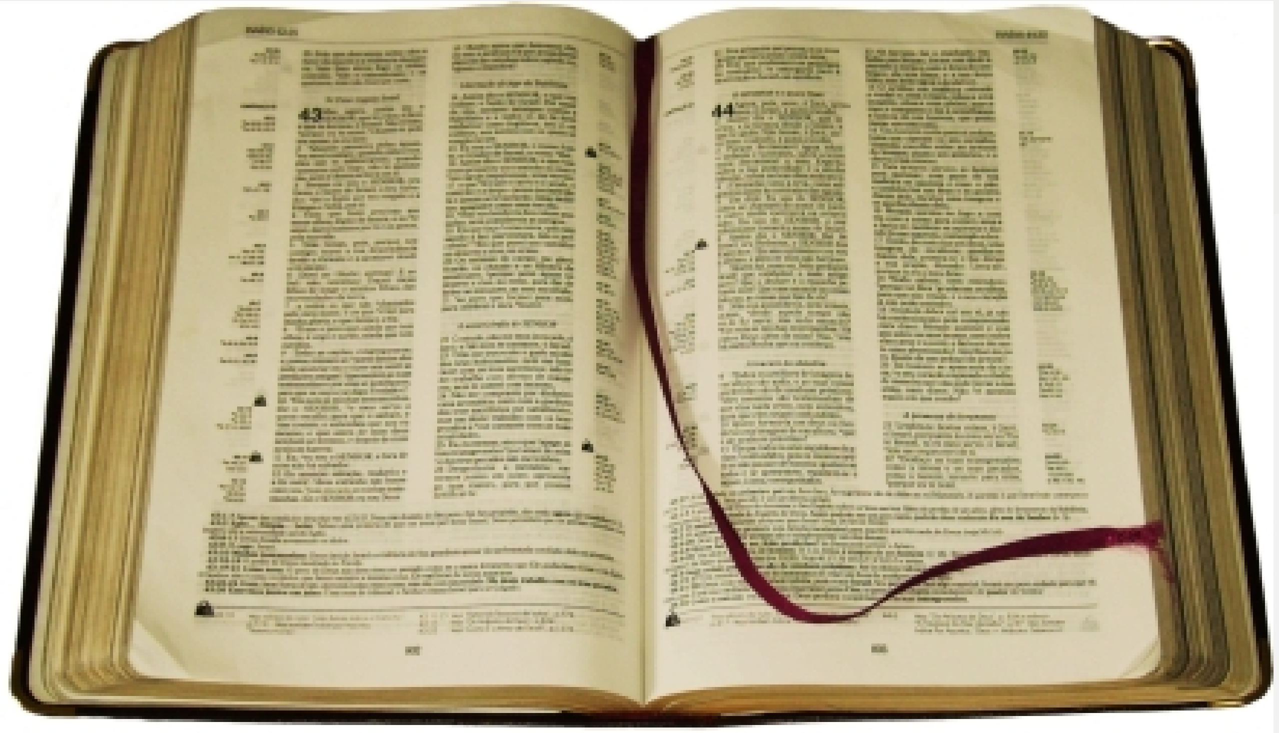
```
<MultiPartyCollaboration name="DropShip">
 <BusinessPartnerRole name="Customer">
  <Performs initiatingRole='//binaryCollaboration[@name="Firm Order"]/
                            InitiatingRole[@name="buyer"]' />
 </BusinessPartnerRole>
 <BusinessPartnerRole name="Retailer">
  <Performs respondingRole='//binaryCollaboration[@name="Firm Order"]/
                            RespondingRole[@name="seller"]' />
  <Performs initiatingRole='//binaryCollaboration[...]/
                            InitiatingRole[@name="buyer"]' />
 </BusinessPartnerRole>
 <BusinessPartnerRole name="DropShip Vendor">
  ...
 </BusinessPartnerRole>
</MultiPartyCollaboration>
```

We Support

ebXML

Example: ThML (Theological Markup Language)

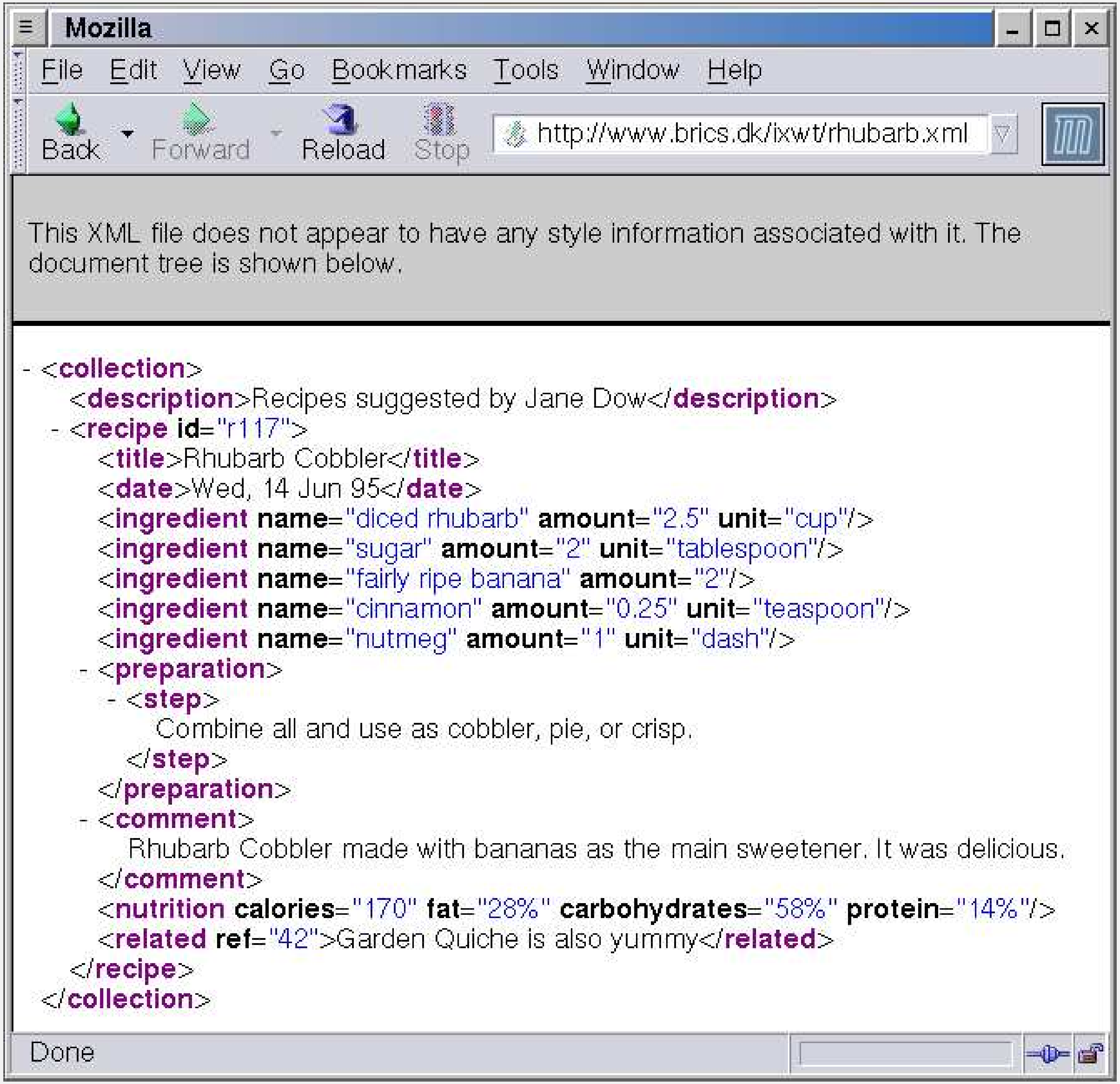
```
<h3 class="s05" id="One.2.p0.2">Having a Humble Opinion of Self</h3>
<p class="First" id="One.2.p0.3">EVERY man naturally desires knowledge
  <note place="foot" id="One.2.p0.4">
    <p class="Footnote" id="One.2.p0.5"><added id="One.2.p0.6">
      <name id="One.2.p0.7">Aristotle</name>, Metaphysics, i. 1.
    </added></p>
  </note>;
  but what good is knowledge without fear of God? Indeed a humble
  rustic who serves God is better than a proud intellectual who
  neglects his soul to study the course of the stars.
  <added id="One.2.p0.8"><note place="foot" id="One.2.p0.9">
    <p class="Footnote" id="One.2.p0.10">
      Augustine, Confessions V. 4.
    </p>
  </note></added>
</p>
```



Exercise: Wake up!

- In groups of ~ 3
- Write an XML file with the minutes of a (fictional) meeting
- Include as much structure as you can
- Try to use all the tag types...

Browsing XML

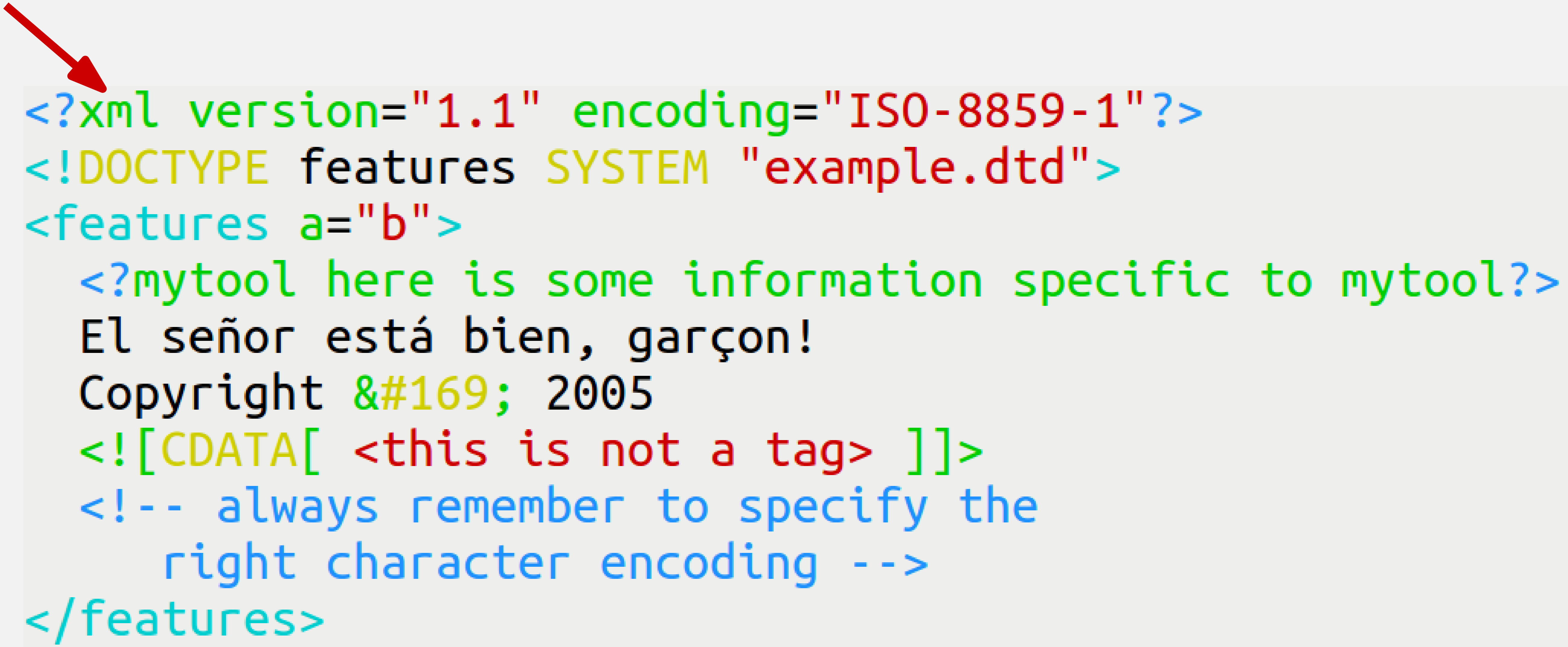

Mozilla
File Edit View Go Bookmarks Tools Window Help
Back
Forward
Reload
Stop
http://www.brics.dk/ixwt/rhubarb.xml
Done

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <collection>
    <description>Recipes suggested by Jane Dow</description>
  - <recipe id="r117">
      <title>Rhubarb Cobbler</title>
      <date>Wed, 14 Jun 95</date>
      <ingredient name="diced rhubarb" amount="2.5" unit="cup"/>
      <ingredient name="sugar" amount="2" unit="tablespoon"/>
      <ingredient name="fairly ripe banana" amount="2"/>
      <ingredient name="cinnamon" amount="0.25" unit="teaspoon"/>
      <ingredient name="nutmeg" amount="1" unit="dash"/>
    - <preparation>
      - <step>
          Combine all and use as cobbler, pie, or crisp.
        </step>
      </preparation>
    - <comment>
        Rhubarb Cobbler made with bananas as the main sweetener. It was delicious.
      </comment>
      <nutrition calories="170" fat="28%" carbohydrates="58%" protein="14%"/>
      <related ref="42">Garden Quiche is also yummy</related>
    </recipe>
  </collection>
```

More Constructs

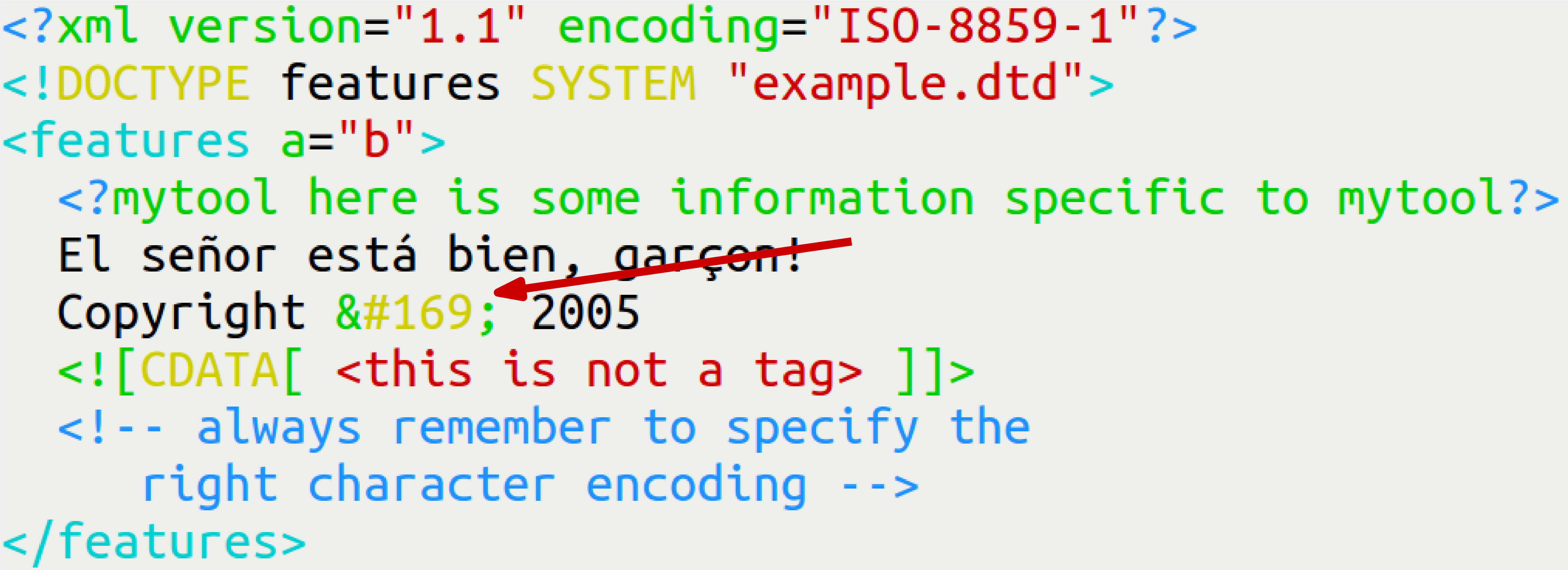
- XML declaration



```
<?xml version="1.1" encoding="ISO-8859-1"?>
<!DOCTYPE features SYSTEM "example.dtd">
<features a="b">
  <?mytool here is some information specific to mytool?>
  El señor está bien, garçon!
  Copyright &#169; 2005
  <![CDATA[ <this is not a tag> ]]>
  <!-- always remember to specify the
     right character encoding -->
</features>
```

More Constructs

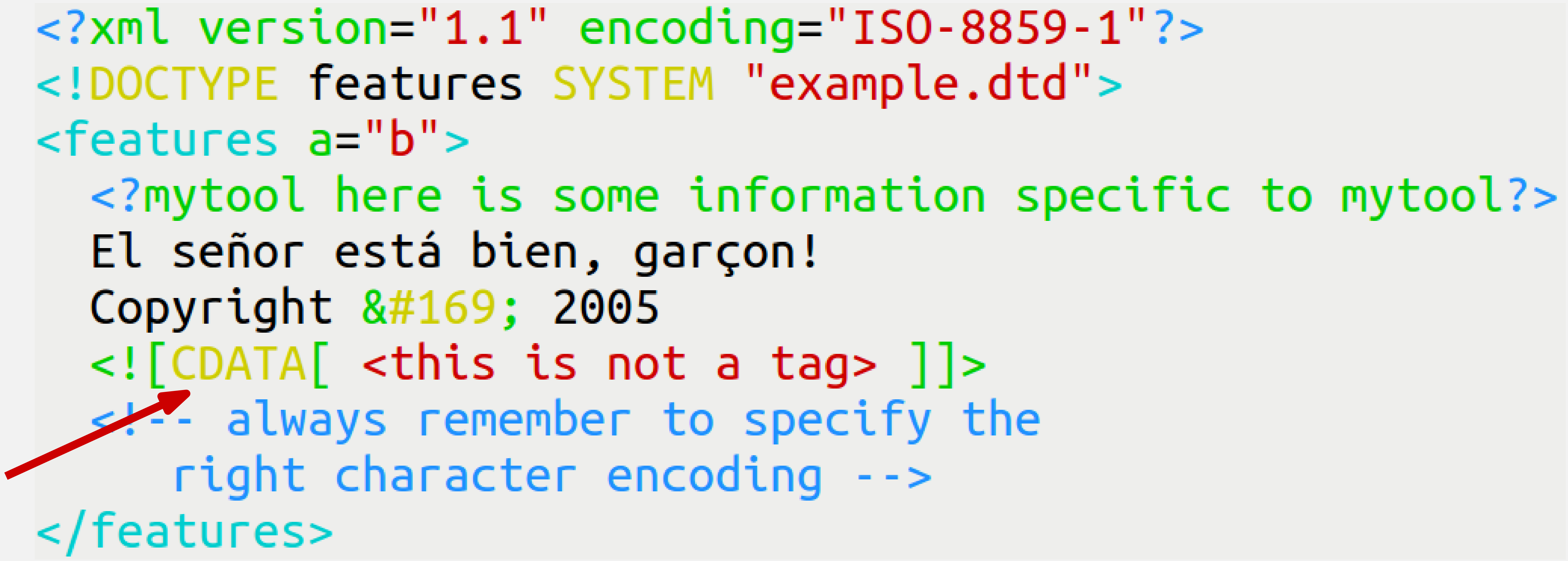
- XML declaration
- Character references



```
<?xml version="1.1" encoding="ISO-8859-1"?>
<!DOCTYPE features SYSTEM "example.dtd">
<features a="b">
  <?mytool here is some information specific to mytool?>
  El señor está bien, garçon!
  Copyright &#169; 2005
  <![CDATA[ <this is not a tag> ]]>
  <!-- always remember to specify the
     right character encoding -->
</features>
```

More Constructs

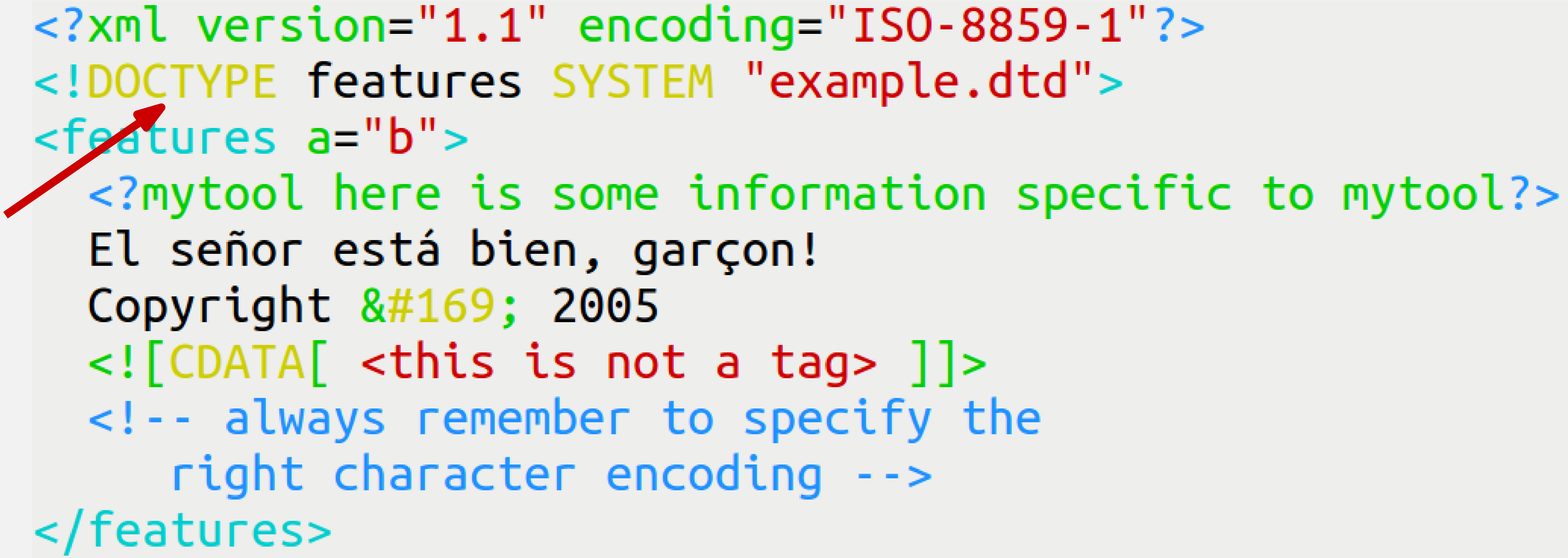
- XML declaration
- Character references
- CDATA sections



```
<?xml version="1.1" encoding="ISO-8859-1"?>
<!DOCTYPE features SYSTEM "example.dtd">
<features a="b">
  <?mytool here is some information specific to mytool?>
  El señor está bien, garçon!
  Copyright &#169; 2005
  <![CDATA[ <this is not a tag> ]]>
  <!-- always remember to specify the
     right character encoding -->
</features>
```

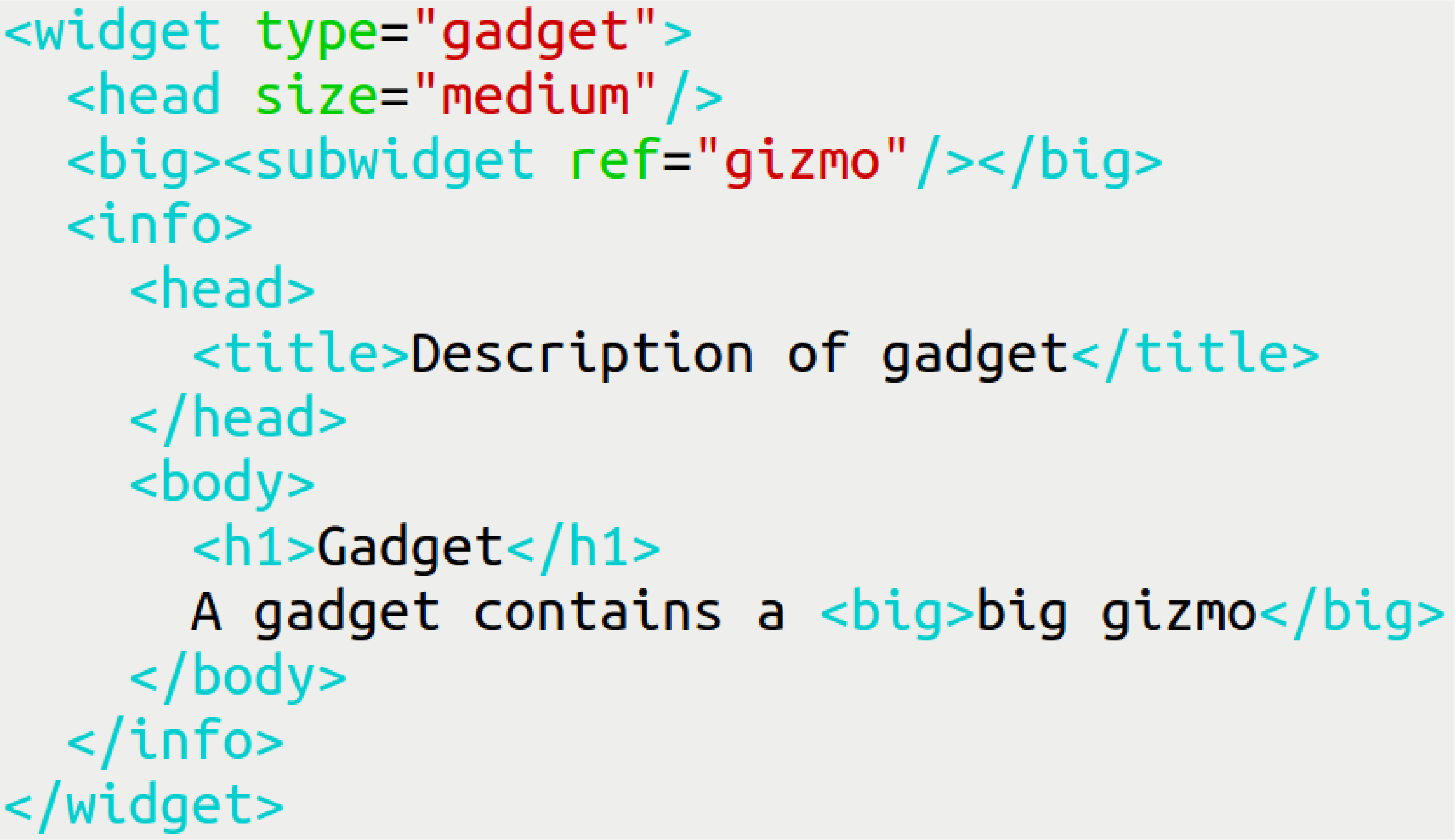
More Constructs

- XML declaration
- Character references
- CDATA sections
- Document type



```
<?xml version="1.1" encoding="ISO-8859-1"?>
<!DOCTYPE features SYSTEM "example.dtd">
<features a="b">
  <?mytool here is some information specific to mytool?>
  El señor está bien, garçon!
  Copyright &#169; 2005
  <![CDATA[ <this is not a tag> ]]>
  <!-- always remember to specify the
     right character encoding -->
</features>
```

XML Namespaces



```
<widget type="gadget">
  <head size="medium"/>
  <big><subwidget ref="gizmo"/></big>
  <info>
    <head>
      <title>Description of gadget</title>
    </head>
    <body>
      <h1>Gadget</h1>
      A gadget contains a <big>big gizmo</big>
    </body>
  </info>
</widget>
```

- When combining languages, element names may become ambiguous!
- Common problems call for common solutions

The Idea

- Assign a URI to every (sub-)language
  e.g. `http://www.w3.org/1999/xhtml` for XHTML 1.0

- Qualify element names with URIs:
  `<{http://www.w3.org/1999/xhtml}head>`

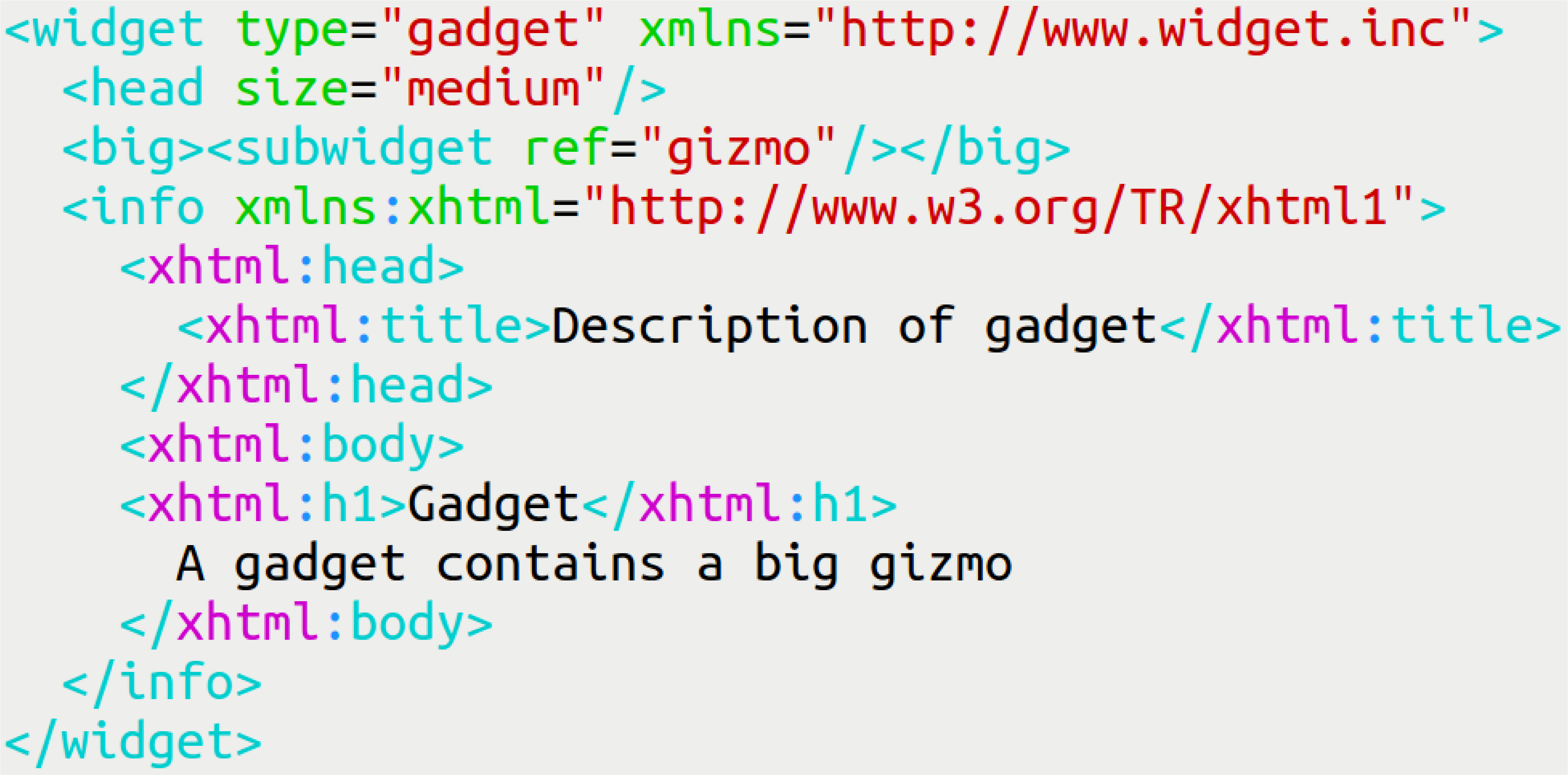
The actual solution

- Namespace declarations bind prefixes to URIs:

```
<... xmlns:foo="http://www.w3.org/TR/xhtml1">
   ...
   <foo:head>...</foo:head>
   ...
</...>
```

- Lexical scope
- Default namespace (no prefix) declared with xmlns=”...“
- Attribute names can also be prefixed

Widgets with namespaces



```
<widget type="gadget" xmlns="http://www.widget.inc">
  <head size="medium"/>
  <big><subwidget ref="gizmo"/></big>
  <info xmlns:xhtml="http://www.w3.org/TR/xhtml1">
    <xhtml:head>
      <xhtml:title>Description of gadget</xhtml:title>
    </xhtml:head>
    <xhtml:body>
    <xhtml:h1>Gadget</xhtml:h1>
      A gadget contains a big gizmo
    </xhtml:body>
  </info>
</widget>
```

- Namespace map: for each element, maps prefixes to URIs

XML parsing

- Two overall approaches:
  - Tree-based (e.g. DOM)
  - Streaming-based (e.g. SAX)
- Tree-based approaches keeps the entire XML tree in memory
- Often, data does not fit in memory, or can only be streamed
- What is streaming for XML documents?
- The SAX framework has the answer...

Exercise! Wake up!

I will show you an XML document.

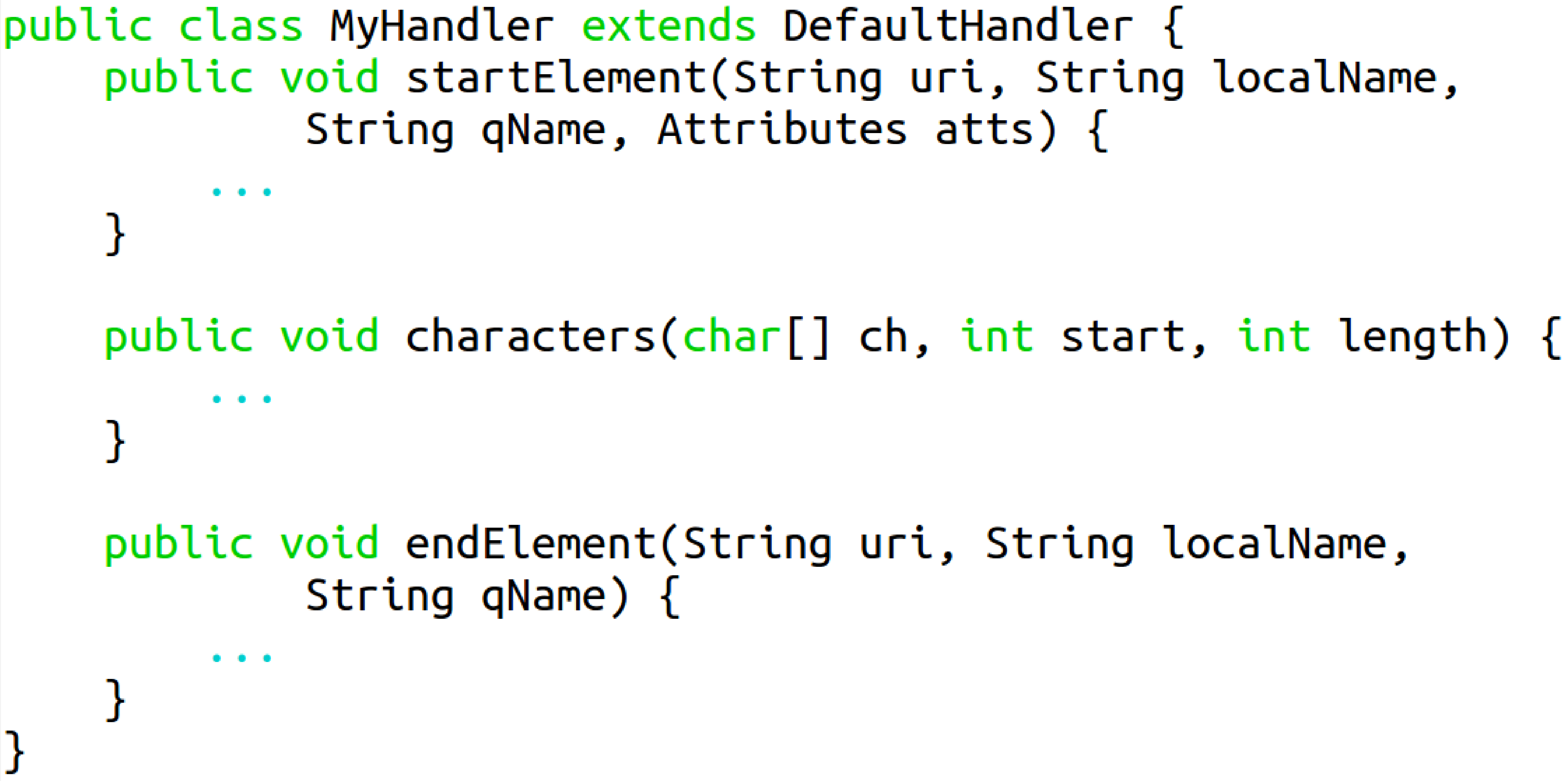
In little pieces.

You must compute it’s height.

Event based parsing

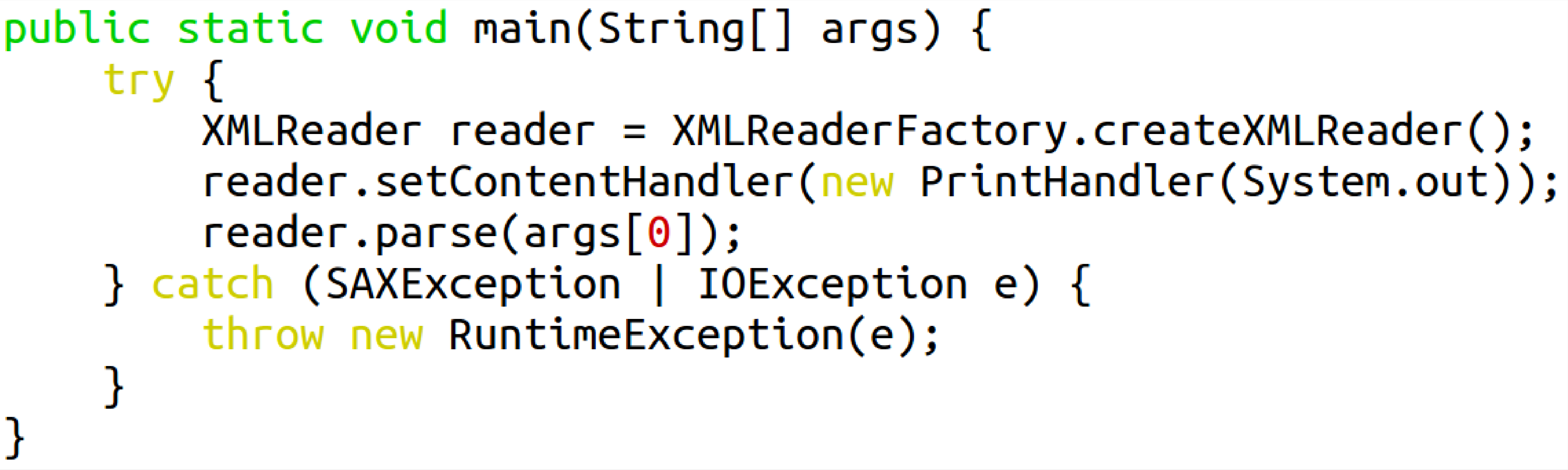
- View the XML document as a stream of events:
  - the document starts
  - a start tag is encountered
  - an end tag is encountered
  - a namespace declaration is seen
  - some whitespace is seen
  - character data is encountered
  - the document ends
- The SAX tool observes these events
- It reacts by calling corresponding methods specified by the programmer

An event handler



```
public class MyHandler extends DefaultHandler {
    public void startElement(String uri, String localName,
            String qName, Attributes atts) {
        ...
    }

    public void characters(char[] ch, int start, int length) {
        ...
    }

    public void endElement(String uri, String localName,
            String qName) {
        ...
    }
}
```

Parsing with a reader



```
public static void main(String[] args) {
    try {
        XMLReader reader = XMLReaderFactory.createXMLReader();
        reader.setContentHandler(new PrintHandler(System.out));
        reader.parse(args[0]);
    } catch (SAXException | IOException e) {
        throw new RuntimeException(e);
    }
}
```

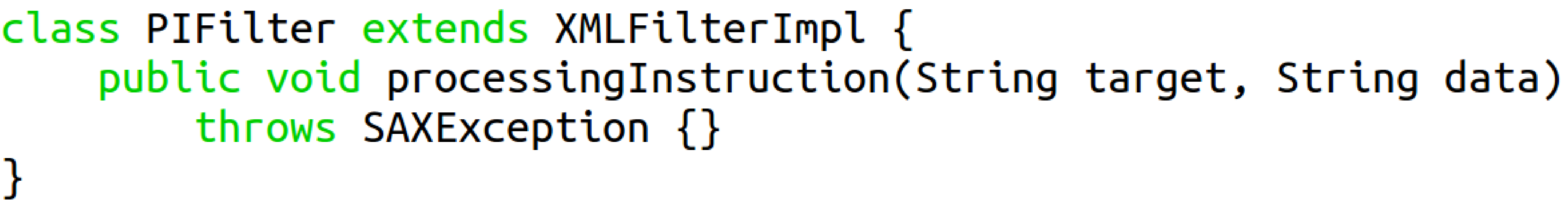
Coding time!

SAX filers

- A SAX application may be turned into a filter
- Filters may be composed (as with pipes)
- A filter is an event handler that may pass events along in the chain
- The next link in the chain is the *parent*
- XMLFilterImpl behavior for all functions: call function in super!

Filter example (1/4)

- A filter to remove processing instructions:

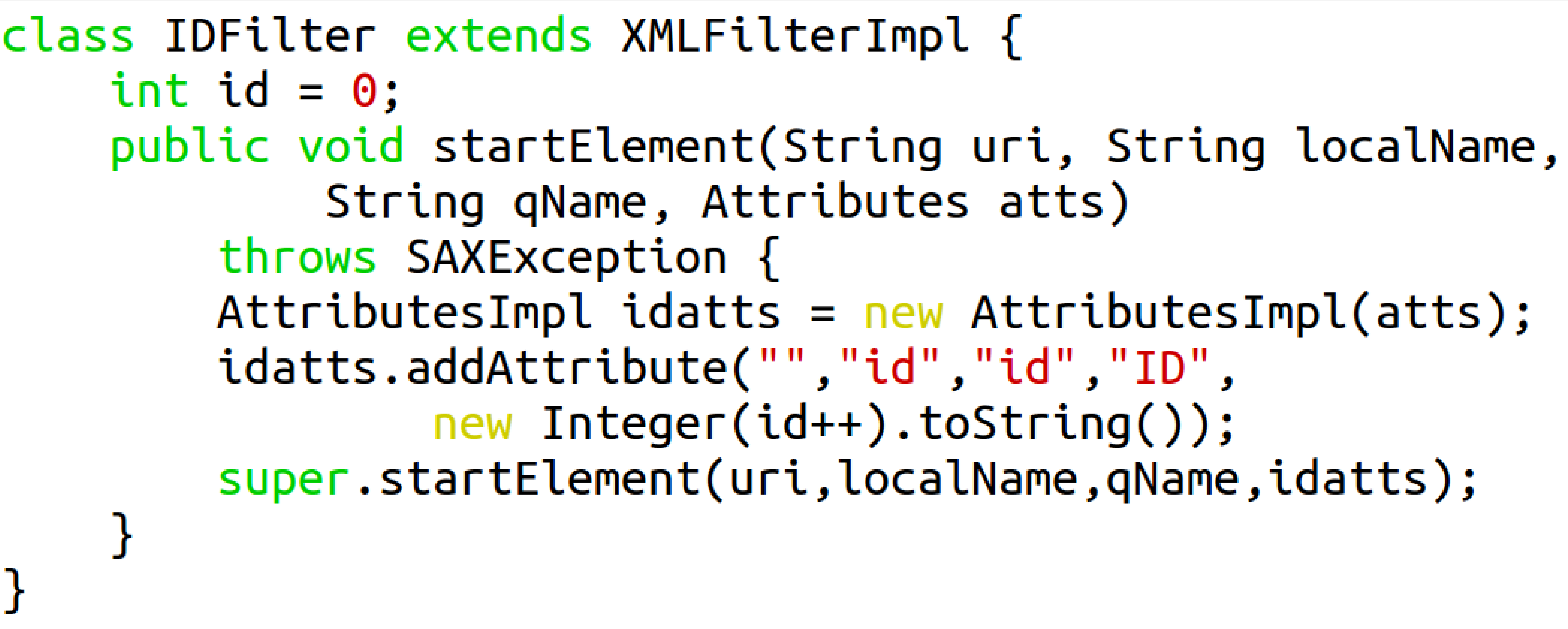


```
class PIFilter extends XMLFilterImpl {
    public void processingInstruction(String target, String data)
        throws SAXException {}
}
```

- Overrides processingInstruction(...) to *not* call super.processingInstruction(...)

Filter example (2/4)

- A filter to create unique id attributes:

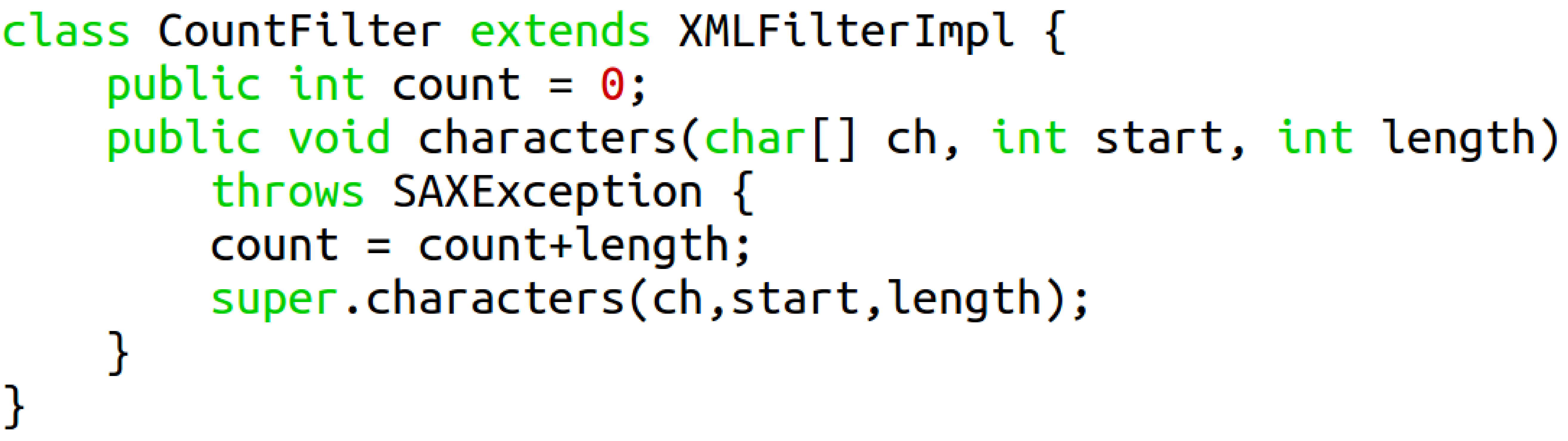


```
class IDFilter extends XMLFilterImpl {
    int id = 0;
    public void startElement(String uri, String localName,
            String qName, Attributes atts)
        throws SAXException {
        AttributesImpl idatts = new AttributesImpl(atts);
        idatts.addAttribute("","id","id","ID",
                new Integer(id++).toString());
        super.startElement(uri,localName,qName,idatts);
    }
}
```

- Calls startElement on super, passing the element along the chain

Filter example (3/4)

- A filter that counts the total number of characters

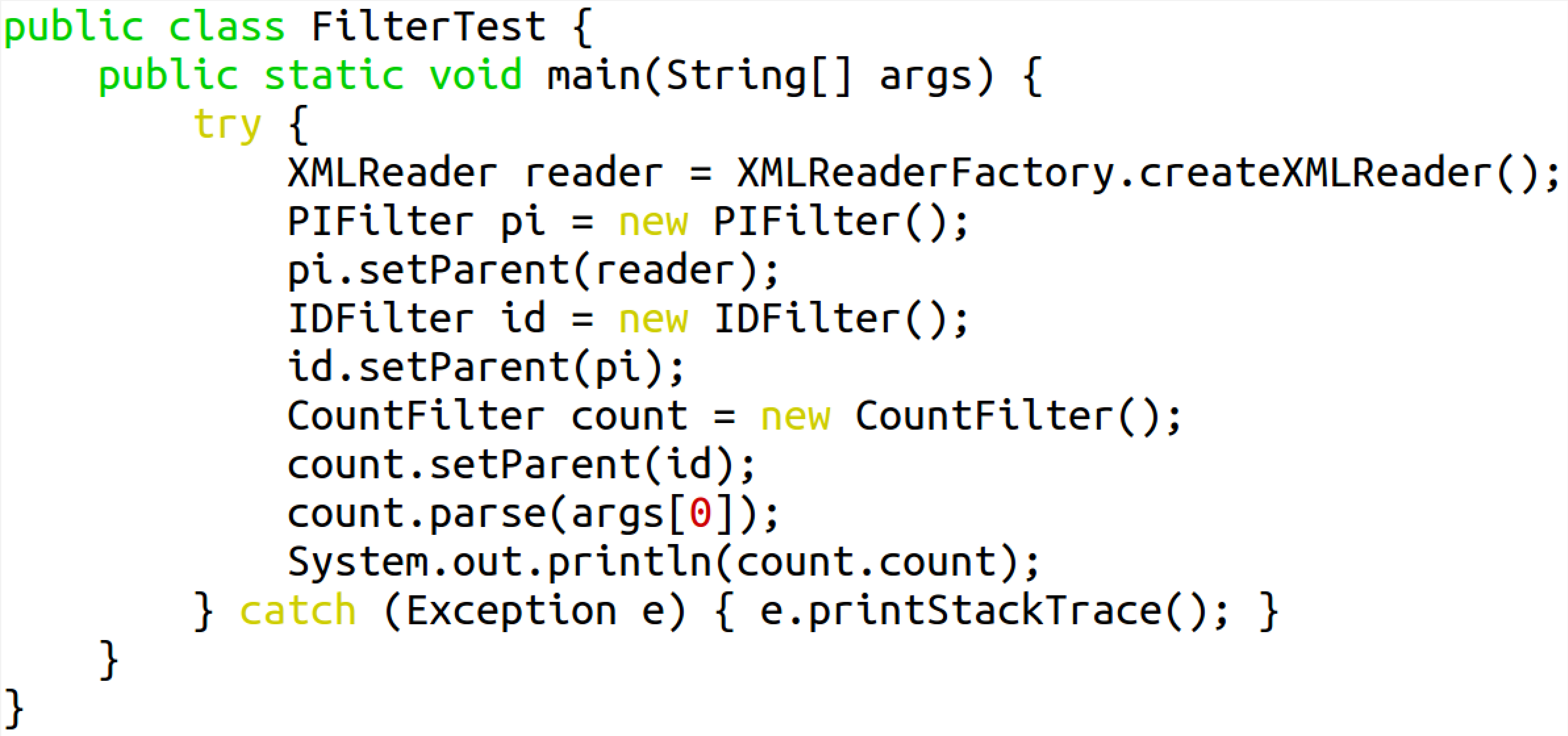


```
class CountFilter extends XMLFilterImpl {
    public int count = 0;
    public void characters(char[] ch, int start, int length)
        throws SAXException {
        count = count+length;
        super.characters(ch,start,length);
    }
}
```

- Does not modify the stream; just observes

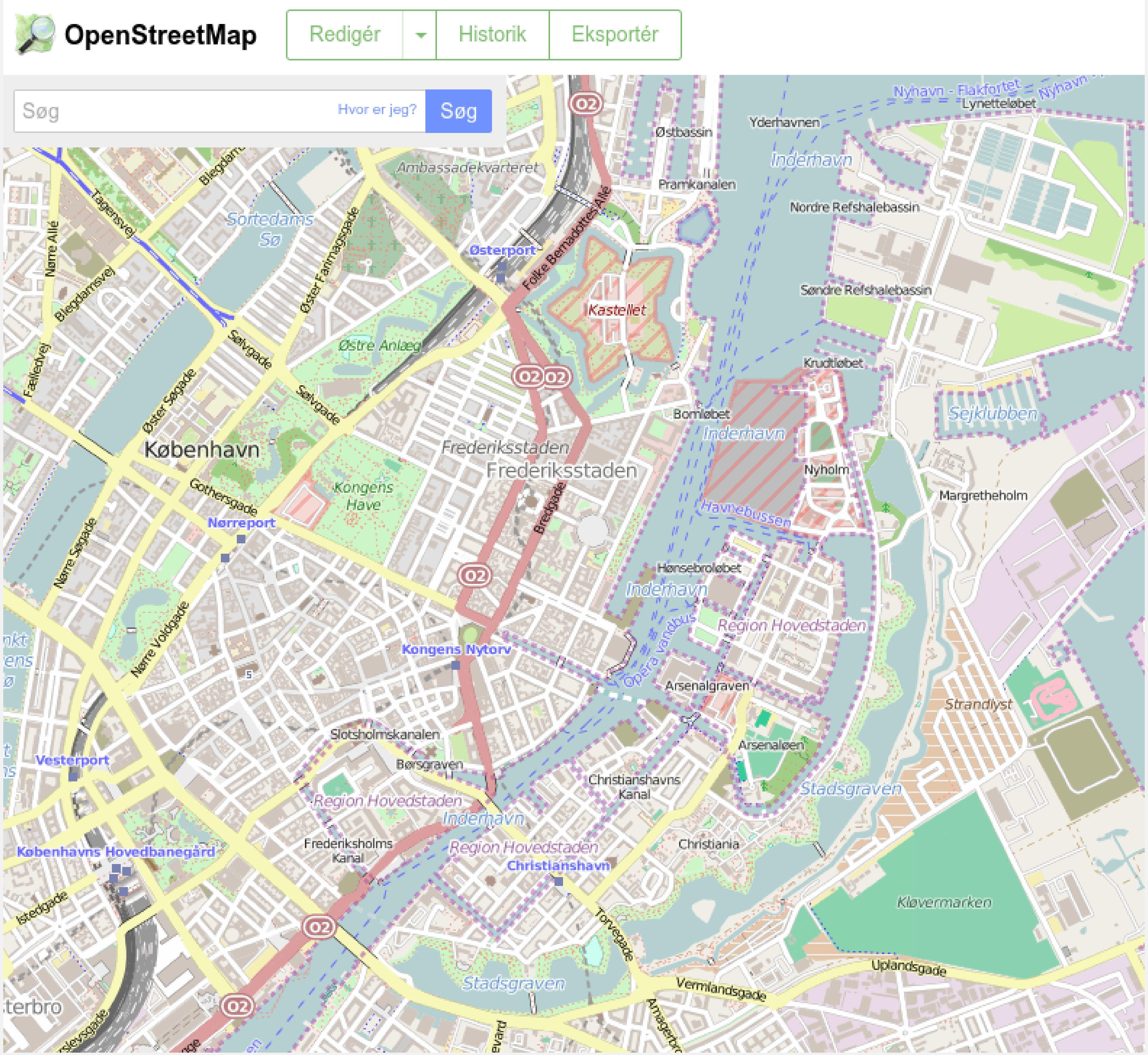
Filter example (4/4)

- Linking the chain together



```
public class FilterTest {
    public static void main(String[] args) {
        try {
            XMLReader reader = XMLReaderFactory.createXMLReader();
            PIFilter pi = new PIFilter();
            pi.setParent(reader);
            IDFilter id = new IDFilter();
            id.setParent(pi);
            CountFilter count = new CountFilter();
            count.setParent(id);
            count.parse(args[0]);
            System.out.println(count.count);
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

Open Street Maps www.openstreetmaps.org



</lecture>