XML in Programming

Patryk Czarnik

XML and Applications 2013/2014
Lecture 4 – 28.10.2013

XML in programming – what for?

- To access data in XML format
- To use XML as data carrier (storage and transmission)
- To support XML applications (Web, content management)
- To make use of XML-related standards
  - (XML Schema, XInclude, XSLT, XQuery, XLink, ...)
- To develop or make use of XML-based technology
  - XML RPC, Web Services (SOAP, WSDL)
  - REST, AJAX

XML in programming – how?

- Bad way
  - Treat XML as plain text files and write low-level XML support from scratch
- Better approach
  - Use existing libraries and tools
- Even better
  - Use standardised interfaces independent of particular suppliers

XML and Java

- Propaganda
  - Java platform provides device-independent means of program distribution and execution.
  - XML is a platform-independent data carrier.
- Practice
  - Java - one of the most popular programming languages, open and portable.
  - Very good XML support in Java platform.
  - Many technologies use XML.

XML in Java – standards

Both included in Java Standard Edition from 6.0

- Java API for XML Processing (**JAXP** 1.x – JSR-206)
  - many interfaces and few classes,
    “factories” and pluggability layer
  - support for XML parsing and serialisation (DOM, SAX, StAX)
  - support for XInclude, XML Schema, XPath, XSLT
- Java API for XML Binding (**JAXB** 2.x – JSR-222)
  - binding between Java objects and XML documents
  - annotation-driven
  - strict relation with XML Schema

Classification of XML access models

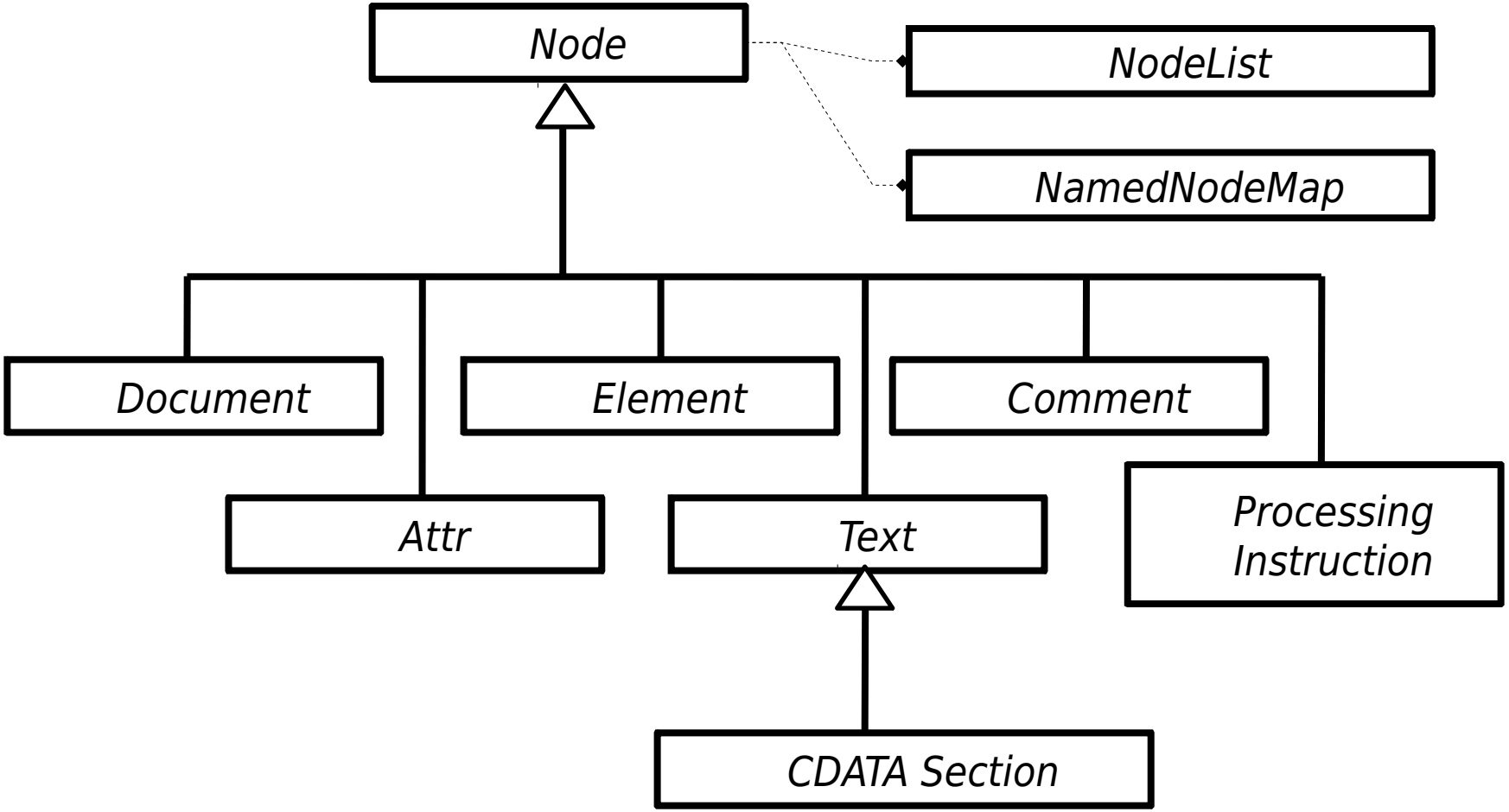
And their standard realisations in Java

- Document read into memory
  - generic interface: **DOM**
  - interface depending on document type/schema: **JAXB**
- Document processed node by node
  - event model (*push parsing*): **SAX**
  - streaming model (*pull parsing*): **StAX**

DOM – status

- W3C Recommendations
  - DOM Level 1 – 1998
  - DOM Level 3 – 2004
  - Several modules. We focus on DOM Core here
- Document model and universal API
  - independent of programming language (IDL)
- Used in various environments
  - notable role in JavaScript model
  - available (in some form) in all modern programming platforms

Primary DOM types



DOM key ideas

- Whole document in memory
- Tree of objects
- Generic interface Node
- Specialised interfaces for particular kinds of nodes
- Available operations
  - reading document into memory
  - creating document from scratch
  - modifying content and structure of documents
  - writing documents

Example: problem introduction

- Count the number of seats in rooms equipped with a projector.

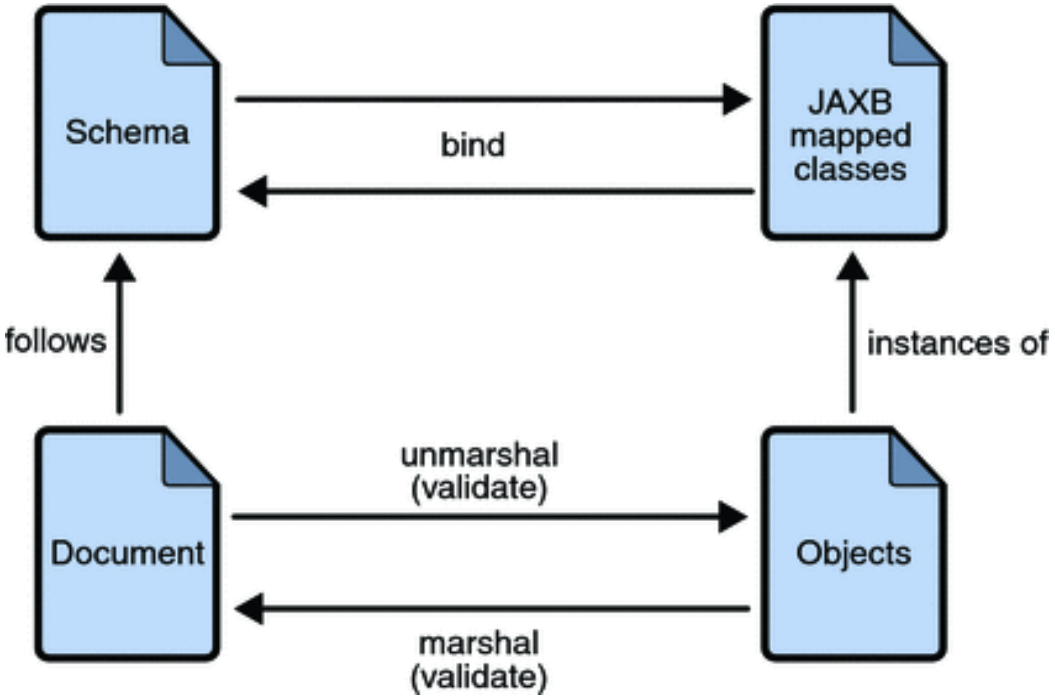
```
<rooms>
    <room>
        <number>2120</number>
        <floor>1</floor>
        <equipment projector="false" computers="false"/>
        <seats>50</seats>
    </room>
    <room>
        <number>3180</number>
        <floor>2</floor>
        <equipment projector="true" computers="false"/>
        <seats>100</seats>
    </room>
    <room>
        <number>3210</number>
        <floor>2</floor>
        <equipment />
        <seats>30</seats>
    </room>
</rooms>
```

Example program

- Two approaches in DOM programming
  - Use only generic Node interface
  - Use specialised interfaces and convenient methods
- See example programs
  - `CountSeats_DOM_Generic`
  - `CountSeats_DOM_Specialised`

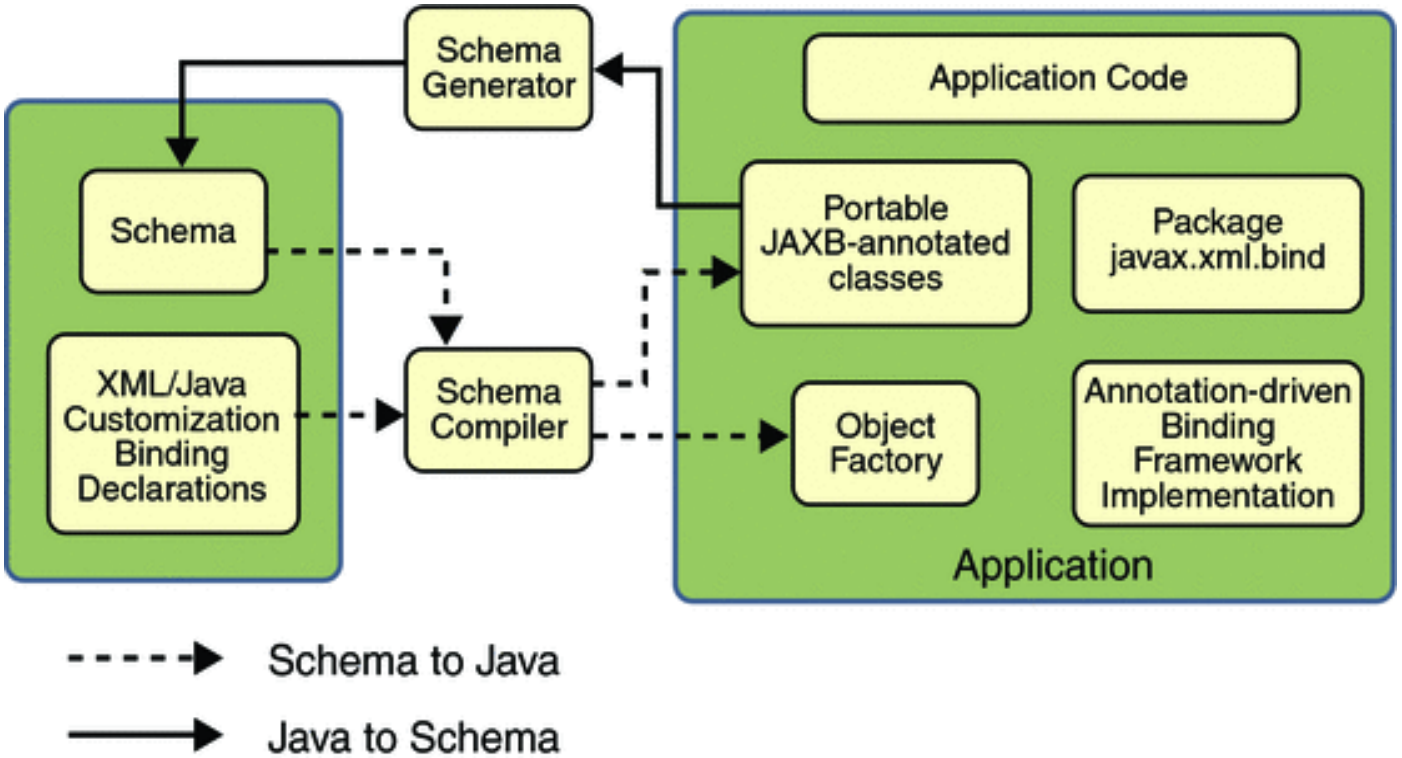
XML binding and JAXB

- Mapping XML to Java
- High-level view on documents
- From programmer's point of view:
  - instead of Integer.parseString(room.getElementsByTagsName("seats").item(0).getTextContent())
  - we simply have room.getSeats()



JAXB 2.x architecture

- Application operates basing on (usually annotated) “JAXB classes”
  - generated from a schema
  - or written manually



Example

- We generate Java classes basing on our schema
  - `xjc -d src -p` *`package_name`* `school.xsd`
- See generated classes and program `CountSeats_JAXB`

JAXB – applications and alternatives

- Main applications:
  - high-level access to XML documents
  - serialisation of application data
  - automatic mapping of method invocations to SOAP messages in JAX-WS
- Many options to customise the mapping using Java or XML annotations
- Some alternatives:
  - Castor
  - Apache XML Beans
  - JiBX

Streaming (and event) processing
Motivation

- Whole document in memory (DOM, JAXB)
  - convenient
  - but expensive
    - memory for document
      (multiplied by an overhead for structure representation)
    - time for building the tree
    - reading always whole document, even if required data present at the beginning
  - sometimes not possible at all
    - more memory required than available
    - want to process document before it ends
- Alternative: Reading document node by node

Event model

- Document seen as a sequence of events
  - “an element is starting”,
  - “a text node appears”, etc.
- Programmer provides code fragments - “event handlers”
- Parser reads a document and
  - controls basic syntax correctness
  - calls programmer's code relevant to actual events
- Separation of responsibility:
  - Parser responsible for physical-level processing
  - Programmer responsible for logical-level processing

SAX

- Simple API for XML – version 1.0 in 1998
- Independent standard designed for and acquired by Java
- Idea applicable for other programming languages

Typical usage:

- Programmer-provided class implementing `ContentHandler`
- Optionally classes implementing `ErrorHandler`, `DTDHandler`, or `EntityResolver`
  - one class may implement all of them
  - `DefaultHandler` – convenient base class to start with

SAX

Typical usage (ctnd):

- Obtain `XMLReader` (or `SAXParser`) from factory
- Create `ContentHandler` instance
- Register handler in reader
- Invoke parse method
  - Parser conducts processing and calls methods of our `ContentHandler`
- Use data collected by `ContentHandler`

SAX events in run

```
<?xml-stylesheet ...?>

<room>

  <equipment projector="true"/>


  <seats>
    60
  </seats>
</room>
```

- startDocument()
- processingInstruction("xml-stylesheet", ...)
- startElement("room")
- startElement("equipment", {projector="true"})
- endElement("equipment")
- startElement("seats")
- characters("60")
- endElement("seats")
- endElement("room")
- endDocument()

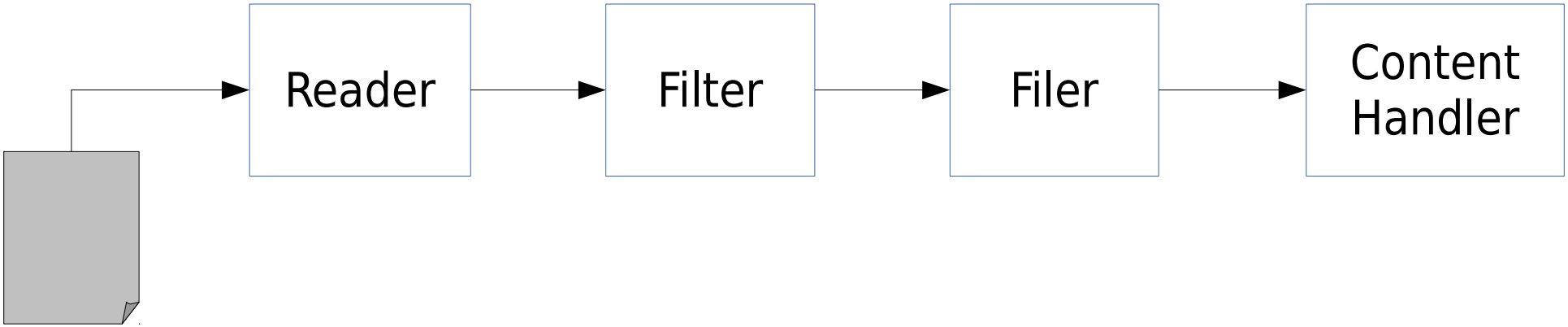
Example

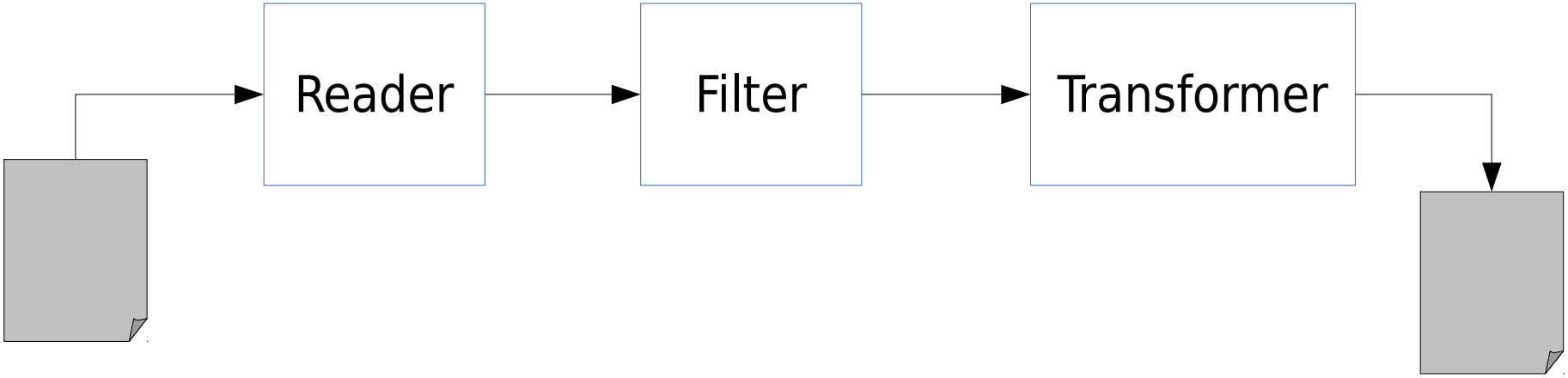
- See example classes:
  - CountSeats_SAX_Traditional and CSHandler_Traditional for traditional scenario of creating parses instance and registering a ContentHandler
  - CountSeats_SAX_JAXP and CSHandler_JAXP for modern JAXP-conformant scenario of combining things together

SAX filters

- Motivation: Joining ContentHandler-like logic into chains
- Realisation:
  - interface `XMLFilter`
    (`XMLReader` having a parent `XMLReader`)
  - in practice filters implements also `ContentHandler`
  - convenient start-point: `XMLFilterImpl`
- Typical implementation of a filter:
  - handle incoming events like in a `ContentHandler`
  - pass events through by manual method calls on the next item in chain
- Filters can:
  - pass or halt an event
  - modify an event or a sequence of events!

Possible usage of SAX filters





SAX – typical problems

- To make implementations portable – we should manually join adjacent text nodes in an element
  - `StringBuffer` is a convenient class
- The same method called for different elements, in different contexts
  - Typical solution – remembering the state (one flag for simple logic or elaborated structures for complex logic)
  - It may become tedious in really complex cases.

StAX: Pull instead of being pushed

- Alternative for event model
  - application “pulls” events/nodes from parser
  - processing controlled by application, not parser
  - idea analogous to: iterator, cursor, etc.
- More intuitive control flow
  - reduced need of remembering the state etc.
- Advantages of SAX saved
  - high efficiency
  - possibility to process large documents

StAX

- Streaming API for XML
- Available in Java SE since version 6

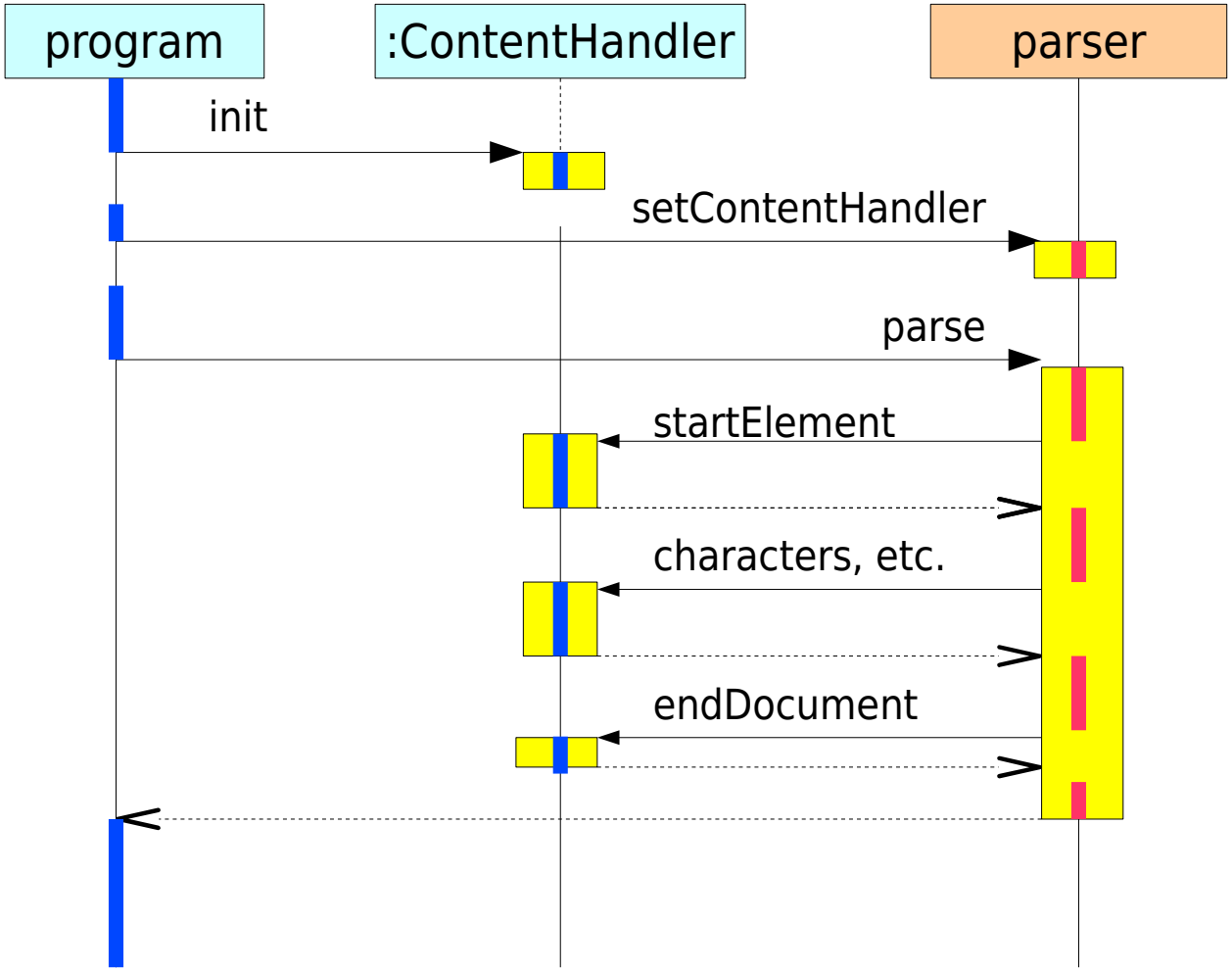
Two levels of abstraction:

- `XMLStreamReader`
  - one object for all purposes
  - most efficient approach
- `XMLEventReader`
  - subsequent events (nodes) provided as separate objects
  - more convenient for high-level programming, especially when programming modification of document “on-the-fly”

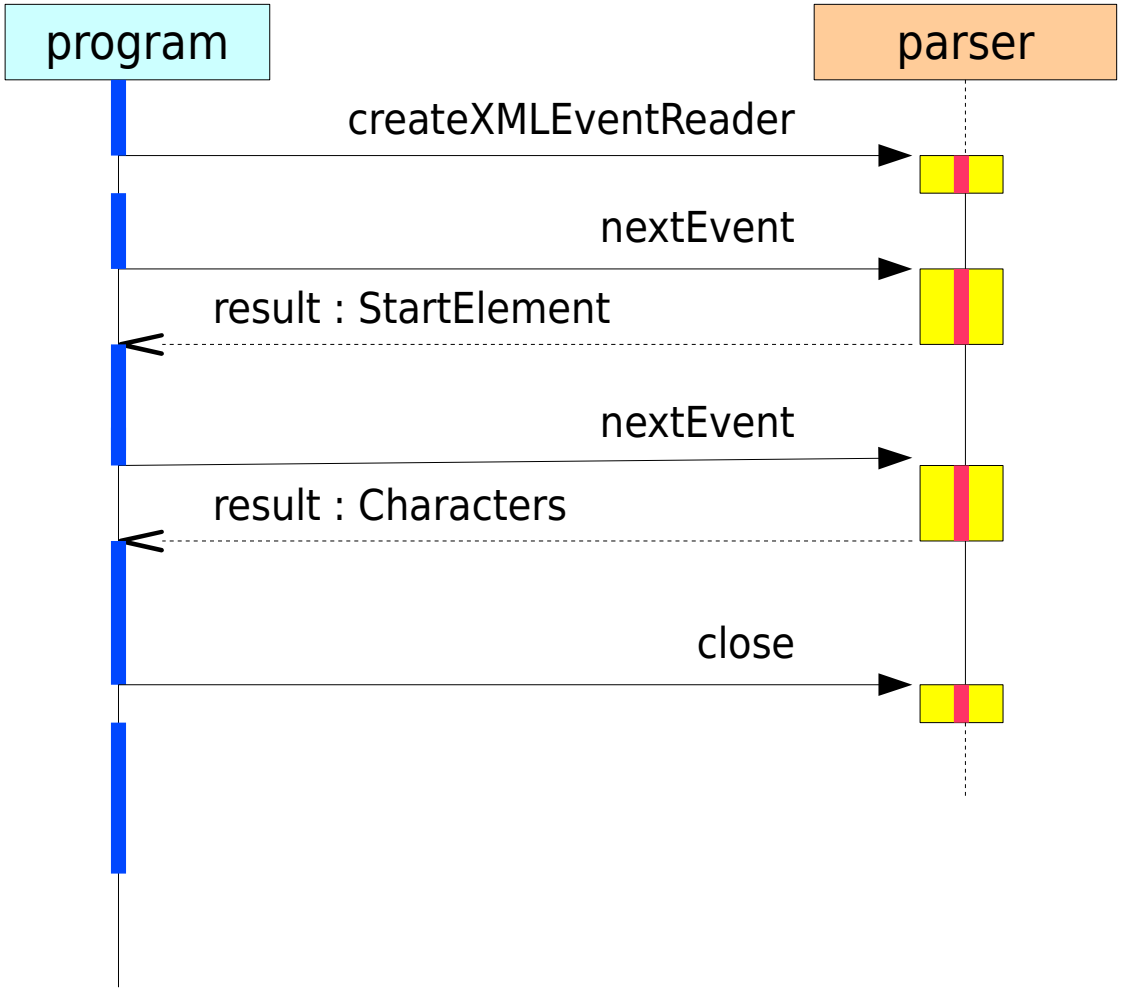
Example

- See programs
  - `CountSeats_Stax_Stream`
    for usage of low-level `XMLStreamReader`
  - `CountSeats_Stax_Event`
    for usage of `XMLEventReader`

Control flow in SAX



Control flow in StAX



StAX features

- API for reading documents: XMLStreamReader, XMLEventReader
- API for writing documents: XMLStreamWriter, XMLEventWriter
- Filters
  - simple definition of a filter: accept(Event): boolean
  - “filtered readers”

Which model to choose? (1)

- Document tree in memory:
  - small documents (must fit in memory)
  - concurrent access to many nodes
  - creating new and editing existing documents “in place”
- Generic document model (like DOM):
  - not established or not known structure of documents
  - lower efficiency accepted
- XML binding (like JAXB):
  - established and known structure of documents
  - XML as data serialisation method

Which model to choose? (2)

- Processing node by node
  - potentially large documents
  - relatively simple, local operations
  - efficiency is the key factor
- Event model (SAX):
  - using already written logic (SAX is more mature)
  - filtering events, asynchronous events
  - several aspects of processing during one reading of document (filters)
- Streaming model (like StAX):
  - processing depending on context; complex states
  - processing should end after data is found
  - reading several documents simultaneously