Outline

- XML documents
- XML in Java
  - DOM
  - SAX
  - XML Output
XML

- HTML-like markup
- Semantic meaning
Example HTML

<html>
    <h3>Message</h3>
    <b>To:</b> you@yourAddress.com<br>
    <b>From:</b> me@myAddress.com<br>
    <b>Subject:</b> XML Is Really Cool<p>
    How many ways is XML cool? Let me count the ways...<p>
</html>
Example

<message>
    <to>you@yourAddress.com</to>
    <from>me@myAddress.com</from>
    <subject>XML Is Really Cool</subject>
    <text>
        How many ways is XML cool?
        Let me count the ways...
    </text>
</message>
XML is not

- a language
- It is a standard for creating task-specific formal languages
- In the past
  - every application has its own representation
  - difficult for applications to interoperate
XML is

- human readable
- not platform-specific
- not presentational
- fixed syntax
- validity separate from interpretation
Differences from HTML

- case-sensitive
- end tags not optional
- empty tags have special syntax
- attribute values must be in quotes
- XHTML
Examples

- members
- flights
Vocabulary

- **Header**
  - `<xml version="1.0"/>

- **Elements**
  - member
  - members

- **Attributes**
  - level="standard"

- **Entities**
  - &
A document type is a specification for a family of documents.

- email messages
- member account data files

Defining a document type:

- deciding what can be said
- how it can be said
- attaching meaningful labels
DTD

- Document Type Definition
  - one of many proposals for defining XML languages
  - others
    - XML Schema, Schematron, DSD, Relax

- DTDs
  - based on the original SGML DTDs
  - not written in XML
  - limited in certain respects

- Easiest to learn
Reading a DTD

- `<!ELEMENT` defines an element
- `<!ATTLIST` associates elements and attributes
- `<!ENTITY` defines entities
Element

- Element name
- content model
  - text or other data
  - elements
  - sequences
  - choices
Attribute Lists

- Name of element
- For each attribute
  - name
  - content type
  - values
  - presence
  - default
Examples

- **Element**
  
  ```xml
  <!ELEMENT members (member*)>
  <!ELEMENT member (name, enroll-date, last-flight-date?, miles, benefit-level)>
  ```

- **Attlist**
  
  ```xml
  <!ATTLIST member id CDATA #REQUIRED>
  <!ATTLIST benefit-level
      level (standard | silver | gold | platinum) "standard">
  ```
Examples

- flights.dtd
- members.dtd
Conformity

- If I just have an XML document
  - I have some data
- If I have XML document and DTD
  - I have some idea what the document is supposed to represent
- If I have XML document validated by DTD
  - I know that the document is organized as the DTD says
DTD Limitations

- No way to specify types of things
  - id must be a number
  - origin must be a three-character string
- Default values only for attributes
- Must have a separate DTD parser
  - not XML
- No import or include
- No inheritance
  - OO element declaration
Programming with XML

- Basic problem
  - use an XML document in a Java program

- Two ways
  - Build an internal representation of the whole thing
  - Extract just the parts you want
DOM

- Document Object Model
- Model
  - of the XML document
  - composed of objects
- Think of the document as a tree
DOM Representation
DOM Representation

- Nodes
  - text nodes
  - element nodes
  - whitespace nodes

- List of children
  - ordered top to bottom
Working with the DOM

- Navigate from node to node
  - from Node to ChildList
  - from ChildList to Node
- Easier with XPath
  - /member[0]/name/last
Example

- DOMExtract
- Extract
  - last name
  - miles
  - benefit level
Better with Validation

- If we validate the document
  - we know that the document obeys the structure
  - simpler destructuring
Example

- DOMExtract2
Problems with DOM

- May not want the whole document
  - Too big
  - Only interested in certain parts
  - Memory intensive
  - Slow
SAX

- Simple API for XML
- Watch the parser parse
- Parsing = a stream of events
  - each event calls a method
  - like GUI event handling
SAX Events

- startDocument
- endDocument
- startElement
- endElement
- characters
- errors
Example

- members.xml
How to write

- state machine
  - beginnings and ends of elements change state
- in each state
  - process relevant information
Example

- SAXExtract
XML Output

- Rather awkward
- Use XML transformations
  - but don’t do any transforming
  - provides an output stream
Example

- DOMOutput
  - Reads in the file
  - Then adds 1000 to each miles entry
  - Outputs the file