XML – Part B

Processing and Storing

Part 4 – JAXP (Java API for XML Processing)

JAXP

Java API for XML Processing

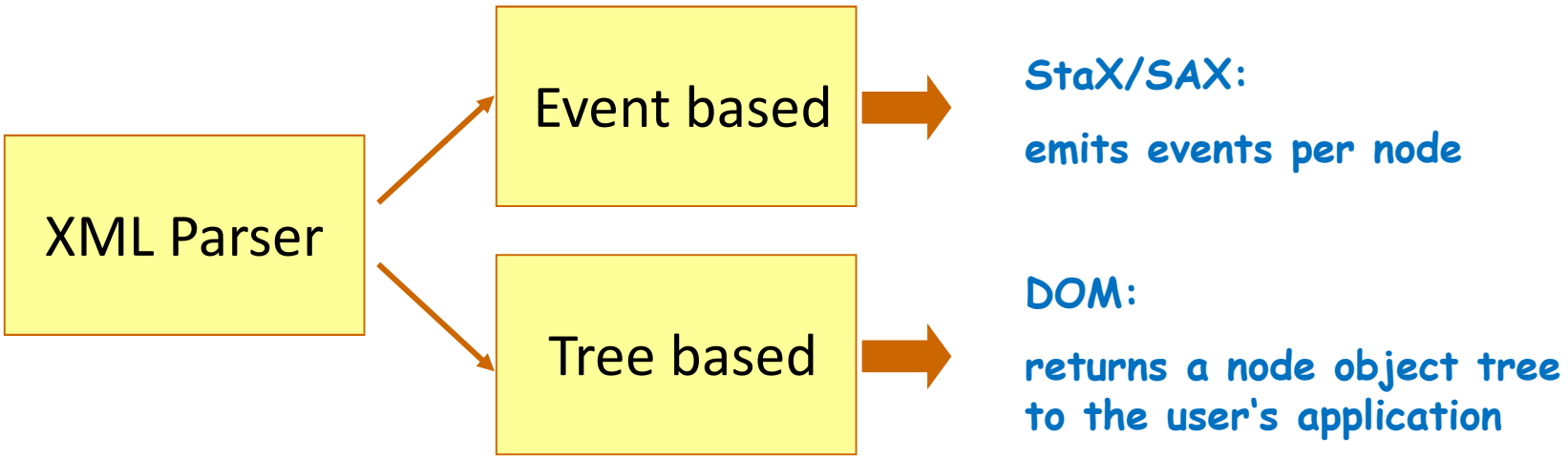
- one of the Java XML programming APIs
- can validate and parse XML documents.

The basic interfaces are:

- the Streaming API for XML or **StAX** interface
- the Simple API for XML parsing interface or **SAX** interface
- the Document Object Model parsing interface or **DOM** interface
- The XSLT interface

XML Parser

There are 2 different implementations of XML parsers:



StAX

Streaming API for XML

StAX

Streaming API for XML, simply called StAX, is an API for reading and writing XML documents.

- Introduced in Java 6.0 and considered as superior to SAX and DOM.
- **Tip:** StAX is cool if you need the control over the XML flow. Otherwise use JAXB.

StAX - Overview

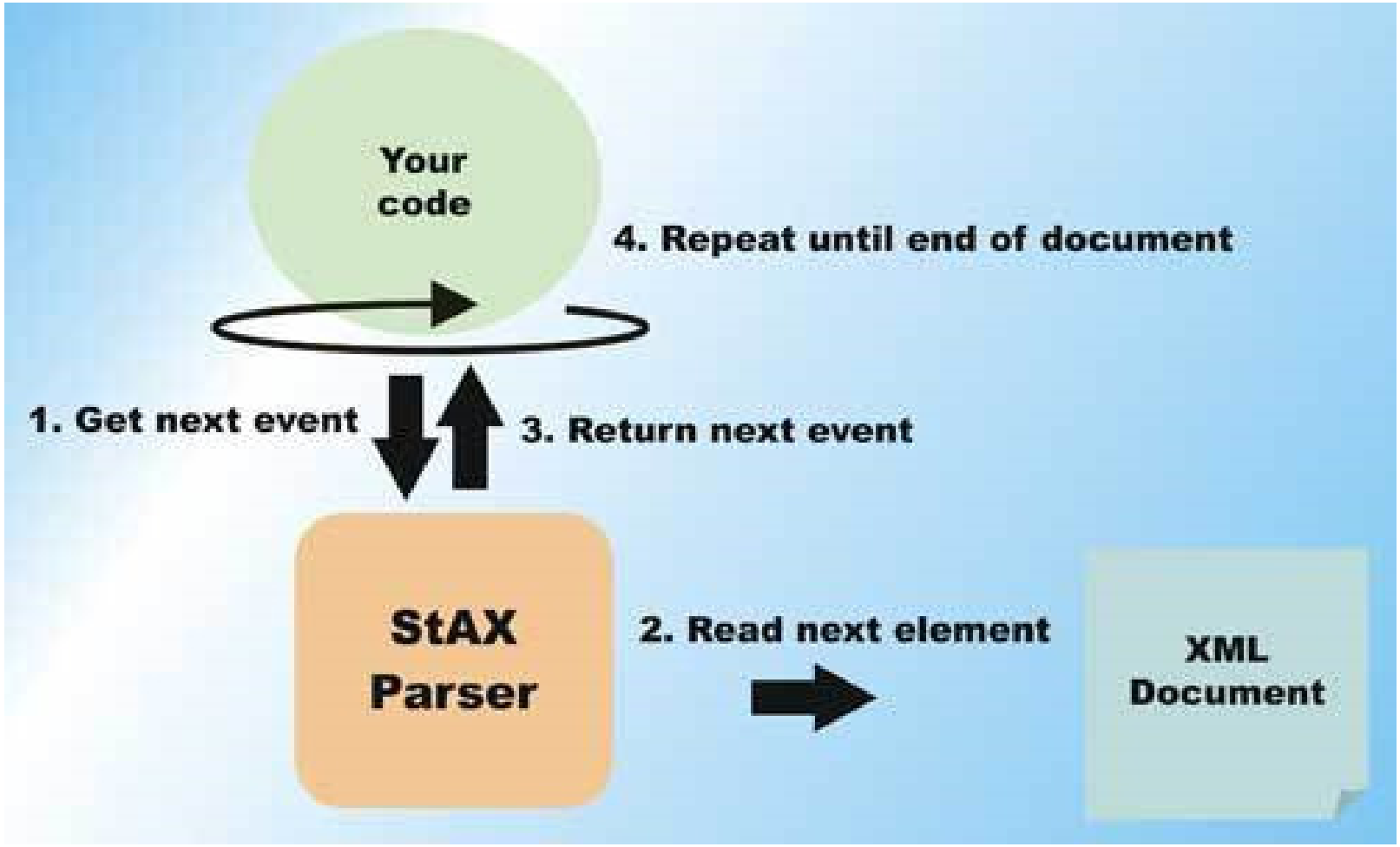
- StAX is a **Pull-Parsing model**.
- Application can take the control over parsing the XML documents by pulling (taking) the events from the parser.
- The core StAX API offers **two categories:**
  - Cursor API
  - Event Iterator API
- Applications can use any of these two API for parsing XML documents.
- The following will focus on the **event iterator API** (it is more convenient to use).

StAX - Event Iterator API

- The event iterator API has **two main interfaces**:
  - `XMLEventReader` for parsing XML
  - `XMLEventWriter` for generating XML

- More information about cursor and iterator API:
  - https://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/SJSXP3.html

StAX - XMLEventReader



StAX Events 1/2

Event Type	Description
`StartDocument`	Reports the beginning of a set of XML events, including encoding, XML version, and standalone properties.
`StartElement`	Reports the start of an element, including any attributes and namespace declarations; also provides access to the prefix, namespace URI, and local name of the start tag.
`EndElement`	Reports the end tag of an element.
`Characters`	Corresponds to XML CData sections and CharacterData entities.
`EntityReference`	Character entities can be reported as discrete events, which an application developer can then choose to resolve or pass through unresolved. By default, entities are resolved..

StAX Events 2/2

Event Type	Description
`ProcessingInstruction`	Reports the target and data for an underlying processing instruction.
`Comment`	Returns the text of a comment
`EndDocument`	Reports the end of a set of XML events.
`DTD`	Reports as `java.lang.String` information about the DTD, if any.
`Attribute`	Attributes are generally reported as part of a `StartElement` event.
`Namespace`	As with attributes, namespaces are usually reported as part of a `StartElement`.

StAX - Sample Event Mapping 1/3

- As an example of how the event iterator API maps an XML stream, consider the following XML document:

```
<?xml version="1.0"?>
<BookCatalogue xmlns="http://www.publishing.org">
  <Book>
    <Title>Yogasana Vijnana: the Science of Yoga</Title>
    <ISBN>81-40-34319-4</ISBN>
    <Cost currency="INR">11.50</Cost>
  </Book>
</BookCatalogue>
```

StAX - Sample Event Mapping 2/3

#	Element/Attribute	Event
1	version="1.0"	`StartDocument`
2	qname = BookCatalogue:http://www.publishing.org attributes = null	`StartElement`
3	isCData = false data = "\n" IsWhiteSpace = true	`Characters`
4	qname = Book attributes = null	`StartElement`
5	isCData = false data = "\n" IsWhiteSpace = true	`Characters`
6	qname = Title attributes = null	`StartElement`
7	isCData = false data = "Yogasana Vijnana: the Science of Yoga\n\t" IsWhiteSpace = false	`Characters`

StAX - Sample Event Mapping 3/3

#	Element/Attribute	Event
8	qname = Title namespaces = null	`EndElement`
…		`EndElement`
15	isCData = false data = "11.50" IsWhiteSpace = false	`Characters`
16	qname = Cost	`EndElement`
17	isCData = false data = "\n" IsWhiteSpace = true	`Characters`
18	qname = Book	`EndElement`
19	isCData = false data = "\n" IsWhiteSpace = true	`Characters`
20	qname = BookCatalogue:http://www.publishing.org	`EndElement`
21		`EndDocument`

StAX – event processing - reading

Create a new XMLInputFactory

Setup a new XMLEventReader

Process the events

Cast the event to get more information

```
XMLInputFactory inputFactory = XMLInputFactory.newInstance();

InputStream in = new FileInputStream("BookCatalogue.xml");
XMLEventReader eventReader = inputFactory.createXMLEventReader(in);

while (eventReader.hasNext()) {
    XMLEvent event = eventReader.nextEvent();
    switch (event.getEventType()) {
        case XMLEvent.START_DOCUMENT: ... break;
        case XMLEvent.START_ELEMENT:
            StartElement s = event.asStartElement();
            ... break;
        ...
        default: ... break;
    }
}
```

StAX - XMLEventReader - Read XML Example

- The application **loops over the entire document** requesting the next *Event*.
- In this example we **read** the following XML document and **create objects** from it.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <item date="January 2015">
    <mode>1</mode>
    <unit>900</unit>
    <current>1</current>
    <interactive>1</interactive>
  </item>
  <item date="February 2015">
    <mode>2</mode>
    <unit>400</unit>
    <current>2</current>
    <interactive>5</interactive>
  </item>
  <item date="December 2015">
    <mode>9</mode>
    <unit>5</unit>
    <current>100</current>
    <interactive>3</interactive>
  </item>
</config>
```

StAX - XMLEventReader - Read XML Example

- You need to define a class to store the individual entries of the XML file -> *Item.java*
  - A simple class with all sub elements as private fields with get and set methods
- You can test the parser with the program -> *ReadTest.java*
- The logic is in class *StAXParser.java*

StAX – event processing – writing (1/2)

Create a new XMLOutputFactory

```
XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
try (OutputStream out = new FileOutputStream("config.xml")) {
    XMLEventWriter eventWriter =
        outputFactory.createXMLEventWriter(out);

    XMLEventFactory eventFactory = XMLEventFactory.newInstance();



    XMLEvent nl = eventFactory.createCharacters("\n");
    StartDocument startDocument =
        eventFactory.createStartDocument();
    StartElement itemElem =
        eventFactory.createStartElement("", "", "item");
```

Setup a new XMLEventWriter

Create an XMLEventFactory

Create events

StAX – event processing – writing (2/2)

```
    eventWriter.add(startDocument);
    eventWriter.add(nl);
     ...
    eventWriter.add(itemElem);
    eventWriter.add(eventFactory.createAttribute("date", "March 2015"));
     ...
    eventWriter.add(eventFactory.createEndElement("", "", "item"));
     ...
    eventWriter.add(nl);
    eventWriter.add(eventFactory.createEndDocument());
} catch (IOException e) {
        e.printStackTrace();
} catch (XMLStreamException e) {
        e.printStackTrace();
}
```

Adds events to the XMLStream

Adds an attribute

Adds events to the XMLStream

Don't forget to handle exceptions

StAX - Write XML File - Example

- See the example classes:
  - *Item.java*
  - *StAXWriter.java*
  - *WriteTest.java*