Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

Az XML és a Java

Sipos Róbert

siposr@hit.bme.hu

2012. 04. 03.

XML (eXtensible Markup Language)

- AZ XML egy szöveg alapú, strukturált jelölő nyelv, amely a szöveget tartalom szerinti struktúrába rendezi.
  - Szabvány (W3C).
  - Nagyon hasonlít a HTML-hez.
- Alapfogalom: tag

```
<?xml version="1.0"?>
<mondat>
    <szo sorszam="1">A</szo>
    <szo sorszam="2">kutya</szo>
    <szo sorszam="3">ugat</szo>
</mondat>
```

XML tag-ek

- Szintaxis

```
<tagNév [attribútum]*>tartalom</tagNév>
<tagNév [attribútum]*/>                  //üres tag
```

- Tageket kötelező lezárni (kivéve a fejlécet <?xml?>)
- Helyes egymásba ágyazottság
- Kötelező gyökérelem
- Tartalom: más tagek, szöveg, entitás
  - a whitespace-eket megőrzi! (feldolgozásnál erre figyelni kell)
  - új sor: mindig LF

XML tag-ek (2)

- Tagnév:
  - Case sensitive
  - Tetszőlegesen hosszú
  - Karakterek: betű, szám, egyéb karakterek (`:_-.`)
  - Nem tartalmazhat space-t
  - Kezdete:
    - az "`xml`" foglalt szó
    - 1. karakter: nem lehet szám vagy központozás
  - A "`:`" foglalt (namespace szabvány)
  - Nem angol karakterek: elvileg igen, gyakorlatilag gyártó-függő
  - Ajánlás: "."-ot ne használjunk

XML attribútumok

- Formátum

```
attribútumNév="érték"
```

- Attribútumnév: ugyanaz, mint a tagneveknél
- Az érték kötelezően idézőjelek között van.
  - Mindegy, hogy " " (idézőjel) vagy ' ' (aposztróf).

```
<szereplő név='Deák "Bill" Gyula'>
<szereplő név="Deák 'Bill' Gyula">
```

- Az attribútumok között csak space, tab vagy újsor lehet

XML egyéb

- Kötelező fejléc

```
<?xml version="1.0" encoding="windows-1252" standalone="yes"?>
```

  - tipikus kódolások: `UTF-8, UTF-16, ISO-8859-1, ISO-8859-2`

- Megjegyzés

```
<!-- comment -->
```

- Feldolgozási paraméterek

```
<?target instructions?>
```

- Speciális jelek

&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	”

Namespaces

- A névütközések elkerülésére való.
- Megadható, hogy milyen névtérben legyenek értelmezve a nevek.
  - default: `<element xmlns="namespace">`
- egyszerű (simple) név ↔ minősített (qualified) név
- Pl.: html4-es névtérben értelmezett dokumentumot készítünk

```
<table xmlns="http://www.w3.org/TR/html4/">
    <tr>
        <td>Aladár</td>
        <td>Béla</td>
    </tr>
</table>
```

- A namespace legyen URI, mutasson egy névtér-specifikációra.

Namespaces (2)

- Többszörös névtér-specifikációk is lehetségesek qualified name-ek megadásához
- Pl.: egy `bk` és egy `isbn` névteret együttesen használunk az alábbi dokumentumban

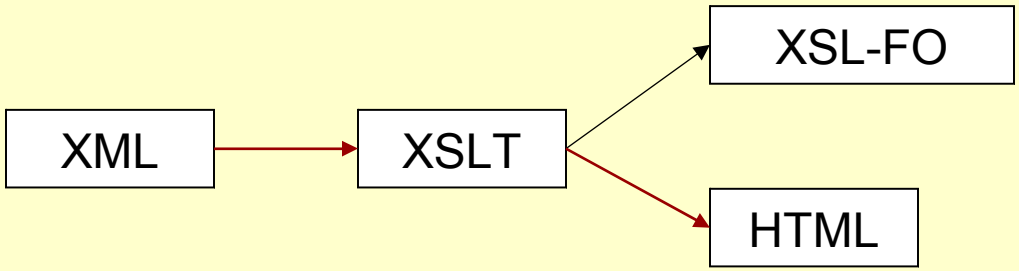
```
<?xml version="1.0"?>

<bk:book xmlns:bk='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <bk:title>Cheaper by the Dozen</bk:title>
    <isbn:number>1568491379</isbn:number>
</bk:book>
```

Jól formázottság és érvényesség

- **Jól formázott (well formed) dokumentum**
  - helyes tag egymásba ágyazottság, van gyökérelem
  - minden tag lezárva
  - szintaktikailag hibátlan
- **Érvényes (valid) dokumentum**
  - jól formázott
  - megfelel a dokumentum definíciónak (DTD vagy Schema)
    → szemantikai megkötések
- DTD és XML Schema: a dokumentum struktúráját írják le
  (pl. a tagek egymásba ágyazhatóságát, tartalmát, attribútumait írja elő)
  - A DTD (Document Type Definition) elavultnak számít,
    ajánlott az XML Schema használata.

XSLT

- XSL: Extensible Stylesheet Language Family
- XSLT: XSL Transformations
- XSL-FO: XSL Formatting Objects
- Lehetőség van az XML dokumentum transzformálására.
- Megjelenítéshez tipikus.



Java API-k az XML-hez

- **JAXP**: Java API for XML Processing
  - XML dokumentum létrehozása, beolvasása, kezelése, transzformálása
- **JAXB**: Java API for XML Binding
  - Java objektumok kiírása (marshalling) és beolvasása (unmarshalling) XML-ként
- **SAAJ**: SOAP with Attachments API for Java
  - alkalmazások közötti szinkron üzenetküldés
- **JAX-RPC, JAX-WS**: Java API for XML-based RPC
  - távoli eljáráshívás XML alapon
- **JAXR**: Java API for XML Registries
  - XML alapú registry létrehozása és használata

JAXP

- Java API for XML Processing
  - XML beolvasása, létrehozása, kezelése parserekkel
    - SAX (Simple API for XML) → eseménysorozat
    - DOM (Domcument Object Model) → fastruktúra
  - XML transzformálása
    - XSLT szerint
- package-ek
  - **javax.xml.parsers** – interface a parserekhez
  - **org.w3c.dom** – DOM API
  - **org.xml.sax** – SAX API
  - **javax.xml.transform** – XSLT API

JAXP

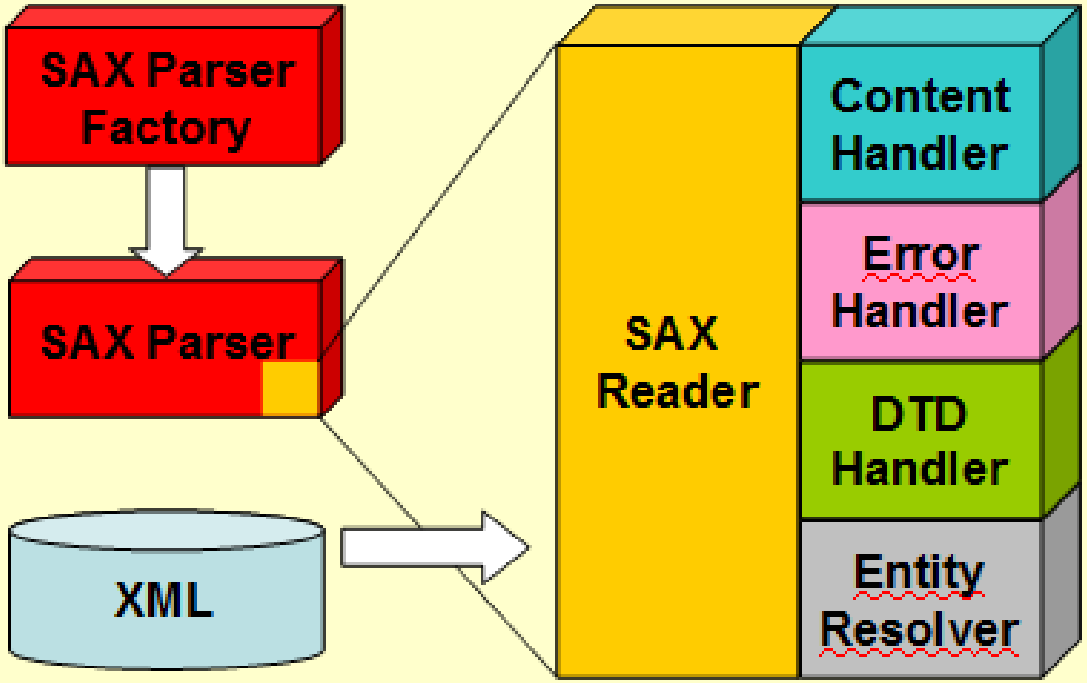
- Java API for XML Processing
  - XML beolvasása, létrehozása, kezelése parserekkel
    - DOM (Domcument Object Model)
      - fastruktúra
    - SAX (Simple API for XML)
      - Eseménysorozat
    - StAX→ fastruktúra
      - Eseménysorozat, de pull jelelgű

DOM vagy SAX

	DOM	SAX
Általában	az egész dokumentumra szükség van	mindig csak a dokumentum egy részére van szükség
Cél	dokumentum manipulálása	„adat filter”
Kapacitás	a memória nem korlát	nincs sok memória

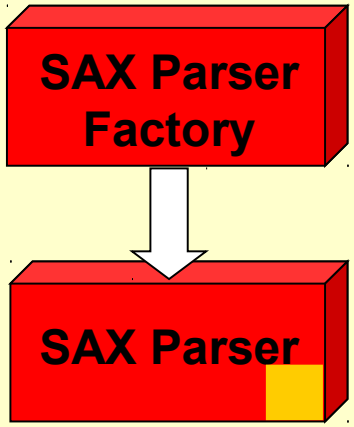
SAX API

- A dokumentumot eseménysorozattá alakítja.
- Ilyen események pl: tag kezdete, tag vége stb.

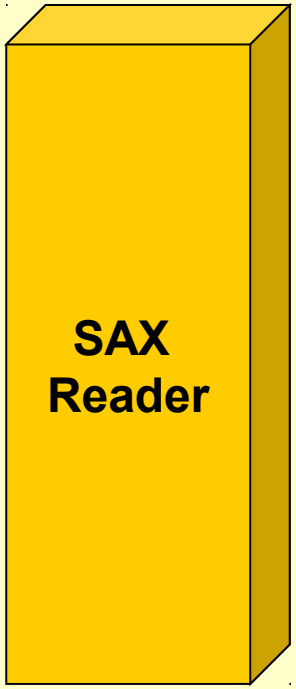


SAX API (2)

- SaxParserFactory
  - `javax.xml.parsers.SAXParserFactory`
  - Létrehozza a SaxParser egy példányát
  - Parser tulajdonságai állíthatók (pl. validating)
- SaxParser
  - `javax.xml.parsers.SAXParser`
  - Egy dokumentumot elemez
  - Többféle `parse()` metódusa van
  - A parse kötelező paraméterei:
    - dokumentum forrása (pl. a file)
    - eseménykezelő



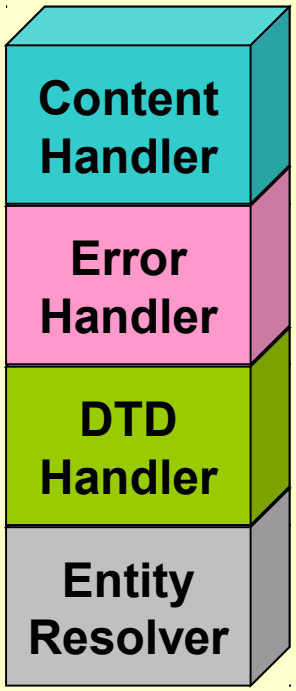
SAX API (3)



- SAXReader (XMLReader)
  - a dokumentum forrása
  - egy interface, amelyet a SaxParser megvalósít
  - tulajdonképp az XmlReader-nek kell ContentHandler, ErrorHandler, DTDHandler és EntityResolver
  - megvalósítása: `org.xml.sax.InputSource` osztály
- `DefaultHandler` (`org.xml.sax.helpers`)
  - (nincs rajta az ábrán)
  - egy osztály, amely megvalósítja a handler-eket és az `EntityResolver`-t, üres metódustörzzsel
  - tipikus használat: a `SaxParser`-nek egy `DefaultHandler`-t adunk meg, és az üres metódustörzseket felüldefiniáljuk

SAX API (4)

- `ContentHandler` interface
  - eseménykezelő metódusok: pl. `startDocument`, `startElement`
- `ErrorHandler` interface
  - hibakezelő metódusok
- `DTDHandler` interface
  - akkor szükséges, ha egy nem elemzett entitáshoz szeretnénk műveletet rendelni
  - szinte sose használjuk
- `EntityResolver` interface
  - külső entitások speciális kezeléséhez van rá szükség
    - pl. a hivatkozott uri-kat a cache-ből szeretnénk előszedni
  - szinte sose használjuk



SAX API (5)

- Tipikus megvalósítás:
  - `SaxParserFactory` létrehozása
  - `SaxParser` elkészítése
    - `InputSource` megadása
    - `DefaultHandler` megadása
      - a `ContentHandler` főbb metódusainak felüldefiniálása

SAX parser

- Készítsünk egy SAX parsert, amely az `a.xml` file teljes tartalmát kiírja a képernyőre.

- Struktúra:
  - Készítsünk egy Echo osztályt (DefaultHandler)
  - Hozzunk létre egy parsert, és adjuk át neki az Echo példányt, mint handler-t
  - Az Echo osztályban majd a megfelelő kezelő metódusokat töltsük ki
    - Dokumentum eleje, vége
    - Tag eleje, vége
    - Szöveg

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
import org.xml.sax.helpers.DefaultHandler;
public class Echo extends DefaultHandler {

    public Echo() {

       SAXParserFactory factory = SAXParserFactory.newInstance();
          //factory.setValidating(true);
          SAXParser saxParser = factory.newSAXParser();
          saxParser.parse( new File("a.xml"), this );
    }

    public static void main(String args[]){
      new Echo();
    }
}
```

SAX parser (3)

- Kész van:
  - factory
  - parser
  - defaultHandler
- Még kell: a handler eseménykezelői
- Fő eseménykezelők:
  - `startDocument`
  - `endDocument`
  - `startElement`
  - `endElement`
  - `characters`

SAX parser (4)

- Kezelőmetódusok:

```
public void startDocument() throws SAXException {
        System.out.println(“---\n<?xml version='1.0'?>“);
}
```

```
public void endDocument() throws SAXException {
        try {
            System.out.println(“---”);
        } catch (IOException e) {
            throw(new SAXException("I/O error", e));
        }
}
```

SAX parser (5)

```
public void startElement(
   String namespaceURI,
   String sName,// simple name (localName)
   String qName, // qualified name
   Attributes attrs )
throws SAXException {

    System.out.print("<"+qName);
    if (attrs != null) {
         for (int i = 0; i < attrs.getLength(); i++){
         System.out.print(attrs.getQName(i)+
             "=\""+attrs.getValue(i)+"\"");
         }
    }
  System.out.println(">");
}
```

SAX parser (6)

```
public void endElement(
String namespaceURI,
String sName, // simple name
String qName  // qualified name
) throws SAXException {
  System.out.println("</"+qName+">");
}
```

```
public void characters(
char buf[],
int offset,
int len ) throws SAXException {
  String s = new String(buf, offset, len);
  System.out.print(s);
}
```

Valódi SAX parser

- Írjuk ki csak a `<nev></nev>` tagek közti szöveget a képernyőre!

```
boolean flag = false;

public void startElement(...) {
  if ("nev".equalsIgnoreCase(qName)) flag = true;
}

public void endElement(...) {
  if ("nev".equalsIgnoreCase(qName)) flag = false;
}

public void characters(...) {
  ...
  if (flag) System.out.println(s);
}
```

SAXParserFactory

- `javax.xml.parsers`
- public **abstract** class SAXParserFactory
- példány kérése:

```
public static SaxParserFactory newInstance()
```

```
SAXParserFactory factory = SAXParserFactory.newInstance();
```

- nem garantált, hogy thread safe
  - célszerű az alkalmazás minden szálánál külön Factory példányt használni
- legfontosabb metódusok

```
isValidating() / setValidating(boolean)
isNameSpaceAware() / setNameSpaceAware(boolean)
newSAXParser()
```

SAXParser

- `javax.xml.parsers`
- legfontosabb metódusok:

```
parse(File, DefaultHandler)
parse(InputStream, DefaultHandler)
parse(String uri, DefaultHandler)
parse(InputSource, DefaultHandler)
```

- illetve

```
isValidating() / setValidating(boolean)
isNameSpaceAware() / setNameSpaceAware(boolean)
setProperty(name, value) / getProperty(name)
```

InputSource

- Az XML file forrása (ha az nem file, akkor stream vagy uri).
- Egy Reader-t csomagol be.
- Legfontosabb konstruktorok:

```
InputSource(Reader)
InputSource(String)
```

- A SAXParser nem módosíthatja a bemenetet. Ha mégis szükséges, akkor egy másolatot módosít.
- Példa a használatra:a bemeneti XML struktúra egy memóriabeli stringben van

```
saxParser.parse(new InputSource(szoveg), handler);
```

DefaultHander

- `javax.xml.helpers`
- **`startDocument`**`()`
  - Dokumentum kezdete.
- **`endDocument`**`()`
  - Dokumentum vége.
- **`startElement`**`(String namespaceURI, String localName, String qName, Attributes atts)`
  - Tag kezdete.
- **`endElement`**`(String namespaceURI, String localName, String qName)`
  - Tag vége.

DefaultHandler (2)

- **`characters`**`(char[] ch, int start, int length)`
  - Tagek közti karakteres adat.
- **`ignorableWhitespace`**`(char[] ch, int start, int length)`
  - White space karakterek.
- **`processingInstruction`**`(String target, String data)`
  - A `<?target data?>` tagben szereplő elemek.
  - Nem tudjuk, hogy a gyökérelem előtt vagy után szerepel.
- **`setDocumentLocator`**`(Locator locator)`
  - Locator beállítása. A Locator nyilvántartja a sorszámot, oszlopszámot.
  - Használata pl.: a parser által nem jelzett szemantikai jellegű hibák helyének meghatározása

Legfontosabb Exception-ok

- `SAXException`
  - Általános hiba
  - A programozó által dobott kivételeket is ebbe kell csomagolni
- `SAXParseException`
  - Megadja a hiba helyét az eredeti XML dokumentumban.
- `SAXNotSupportedException`
  - A szintaxis jó, de a parser nem támogatja a funkciót

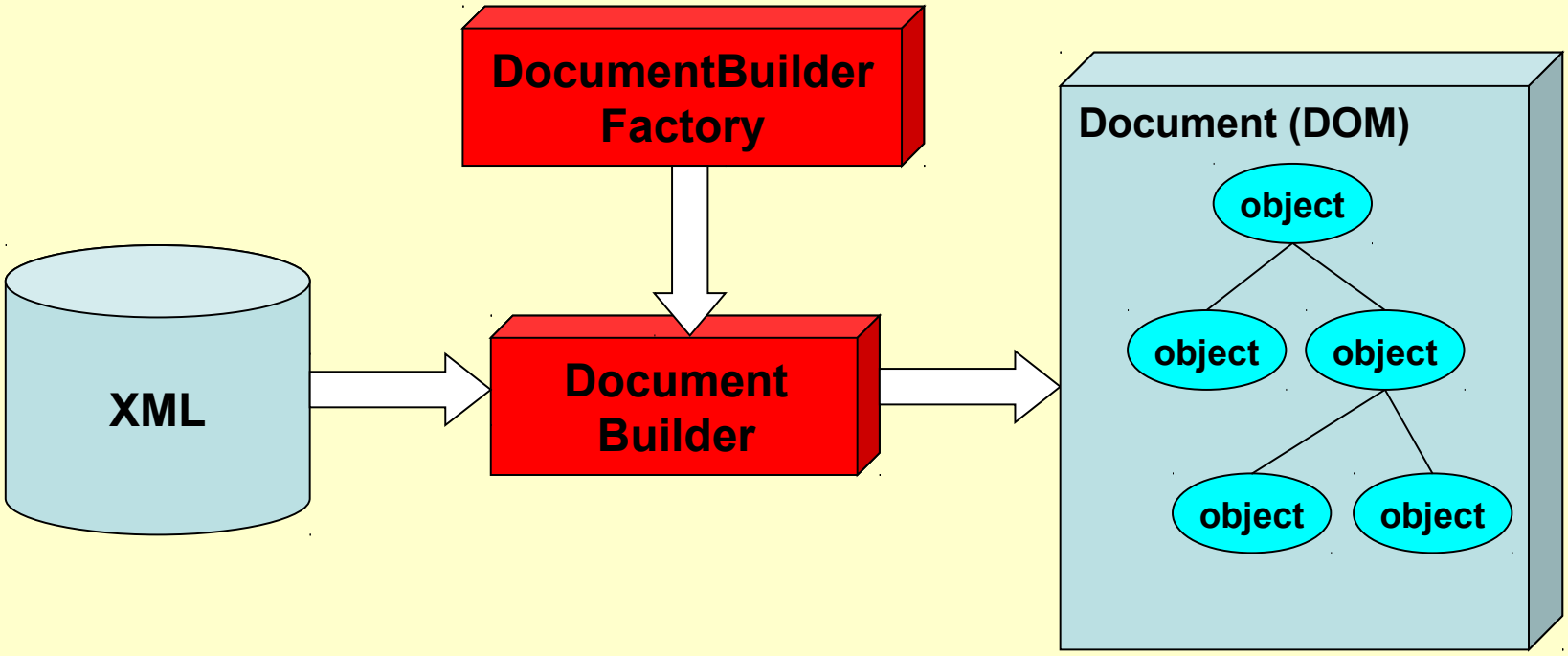
Locator példa

- A `Locator` használata a hibakezelésben

```
public class MyParser extends DefaultHandler {
   private Locator docLocator;

    public void setDocumentLocator(Locator l){
        this.documentLocator = l;
    }

    public void parse(){
     . . .
        } catch (SAXParseException spe){
            System.err.println("Hiba a következő helyen (SOR/OSZLOP):
" + docLocator.getLineNumber() + "/" + docLocator.getColumnNumber());
        }
    }
}
```

DOM API

- A dokumentumot egy memóriabeli fa struktúrába olvassa.



DOM API (2)

- `DocumentBuilderFactory`
  - Példányosítja a a `DocumentBuilder`-t.
- `DocumentBuilder`
  - Elemzi a dokumentumot

- Kimenet: fa struktúra
  - `interface Document`

- Valójában egy SAX parserrel beolvassa a dokumentumot és egy fát épít belőle.

DOM parser

- Feladat: olvassuk be az „a.xml”-t egy DOM parserrel!

```
//A parser létrehozásához szükséges osztályok:
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

//Az elemzés hibakezeléséhez:
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.DOMException;

//A filekezeléshez:
import java.io.File;
import java.io.IOException;
```

DOM parser (2)

```
Document document;

  try {
    DocumentBuilderFactory factory=
                  DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    document = builder.parse( new File("a.xml") );
  } catch (...Exception e) {
    ...
  }
```

DocumentBuilderFactory

- Legfontosabb metódusok:
  - `newInstance()` → új factory példány
  - `newDocumentBuilder()` → új builder példány
  - `setValidating(boolean)`
  - `setNameSpaceAware()`
  - `is / setIgnoringElementContentWhitespace(boolean)`
  - `is / setIgnoringComments(boolean)`
  - `is / setCoalescing(boolean)` → a nem text elemeket (CDATA) textté alakítsa-e

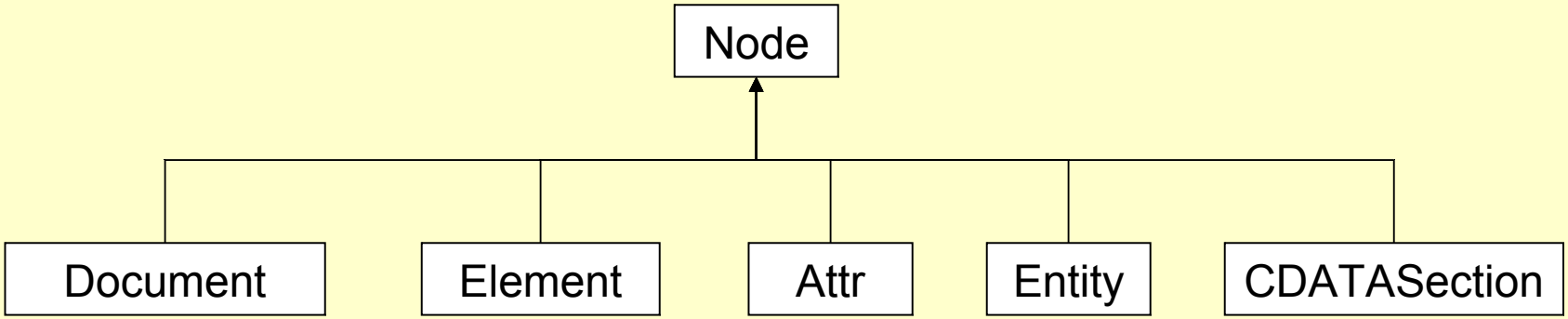
DocumentBuilder

- Legfontosabb metódusai:
  - `newDocument()` → új `Document` létrehozása
  - `parse(File)`
  - `parse(InputSource)`
  - `parse(InputStream)`
  - `parse(String uri)`
  - `setErrorHandler(ErrorHandler eh)`

- Tulajdonképpen kívülről minden ugyanolyan, mint a SAX parsernél.
- Ami igazán fontos: a létrejött fa struktúra kezelése

Interface Document

- Interface, azaz nem lehet példányosítani.
- Létrehozása: `builder.newDocument()`



Interface Document

- Legfontosabb metódusai:
  - `createAttribute(String name)`
  - `createComment(String comment)`
  - `createElement(String tagName)`
  - `createTextNode(String text)`

  - `Element getDocumentElement()` → gyökérelem elkérése
  - `DocumentType getDoctype()` → DTD elkérése
  - `NodeList getElementsByTagName()` → az elemek listája

Interface Node

- Tartalmi információk
  - `NodeList getChildNodes()` → leszármazottak lekérése
  - `NamedNodeMap getAttributes()` → attribútumlista lekérése
  - `String getNodeName()` → tag neve
  - `String getNodeValue()` → tag tartalma
  - `boolean hasAttributes()` → van-e attribútum
  - `boolean hasChildNodes()` → van-e leszármazott
- Leszármazott elemekkel kapcsolatos műveletek
  - `Node appendChild(Node)` → új leszármazott befűzése
  - `Node removeChild(Node)` → leszármazott törlése
  - `Node replaceChild(Node, Node)` → leszármazott cseréje

Node típusok tulajdonságai

Interface	nodeName	nodeValue	attributes
Attr	same as Attr.name	same as Attr.value	null
CDATASection	"#cdata-section"	the content of the CDATA Section	null
Comment	"#comment"	the content of the comment	null
Document	"#document"	null	null
DocumentFragment	"#document-fragment"	null	null
DocumentType	same as DocumentType.name	null	null
Element	same as Element.tagName	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	same asProcessingInstruction.target	same as ProcessingInstruction.data	null
Text	"#text"	the content of the text node	null

NodeList, NamedNodeMap

- `NodeList`
  - `Node`-ok listája
  - Index szerint hozzáférhető
  - A `Node` `getChildNodes` metódusa használja
- `NamedNodeMap`
  - Szintén `Node`-okat tartalmaz
  - Név vagy index szerint hozzáférhető
  - A `Node` `getAttributes` metódusa ezt használja
    - Kényelmesebb, mert az attribútumokhoz név szerint is hozzáférhetünk

Document létrehozása

- Dokumentum létrehozása DOM-mal

```
document = builder.newDocument();
Element root = document.createElement("root_element");
document.appendChild(root);
Text textNode = document.createTextNode("szoveg");
root.appendChild(textNode);
textNode = document.createTextNode(" es meg egy szoveg");
root.appendChild(textNode);
root.normalize();
```

Document módosítása

- A beolvasott dokumentum végére fűzzünk be egy `<modositva>` taget, melyben attribútumként szerepel, hogy ki és mit módosított legutoljára.

```
DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
factory.setValidating(true);
DocumentBuilder builder = factory.newDocumentBuilder();
document = builder.parse( new File(argv[0]) );

Element modositva = document.createElement("modositva");
modositva.setAttribute("szemely", "Bela");
modositva.setAttribute("mitcsinalt", "semmit");

Element root = document.getDocumentElement();
root.appendChild(modositva);
```

Attribútum kinyerése

- Dolgozzunk fel egy XML dokumentumot, melyben többek között lakcímek adatai is találhatók `<lakcim ... irsz="1117" .../>` formában, mi az irányítószámra vagyunk kíváncsiak

```
Document document = null;
try {
   DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
   DocumentBuilder builder = factory.newDocumentBuilder();
   document = builder.parse( "C:\\myXml.xml" );
} catch (Exception e) {
   e.printStackTrace();
}
```

Attribútum kinyerése (2)

```
NodeList lakcimek = document.getElementsByTagName("lakcim");

if ( lakcimek == null ) return;

for(int i = 0; i < lakcimek.getLength(); i++){
 Node currentCim = lakcimek.item(i);
 NamedNodeMap attributes = currentCim.getAttributes();
 String irsz = attributes.getAttribute("irsz");
 if ( irsz != null )System.out.println(irsz);
}
```

Címketartalom kinyerése

- Az előző eset módosítása: most egy `<lakcim>1117 Budapest, Magyar tudósok körútja 2.</lakcim>` alatt található lakcímre van szükség
- Fontos: a szöveg egy külön Node a `<lakcim>` alatt

```
NodeList lakcimek = document.getElementsByTagName("lakcim");

if ( lakcimek == null ) return;

for(int i = 0; i < lakcimek.getLength(); i++){
   Node currentCim = lakcimek.item(i);
   NodeList lakcimChildren = currentCim.getChildNodes();
   if ( lakcimChildren != null){
      Node content = lakcimChildren.item(0);
      if ("#text".equalsIgnoreCase(content.getNodeName()))
         System.out.println( content.getNodeValue() );
      else System.out.println( "A cím nincs megadva");
}
```

Fa struktúra kezelése

- Általános menet:
  - befűzés
    - Document létrehozása
    - `Document.create***` (`***` = `Element`, `Comment`, `Attribute`, ...)
    - új elem befűzése node.appendChild/replaceChild-dal
  - törlés
    - `node.removeChild`
- Exception-ök:
  - DOMException
- Idegen dokumentum által létrehozott Node-okat importálni kell (`document.importNode(foreignNode)`), mielőtt befűznénk

XSLT

- Bemenet: SAX vagy DOM + transzformációs szabályok
- Kimenet: XML dokumentum
- Menete:
  - `TransformerFactory` létrehozása
  - `Transformer` létrehozása a stylesheet megadásával
  - `tranformer.transform(Source xmlSource, Result outputTarget)`
- Transzformációs szabályok: XSLT stylesheet (szintén XML)
- Fontos: namespace-eket használ, ezért csak namespace-aware parserrel elemzett dokumentumot képes hibátlanul transzformálni!

XSLT

```
<?xml version="1.0"?>
<ARTICLE>
<TITLE>Mintacikk</TITLE>
<SECT>...</SECT>
</ARTICLE>
```

- Stylesheet

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet
xmlns:xsl=http://www.w3.org/1999/XSL/Transform
version="1.0">
   <xsl:template match="/">
        <html> <body>
        <xsl:apply-templates />  </body> </html>
   </xsl:template>
   <xsl:template match="/ARTICLE/TITLE">
        <h1 align="center"> <xsl:apply-templates />  </h1>
   </xsl:template>
</xsl:stylesheet>
```

XPath

- XML dokumentum elemeinek eléréséhez és azokra való hivatkozáshoz, a fa struktúrában egy útvonalat ad meg
- Számos segédfüggvényt definiál
- **Csomópont neve**: adott nevű csomópont gyerekei
- */*: gyökérelem
- *//*: aktuális elemtől
- **.**: aktuális elem
- **..**: aktuális elem szűlője
- **@**: attribútum
- ***: bármely elem
- **[]**: predikátumok, szűréshez
- pl.: /page/ROWSET/ROW[@num > 1 and @num < 5]

Osztályok

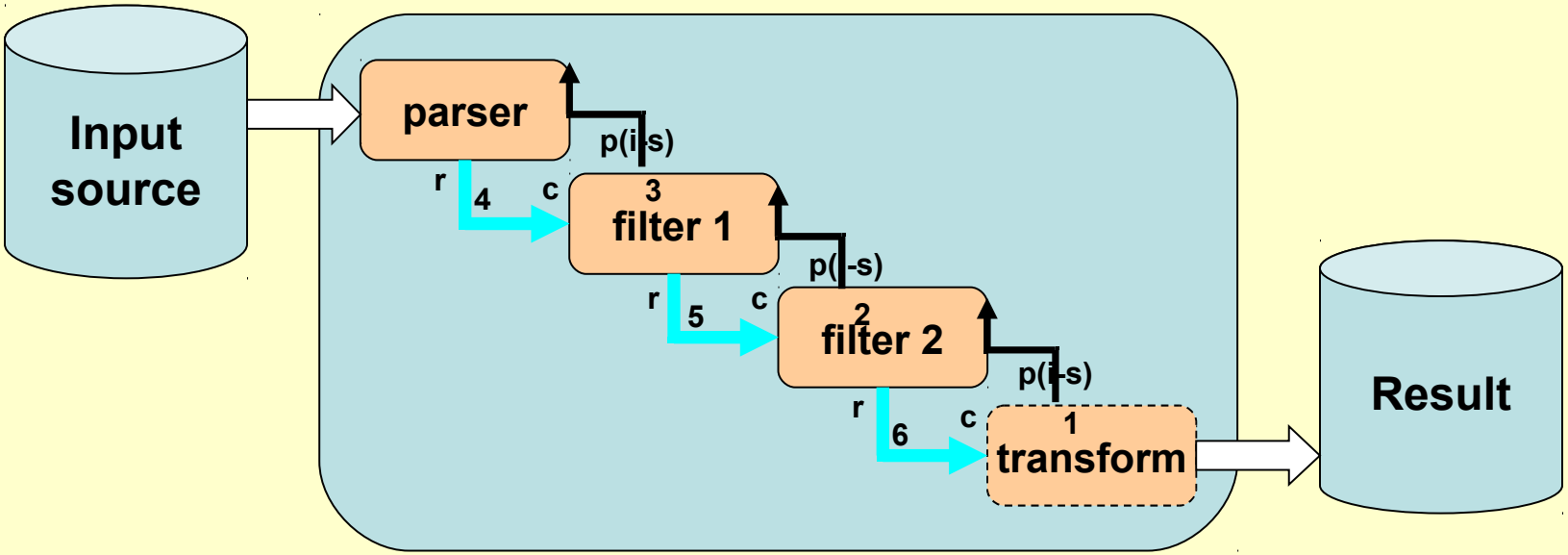
- `javax.xml.transform`
  - `TransformerFactory`
  - `Transformer`
    - Rules (xslt)
    - Source
    - Result
- `javax.xml.transform.dom`
  - `DOMSource`
  - `DOMResult`
- `javax.xml.transform.sax`
  - `SAXSource`
  - `SAXResult`
  - `SAXTransformerFactory`
- `javax.xml.transform.stream`
  - `StreamSource`
  - `StreamResult`
- `javax.xml.transform.stax`
  - `StaxSource`
  - `StaxResult`

Transzformáció DOM alapján

```
...
Document document = builder.parse(datafile);


TransformerFactory tFactory = TransformerFactory.newInstance();
StreamSource rules = new
   StreamSource("http://example.com/stylesheet.xsl");
Transformer transformer = tFactory.newTransformer(rules);


DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
```

Transzformáció SAX alapján

- Elemek: parser, filterek, transform



- **r**: reader, SAX események generálása
- **c**: content handler, a generált esemény lekezelése
- **p(i-s)**: parse(inputSource), az inputsource feldolgozása

Transzformáció SAX alapján

```
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser parser = spf.newSAXParser();
XMLReader reader = parser.getXMLReader();
InputSource inputSource = ...

XMLFilter filter1 = stf.newXMLFilter(
   new StreamSource(stylesheet1));
filter1.setParent(reader);
StreamResult result = new StreamResult(System.out);
SAXSource transformSource = new SAXSource(filter1, inputSource);

SAXTransformerFactory stf =
   (SAXTransformerFactory) TransformerFactory.newInstance();
Transformer transformer = stf.newTransformer();
transformer.transform(transformSource, result);
```

JAXP API Összefoglalás

- SAX
  - A dokumentumot eseménysorozattá alakítja.
  - Az eseménykezelőt kell elkészíteni.
- DOM
  - A dokumentumot egy fa struktúrába olvassa.
  - A fa kényelmesen módosítható.
- Transform
  - XSLT alapján elvégzi a DOM vagy SAX elemzett dokumentum transzformációját.

Java WebServices

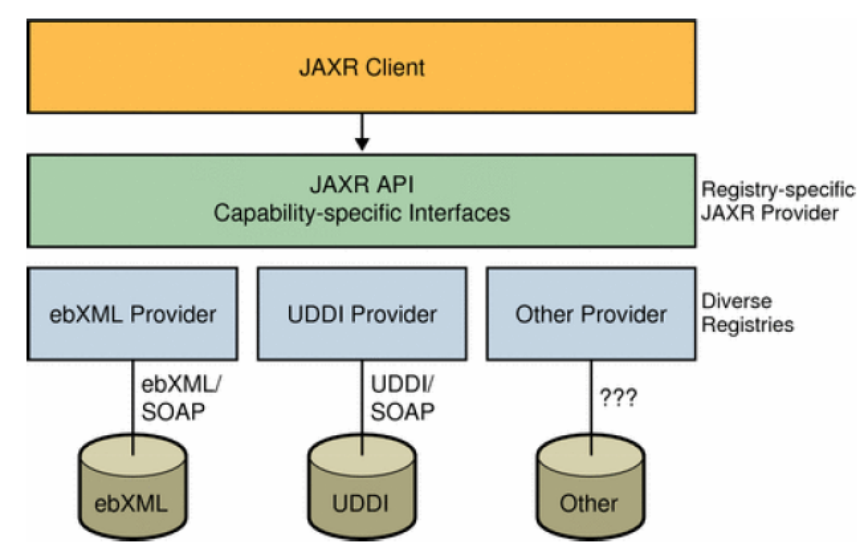
- XML alapú átviteli protokollok
- XML alapú adatcsere
- A Java EE alapértelmezetten tartalmaz minden szükséges eszközt
- Külön csomagként is letölthető (http://java.sun.com/webservices)

WS komponensek

- JAXP (Java API for XML Processing): XML feldolgozók (SAX, DOM)
- **JAXR** (Java API for XML Registries): általános interfész XML registryk hozzáféréséhez
- **JAXB** (Java Architecture for XML Binding): XML alapú osztálygenerálás
- **JAX-RPC** (Java API for XML-based RPC): RPC XML hívásokkal
- **JAX-WS** (Java API for XML Web Services) (EA2)
  - a JAX-RPC-t fogja kiváltani
- **SAAJ** (SOAP with Attachments API for Java): egyszerű SOAP üzenetküldés
- StAX (Streaming API for XML): adatfolyam alapú XML feldolgozók
- XWSS (XML Web Services Security): SAAJ és JAX-WS biztonságossá tételéhez
- XMLDSig (XML Digital Signatures)
- JAX-WSA (Java API for Web Services Addressing)

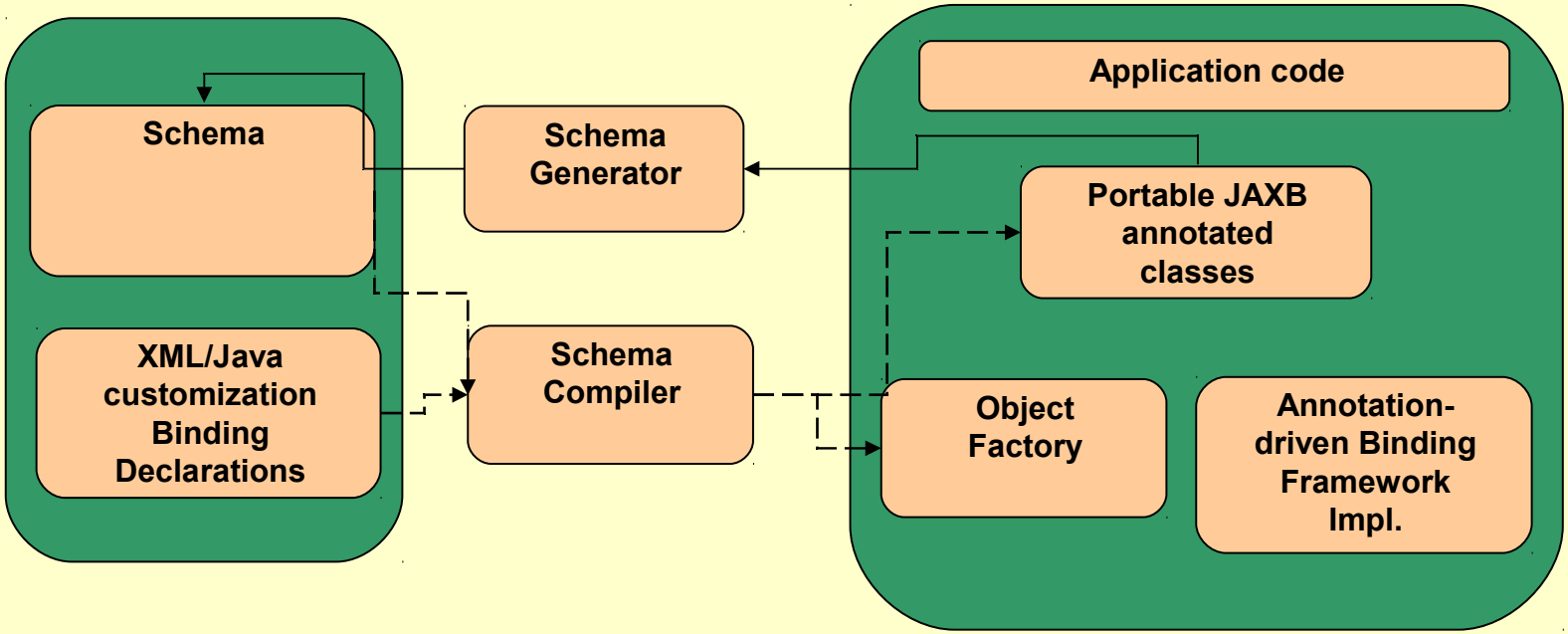
JAXR

- A `javax.xml.registry` és `com.sun.xml.registry` csomagok
- Általánosan használható pl. ebXML, ISO 11179, OASIS, eCo Framework, UDDI registrykkel
- Műveletek
  - böngészés (browse)
  - közzététel (publish)



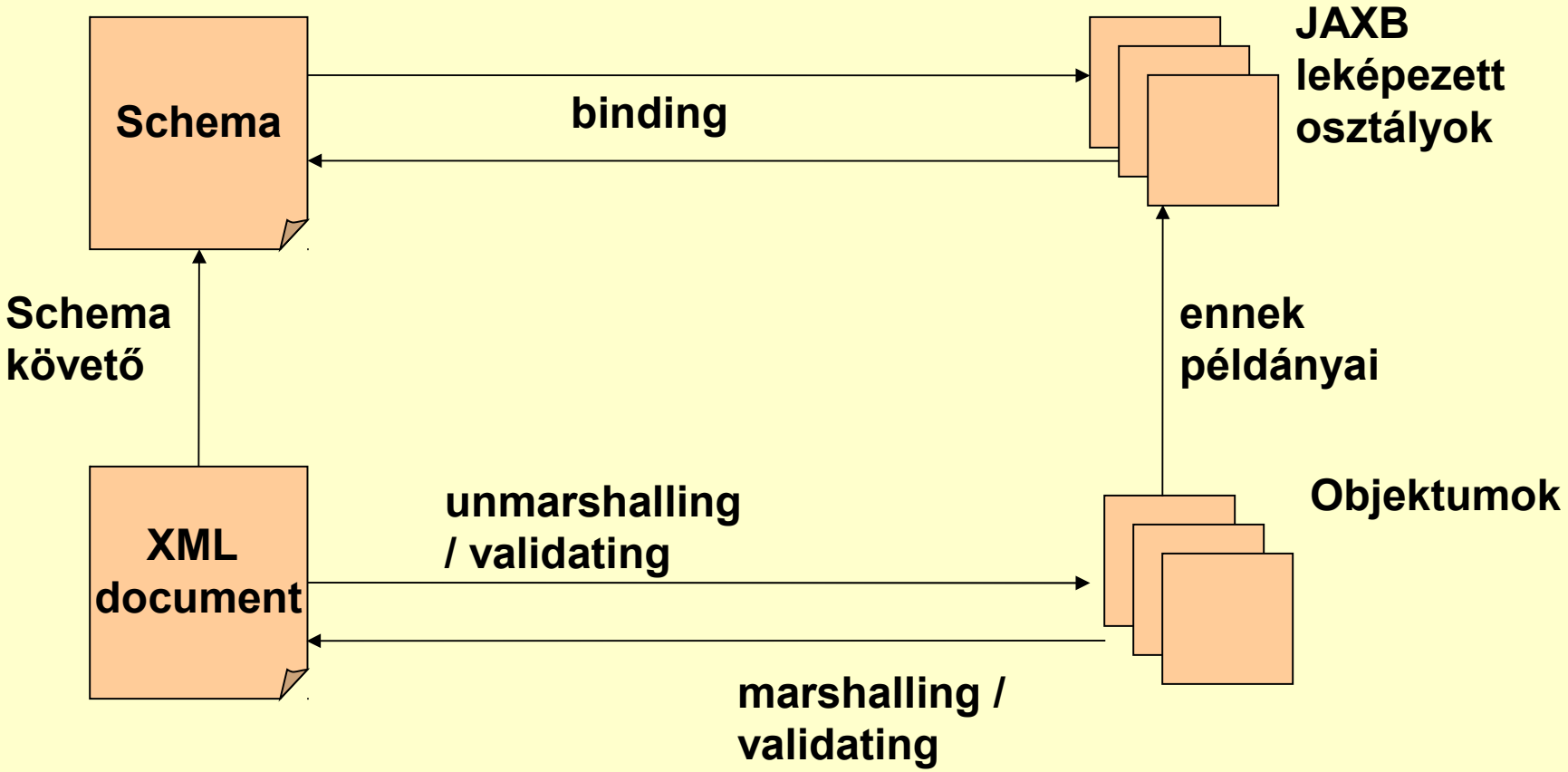
JAXB architektúra

- Java Architecture for XML Binding: XML ←→ Java forráskód leképezést tesz lehetővé
- Eszközök
  - `xjc` (XML-to-Java compiler): sémából forráskód generálása
  - `schemagen <sourcefile(s)>`: sémagenerálás forrásfájlokból



JAXB

- A kötési folyamat

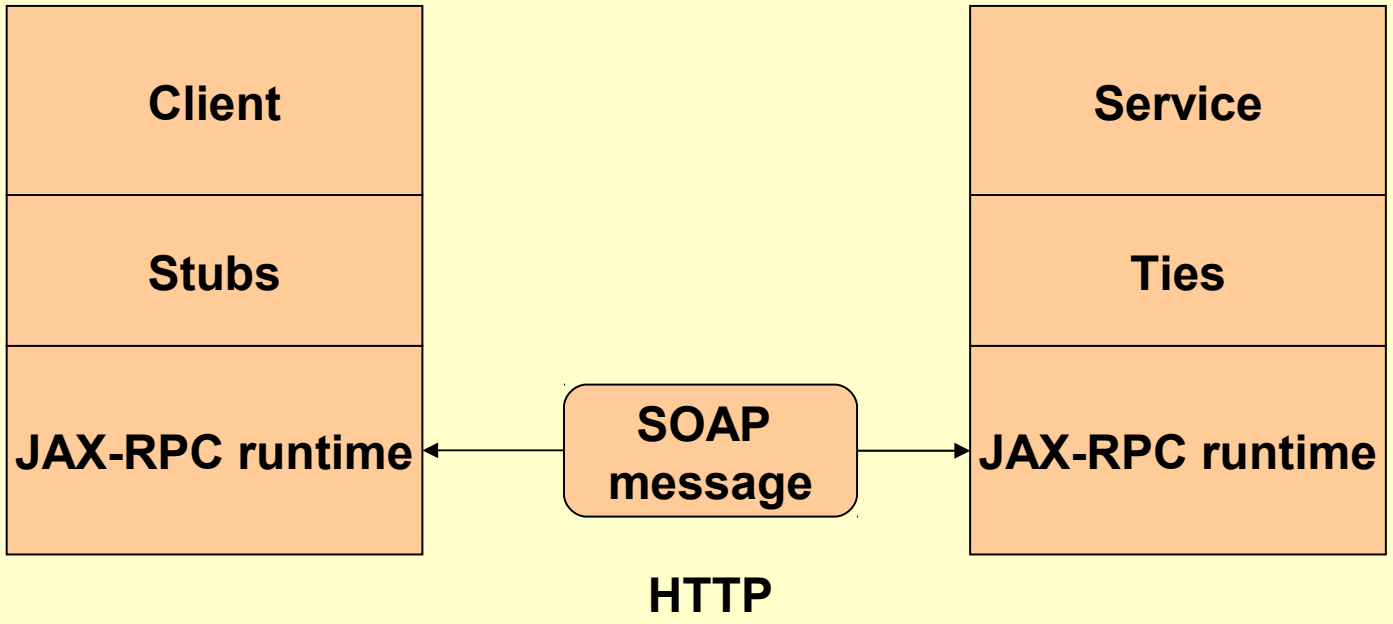


JAX-RPC

- Java API for XML-based RPC
- Lényege:
  - Kliens-szerver típusú szolgáltatás megvalósítása és használata.
  - A webszolgáltatás egy WSDL nyelvű (Web Services Description Language) XML dokumentummal írja le magát.
  - A kliens a WSDL leírás alapján fér hozzá a szolgáltatáshoz.
  - Hozzáférés: távoli eljáráshívás, HTTP felett.
  - Az üzenetek SOAP (Simple Object Access Protocol) message-ek.

JAX-RPC

- Hívási folyamat



JAX-RPC lépések

- Szerveroldal:
  1. Szolgáltatást definiáló interface és implementáció elkészítése, fordítása
  2. `config.xml` elkészítése
     —IF és impl információk
  1. Stub-ok és tie-ok legenerálása (`xrpcc.sh` vagy `xrpcc.bat` használható).
  2. Deployment descriptor elkészítése (`web.xml`)
     —Konfigurációs információk az app. containernek
  1. Service Package készítése (WAR: Web Application Archive)
  2. Elhelyezés a weben (app. container)
- Kiensoldalon:
  1. Kliens kódolása
  2. Fordítás
  3. Futtatás

JAX-RPC – Szolgáltatás definíció IF és Impl

```
package hello;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface HelloIF extends Remote {
   public String sayHello(String s)
   throws RemoteException;
}
```

```
package hello;
public class HelloImpl implements HelloIF {
    public String message = new String("Hello");
    public String sayHello(String s) {
        return new String(message + s);
    }
}
```

JAX-RPC – config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/jax-rpc-ri/xrpcc-config">
   <rmi name="HelloWorldService"
   targetNamespace="http://hello.org/wsdl"
   typeNamespace="http://hello.org/types">
    <service name="HelloWorld" packageName="hello">
        <interface name="hello.HelloIF"
        servantName="hello.HelloImpl"/>
    </service>
   </rmi>
</configuration>
```

JAX-RPC kliens oldal

```
package hello;
public class HelloClient {
  public static void main(String[] args) {
    try {
      HelloIF_Stub stub =
           (HelloIF_Stub)(new HelloWorldImpl().getHelloIF());
      stub._setTargetEndpoint(args[0]);
      System.out.println(stub.sayHello("Szevasz!"));
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

http://example.com:8080/jaxrpc-hello/jaxrpc/HelloIF

JAX-WS

- Java API for XML Web Services

- JAX-RPC helyett (azonos célra); átgondoltabb, összeszedettebb
- Üzenetküldés, hívás u.a., mint a JAX-RPC-nél (HTTP feletti SOAP message)
- Open source, a „Glassfish” projekt részeként (https://jax-ws.dev.java.net/)
- A Java→XML leképezésre a JAXB-t használja
  - Jobb mappinget tesz lehetővé
  - (A JAX-RPC idején még nem volt egy megfelelő JAXB, ezért saját leképezést valósít meg)
- A kódolás az annotációk használatával jóval egyszerűbb

JAX-WS

- Az implementáció menete:
  1. A szolgáltatás osztály kódolása, lefordítása.
  2. WAR készítése.
  3. A szolgáltatás telepítése az app. szerverre. A tie osztályok (szerveroldali váz) automatikusan generálódnak telepítéskor.
  4. A kliens kódolása.
  5. A `wsimport` toollal a stub-ok (kliensoldali váz) előállítása.
  6. A kliens lefordítása és futtatása.

JAX-WS – service

```
package helloservice.endpoint;

import javax.jws.WebService;

@WebService()
public class Hello {
  private String message = "Hello, ";

  public void Hello() {}

  @WebMethod()
  public String sayHello(String name) {
    return message + name + ".";
  }
}
```

JAX-WS – kliens

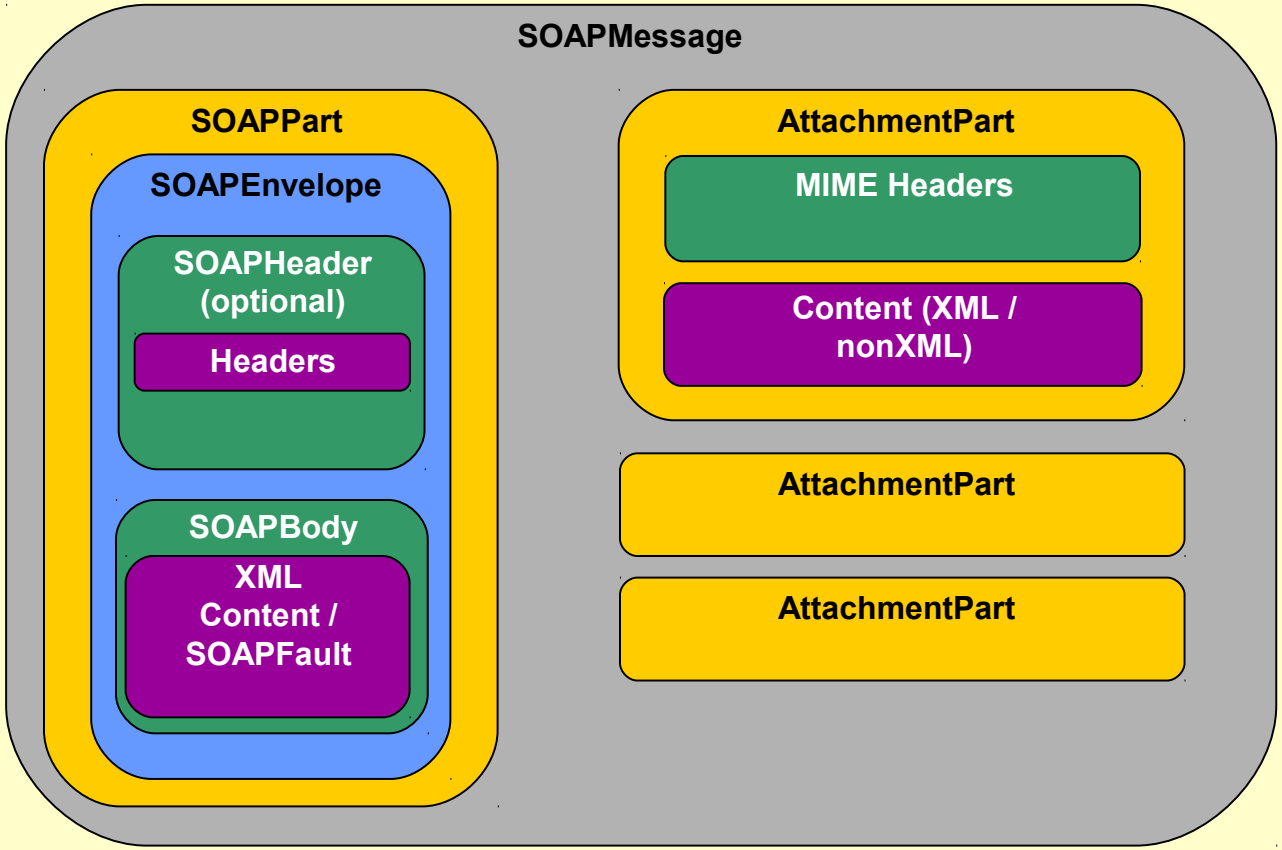
```
public class HelloClient {
  @WebServiceRef(wsdlLocation="http://localhost:8080/
      helloservice/hello?wsdl")
  static HelloService service;

  public void doTest(String name) {
    try {
      System.out.println("Retrieving the port from
          the following service: " + service);
      Hello port = service.getHelloPort();
      System.out.println("Invoking the sayHello operation
          on the port.");

      String response = port.sayHello(name);
      System.out.println(response);
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

SAAJ

- SOAP with Attachments API for Java: `javax.xml.soap`
  - korábban volt: JAXM (Java API for XML Messaging), `javax.xml.messaging`
- Elemek
  - Üzenet (Message)
  - Kapcsolat (Connection)
- Az üzenetek SOAP (Simple Object Access Protocol) formátumú üzenetek

SAAJ – Az üzenet elemei

- SOAPPart: szabványos (W3C) SOAP üzenetet tartalmaz
- AttachmentPart: csatolt tartalom, MIME-típusmegjelölés sel



SOAPMesssage

```
MessageFactory factory = MessageFactory.newInstance();
SOAPMessage message = factory.createMessage();

SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
SOAPBody body = envelope.getBody();
// body kitöltése: addBodyElement(), addTextNode(), ...
Name bodyName = envelope.createName(...);
body.addBodyElement(bodyName);
```

SOAPConnection

```
SOAPConnectionFactory factory =
    SOAPConnectionFactory.newInstance();
SOAPConnection con = factory.createConnection();
// üzenet kitöltése...
java.net.URL endpoint = new java.net.URL(
    "http://example.com/srv"); // valamilyen webservice a végpont
SOAPMessage response = con.call(message, endpoint);
con.close();
```

- Pont-pont kapcsolat
- Blokkoló üzenetküldés

Összefoglalás

- Felületek:
  - JAXP – feldolgozás
  - JAXR – szolgáltatások közzététele
  - JAXB – osztálygenerálás és -példányosítás XML-ből
  - JAX-RPC, JAX-WS – elosztott hívási környezet kialakítása
  - SAAJ – üzenetküldés
- XML
  - bemeneti adat
  - paraméter, használati utasítás
  - üzenetformátum
- Szabványos felület.