# XML adatkezelés

## A SAX szabvány

# A SAX szabvány

Az előadás anyaga

Kovács László: Adatkezelés XML környezetbe és

Kollár Lajos, Sterbinszky Nóra: Programozási technológiák

jegyzete és további irodalom alapján készült el

# Témakör kérdései

1. A SAX alapjai – típusok
2. SAX programozási modell
3. A SAX működési modellje
4. A SAX Java API elemei
5. SAX kezelő felület
6. SAX kezelő felület - SAX API elemek
7. SAX kezelő felület - SAX API mintapélda
8. SAX kezelő felület  - XML dokumentum validálása

# Igényelt kompetenciák

- A SAX alapjai (típusok) megismerése
- SAX kezelő felület - SAX API elemek áttekintése
- Környezet: XML szerkesztő (Oxygen, jDeveloper, EditIX, Eclipse, ….)

# XML jellemzése - ismétlés

- *„Szöveges dokumentum formátum*
  – lehetővé teszi a *szövegtartalom könnyű gépi feldolgozását*,
  – *keresés* helyett *lekérdezéssel jutunk az információhoz*.
- *Metanyelv*
  – a tartalom leírására (jelölésére) *mesterséges nyelvet alkothatunk*,
  – ezáltal lehetővé válik a *jelölt tartalom gépi „megértése”*.

# XML jellemzése - ismétlés

- *Szabványcsalád*
  - – széles körben *elfogadott ipari szabványai vannak*,
  - – gyártói egyetértés.
- *Technológia*
  - – sok területen alkalmazhatunk *XML-re épülő megoldásokat és eszközöket.*

# XML család – ismétlés

**Az XML-lel kapcsolatos specifikációk**: az XML lehetőségeit bővítik.

- Lehetővé teszik XML dokumentumok *szerkezetére* és *tartalmára vonatkozó megszorítások kifejezését* (*XML sémanyelvek).*
- Lehetővé teszik XML dokumentumokból *információ kinyerését (lekérdező nyelvek).*
- Lehetővé teszi XML dokumentumok *más formába alakítását (transzformációs nyelvek).*

# XML család – ismétlés

- **Alkalmazások**: alkalmazási terület-specifikus XML formátumok
  - Webes tartalomszolgáltatás (pl. Atom, MathML, RSS, SVG, XHTML)
  - Tartalomszolgáltatás (pl. DocBook, DITA, EPUB)
  - Kommunikáció (pl. XMPP, Extensible Messaging and Presence Protocol – azonnali üzenetküldő protokoll)
  - Speciális célú szabványos formátumok (pl. KML (Keyhole Markup Language), X3D)
  - Szemantikus web (pl. RDF, XMPP)

# XML család – ismétlés

- **Alkalmazásprogramozási interfészek** (**API-k**):

  – Lehetővé teszik XML dokumentumok feldolgozását programnyelvekből (pl.: SAX, StAX, DOM, JAXB, JAXP, JDOM).”

# XML lehetőségeit bővítő specifikációk

- *Associating Style Sheets with XML documents 1.0 (Second Edition)* (W3C ajánlás, 2010. október 28.) https://www.w3.org/TR/xml-stylesheet/
- *Namespaces in XML 1.0 (Third Edition)* (W3C ajánlás, 2009. december 8.) https://www.w3.org/TR/xml-names/
- *XML Base (Second Edition)* (W3C ajánlás, 2009. január 28.) https://www.w3.org/TR/xmlbase/
- *XML Inclusions (XInclude) Version 1.0 (Second Edition)* (W3C ajánlás, 2006. november 15.) https://www.w3.org/TR/xinclude/
- *XML Information Set (Second Edition)* (W3C ajánlás, 2004. február 4.) https://www.w3.org/TR/xml-infoset/
- *XML Linking Language (XLink) Version 1.1* (W3C ajánlás, 2010. május 6.) https://www.w3.org/TR/xlink11/

*Forrás: JeszenszkyP:XML*

# XML szabvány -ismétlés

- *Extensible Markup Language (XML) 1.0 (Fifth Edition)* (W3C ajánlás, 2008. november 26.) https://www.w3.org/TR/xml/
  – Ez az elterjedten használt, de vele párhuzamosan létezik az XML 1.1 szabvány.
- *Extensible Markup Language (XML) 1.1 (Second Edition)* (W3C ajánlás, 2006. augusztus 16.) https://www.w3.org/TR/xml11/
  – Nem elterjedt a használata.

# Mi az XML dokumentum szabvány erőssége?

Mi az XML dokumentum szabvány erőssége?

- szerkezet rugalmassága,
- metaadatok közvetlen csatolása,
- számtalan *feldolgozó program, leíró nyelv* létezik hozzá.

Az XML dokumentumok *szerkezete szabványos*, ezért a *feldolgozó programok univerzálisan felhasználható* az XML dokumentumok kezelésében.

# XML dokumentumok kezelése, feldolgozása

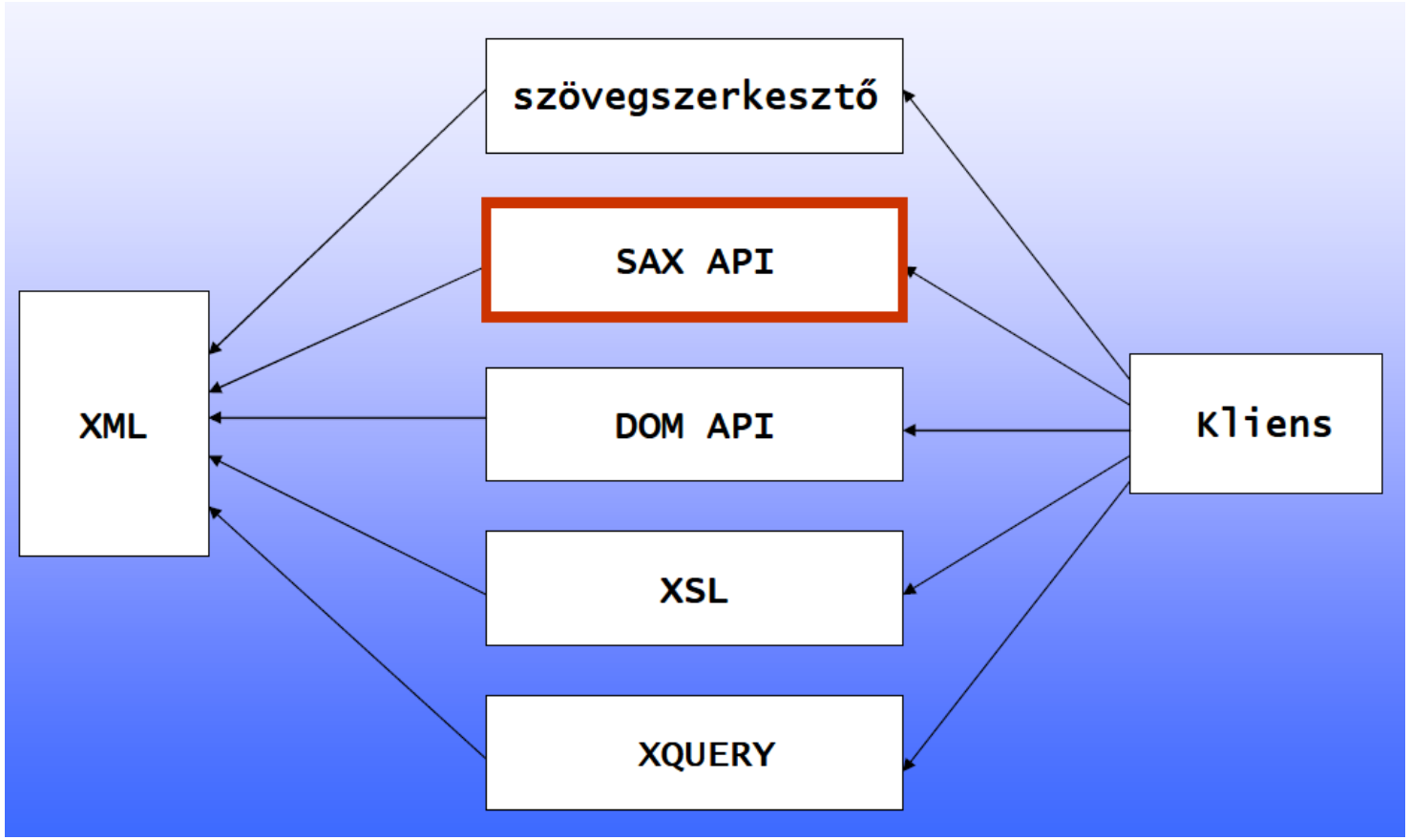
„Az *online* és *egyéb rendszerek* az *adatok tárolására* és *továbbítására legtöbbször* a széles körben elterjedt *XML formátumot használják*.

Az XML dokumentum *helyes* és *hibátlan feldolgozása* megköveteli, hogy a *dokumentum jólformált* és *érvényes le*gyen.

Legismertebb séma nyelvek: DTD, XMLSchema, Relax etc.

# XML adatok kezelési lehetőségei

„Több XML-API szabvány fejlődött ki.



# XML dokumentumok kezelése, feldolgozása

A Java nyelv *több API-t* is biztosít XML dokumentumok feldolgozásához.

Ezek összefoglaló neve JAXP (Java API for XML Processing).

A JAXP háromféle *XML feldolgozó* program könyvtárat tartalmaz:

- Simple API for XML (SAX), és a
- Streaming API for XML (StAX)
- Document Object Model (DOM)

# XML dokumentumok kezelése, feldolgozása

Az ***XML-feldolgozók*** olyan ***programok***, amelyek képesek XML-dokumentumokat *beolvasni*, továbbá *hozzáférést biztosítanak* a dokumentum *tartalmához és szerkezetéhez*.

Mindhárom *API közös jellemzője*, hogy az ***Absztrakt gyár*** (Abstract factory) *tervezési mintát* használja az *XML elemző létrehozására*.

# JAXP API - áttekintés

A JAXP API központi csomagja a `javax.xml.parsers` csomag.

Ebben *gyártófüggetlen absztrakt gyár osztályok* helyezkednek el:

- SAXParserFactory,
- DocumentBuilderFactory és a
- TransformerFactory.

Ezek egy *SAX* (SAXParser) és egy *DOM stílusú* (DocumentBuilder*) feldolgozót*, valamint egy *XSLT-transzformátort* tartalmaznak.

# JAXP API - áttekintés

A három *feldolgozási modell* közül a *SAX csak létező dokumentumok feldolgozását* teszi lehetővé.

A másik két módszer (a DOM és a StAX) a létező dokumentumok feldolgozásán túlmenően egyaránt alkalmas *dokumentumok létrehozására* is.

# A SAX alapjai - Bevezetés

A *segédprogramokat osztályozhatjuk* aszerint, hogy *milyen formában fogadják a végrehajtandó parancssort*.

**1.** *gazdanyelvi API elemekkel meghívott utasítások*,

**2.** XML formátumú parancsállományok,

**3.** speciális nem-XML formátumú parancsállományok,

**4.** speciális, nyelvbe épített modulokkal.

# A SAX alapjai - Bevezetés

**1.** Egy *programozási nyelv utasításai* között lehet elhelyezni *az XML-kezelő API hívásait*.

Ezzel módszerrel írhatunk *XML adatkezelő utasításokat* végző C, Java vagy Pascal programokat.

Ide tartozik: *SAX* és a *DOM API szabványok*.

# A SAX alapjai - Bevezetés

**2.** Egy XML formátumban megírt dokumentumot, mint

*parancs állományt, programállományt* értelmezi a feldolgozó.

Ehhez elegendő egyetlen *speciális XML dokumentum* formát elsajátítani.

Ide tartozik a *XSLT, XMLSchema szabványokat*.

# A SAX alapjai - Bevezetés

**3.** Egy *szövegállományt hozhatunk* létre, melyben egy *speciális parancsnyelv utasításai* vannak letárolva.

Ezt képviseli a *XQuery nyelv*.

**4.** Egy *létező parancsnyelvet* egészítenek ki olyan m*odulokkal*, melyek *egyedi beágyazási lehetőséget adnak* az adott *rendszer parancsainak kibővítésére* az XML dokumentumok kezelésére. Ez az *SQL nyelv.*

# A SAX alapjai

**1.** Egy *programozási nyelv utasításai* között lehet elhelyezni *az XML-kezelő API hívásait*.

Gazdanyelvként a *Java nyelvet*, s a minták megvalósítását az *Oracle JDeveloper* ingyenes *Java fejlesztővel is* végezzük el.

Fejlesztő-környezetek közül: *Eclipse*, *NetBeans, STS* etc.

Az SAX-t eredetileg Java-ban alkalmazták, de szinte az összes fő *programozási nyelv* támogatja.

# Mi a SAX?

- A SAX egy *eseményvezérelt API*, amely *soros elérést biztosít* az XML dokumentum elemeihez.
- Az *egyes elemek* feldolgozása független a korábban feldolgozott elemektől.
- *SAX feldolgozás* során szükség van egy ún. *kezelő objektumra* (handlerre) is, amelyben leírjuk, hogy az XML dokumentum egy adott részének beolvasásakor mi történjen.

# A SAX nyelv célja

A *nyelv célja:* olyan API felület biztosítása, amely lehetővé teszi a megadott *XML dokumentum szerkezetének feltárását*.

A SAX elnevezés a *Simple API for XML* (egyszerű programozási felület az XML-hez).

Az 1990-es évek végén *az XML-DEV(eloper) levelezőlista* tagjainak közös munkája teremtette meg a SAX szabványt.

# A SAX nyelv célja

*David Megginson (kanadai)* javasolta az *API működési módját* és négy hónap alatt fejezték be az *XML-DEV levelezőlista* tagjainak segítségével.

A SAX 1.0 - 1998 májusában lett kész.

Jelenleg a SAX 2.0.2 az aktuális változat (2004. 04.27).

*Lásd:* `www.saxproject.org` (Ez a SAX hivatalos weboldala)

*Számos parser támogatja*, pl. a standard JDK-ban van SAX parser.
https://docs.oracle.com/javase/8/docs/api/javax/xml/parsers/SAXParser.html

# XML API-k

Az XML API-knak két fő típusa van:

*Event-based API:* az *eseményalapú API* az *eseményeket* közvetlenül a *gazdanyelvnek* jelentik un. *callback mechanizmuson* keresztül – nem épít fát.

Ilyen **a SAX.**

*Tree-based API:* az XML dokumentumot egy *belső fa struktúrába képeznek* le, majd lehetővé teszik egy alkalmazás számára, hogy az adott fában navigáljon. Ilyen a DOM.

# SAX programozási modell – előnyök

*Előnyök:*

- *Egyszerű XML parser* (bár a nagy, séma validáló XML parser-ek bonyolultak tudnak lenni, pl.: százmillió ember adatai vannak benne).
- *Minimális memóriaigény* - a teljes XML dokumentumot sosem kell tárolni a memóriában.
- *Inkrementális feldolgozás egyszerű* (az XML dokumentumot a beérkezése közben dolgozza fel).

# SAX programozási modell – hátrányok

*Hátrányok:*

- Nehezebben áttekinthető alkalmazáskód, bonyolult.
- XML lekérdezést, feldolgozást segítő API -nak szüksége van a teljes dokumentum elérésére.
- Nem tudja módosítani a dokumentumot.
- Előre ismerni kell az XML dokumentum felépítését.

# A SAX modell jellemzése

*A SAX jellemzése:*

- csak olvassa a dokumentumot, nem módosít,
- csak szekvenciálisan, egyszer olvassa át a dokumentumot,
- kis erőforrásigényű.

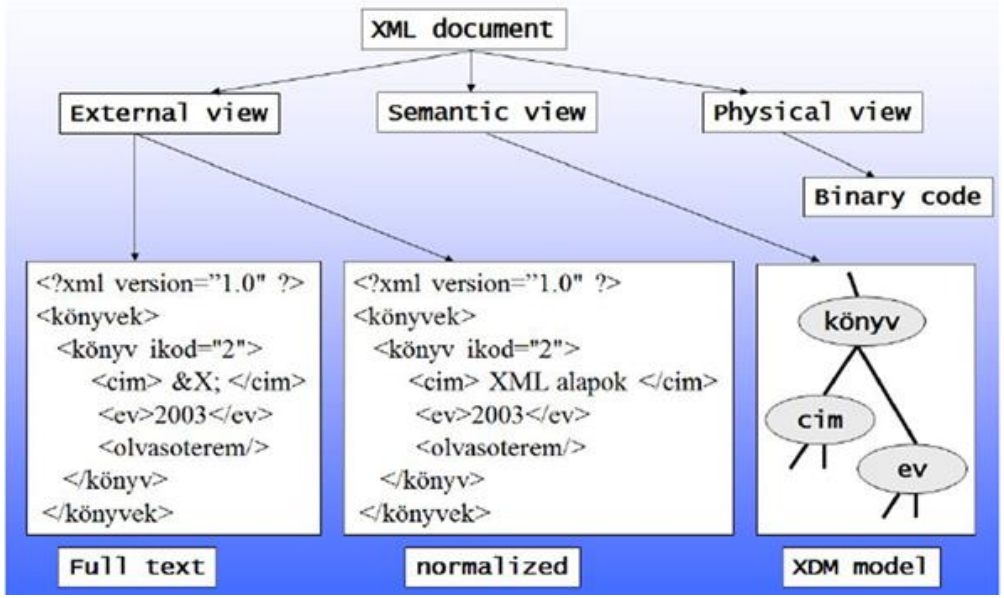
# A SAX modell jellemzése

- Csak olvasási műveletet igénylő feldolgozásokat támogatja.
- Megkönnyíti a tartalom feldolgozását.
- Elterjedt szabvány.
- Könnyű implementálni.
- Összetettebb feldolgozó API-k épülnek rá.
- Az XML dokumentumot szövegfile-ként lehet előállítani.
- Több forrásnyelvhez is implementált.

# Mire jó a SAX?

*Mire jó tehát a SAX?*

A SAX segítségével lehet feltárni, hogy *milyen építőelemek vannak a dokumentumban*.



# A SAX működési modellje

*Hogyan?*

A *gazdaprogram* (a Java nyelv) a *SAX feldolgozón keresztül* tárja fel az *XML dokumentum szerkezetét*.

A *gazdaprogram feladata:* megfelelően *reagáljon az egyes feltárt elemekre.*

A *gazdaprogram* és a *SAX feldolgozó* közötti kommunikáció *struktúrája* az ún. *CALLBACK mechanizmuson* alapszik.

# A SAX működési modellje - kitérő

*Lényege:* a gazdaprogramban szerepelni kell olyan *feldolgozó moduloknak*, melyek alkalmasak az *események kezelésére.*

*Callback* (visszahívás): olyan programozási modellt értünk, amikor *egy futtatható kódot* egy *másik futtatható kódnak* adunk át, amely a *saját logikája alapján meghívja azt*.

Java nyelvben az *interfészek* és a *késői kötés* szolgálja a *callback mechanizmus* megvalósítása során:

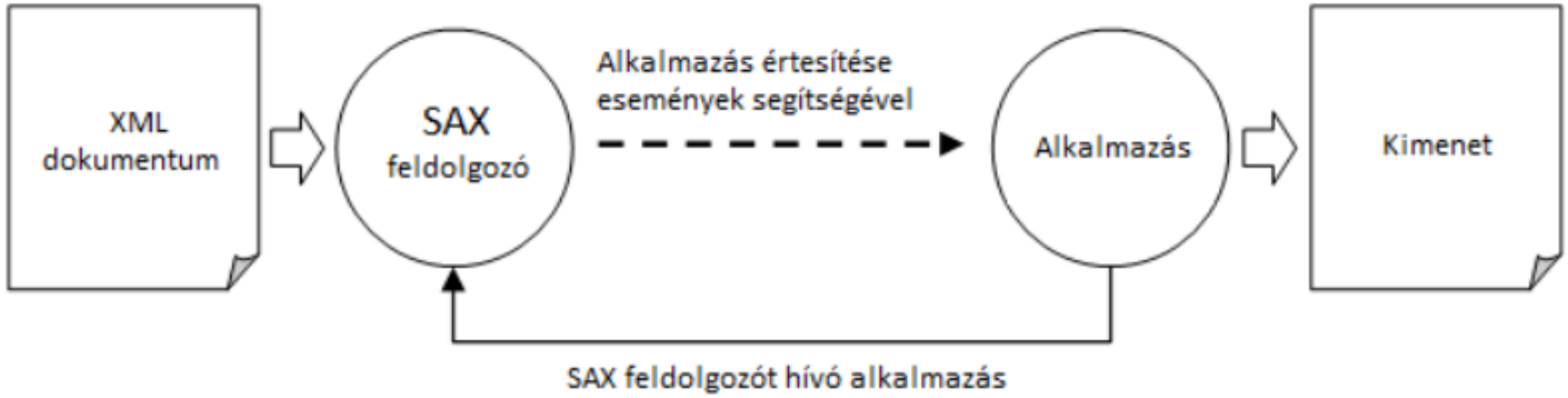
# A SAX működési modellje

- A SAX API által definiált *interfészt* a programozó implementálja (ez a kezelő), de meghívni a SAX implementáció fogja.
- Az elemző (parser) a k*ezelőben* definiált *callback metódusokat hívja*, amikor beolvassa az XML dokumentum egyes részeit.
- Ilyen pl.: dokumentum kezdete és vége, egy XML nyitócímke ill. zárócímke, etc. – hatására meghívja a kezelőben leírt, az adott események kezelésére szolgáló *callback metódusokat.*

# A SAX működési modellje

A *SAX feldolgozó* az XML dokumentum beolvasása során ha talál ún. *elemzési események* – akkor meghívja *callback metódusokat*.

**Az XML dokumentumok SAX stílusú feldolgozásának sematikus modellje**



# A SAX működési modellje

*Következő lépés:* SAX modul elkezdi szekvenciálisan feldolgozni a XML dokumentumot.

- Ha az *esemény* bekövetkezik (pl. új elem kezdetének észlelése), SAX modul megkeresi, *hogy a gazdanyelvben melyik modul felelős* az *eseményért.*
- Ha megtalálta a megfelelő *eseménykezelőt*, akkor *átadja neki a vezérlést* a *paraméterekkel együtt*.
- Ezután a vezérlés vissza kerül a SAX modulhoz, amely az előbbi helyről folytatja tovább az XML dokumentum feltárását.

# A SAX működési modellje

*A SAX modul* és a *gazdaprogram* között szabványosítani kellett az *interfészt*.

Az *interfész* megadja: a *meghívandó metódusok szignatúráját, nevet* és *paraméterezését*.

A SAX modell az alábbi *eseményosztályokat* definiálja:

- dokumentum kezdete,
- dokumentum vége,
- elem kezdete,

# A SAX működési modellje

- elem vége,
- karakter rész elérése,
- entity észlelése,
- direktíva észlelése,
- névtér definíció.

Ezekre az eseményekre lehet reagálni, melyek közül egyik sem kötelező.

# A SAX működési modellje - feldolgozó keret feladata

A *SAX feldolgozó keret feladata*: a *kliens program* értesüljön az *XML dokumentum szerkezeti elemeiről*.

*A SAX az alábbi tevékenységet végzi:*

- *elemhatárok* felismerése,
- *elemjellemzők* és *értékeinek felismerése* és összerendelése,
- kara*kterek összevonása egységbe*, szövegértékké,
- *szabványos hibakezelés*,
- az *esemény helyének átadása egy speciális osztályon* (Locator).

# A SAX működési modellje

*A SAX modul komponensei:*

- a hibakezelés (ErrorHandler)
- az elemjellemzők kezelése (AttributeList)
- a szimbólumok kezelése (EntityResolver)
- a sémakezelés (DTDHandler)

# A SAX Java API elemei - működési modellje

A *gazdaprogramban* megfelelő sorrendben kell meghívni a *SAX modult kezelő utasításokat.*

*Ez a következő lépésekből áll:*

**1**. dokumentum olvasó létrehozása,

**2**. tartalomkezelő keret létrehozása,

**3**. az eseménykezelő metódusok létrehozása,

**4**. hibakezelő metódusok létrehozása.

# A SAX Java API elemei - működési modellje

A *SAX kezelő eljárások elérése* a megfelelő *csomagok importálásan* keresztül érhető el – ez a W3C SAX modulja.

Az *import utasítások a programkód elején* az alábbi formátumban adhatok meg.

# SAX Java API elemei - csomagok importálása

Az *import utasítások a programkód elején:*

```
import java.io.File;
import java.io.IOException;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
```

# A SAX Java API elemei - működési modellje

A *gazdaprogram fő programkódjában kell példányosítani* a *SAX kezelőt* és a *saját eseménykezelőt*.

A *SAX kezelőnek* át kell adni az *eseménykezelő objektum elérési adatait.*

Ezek után lehet meghívni a *SAX feldolgozót végző eljárását*, melyet a *PARSE modul hajt végre*.

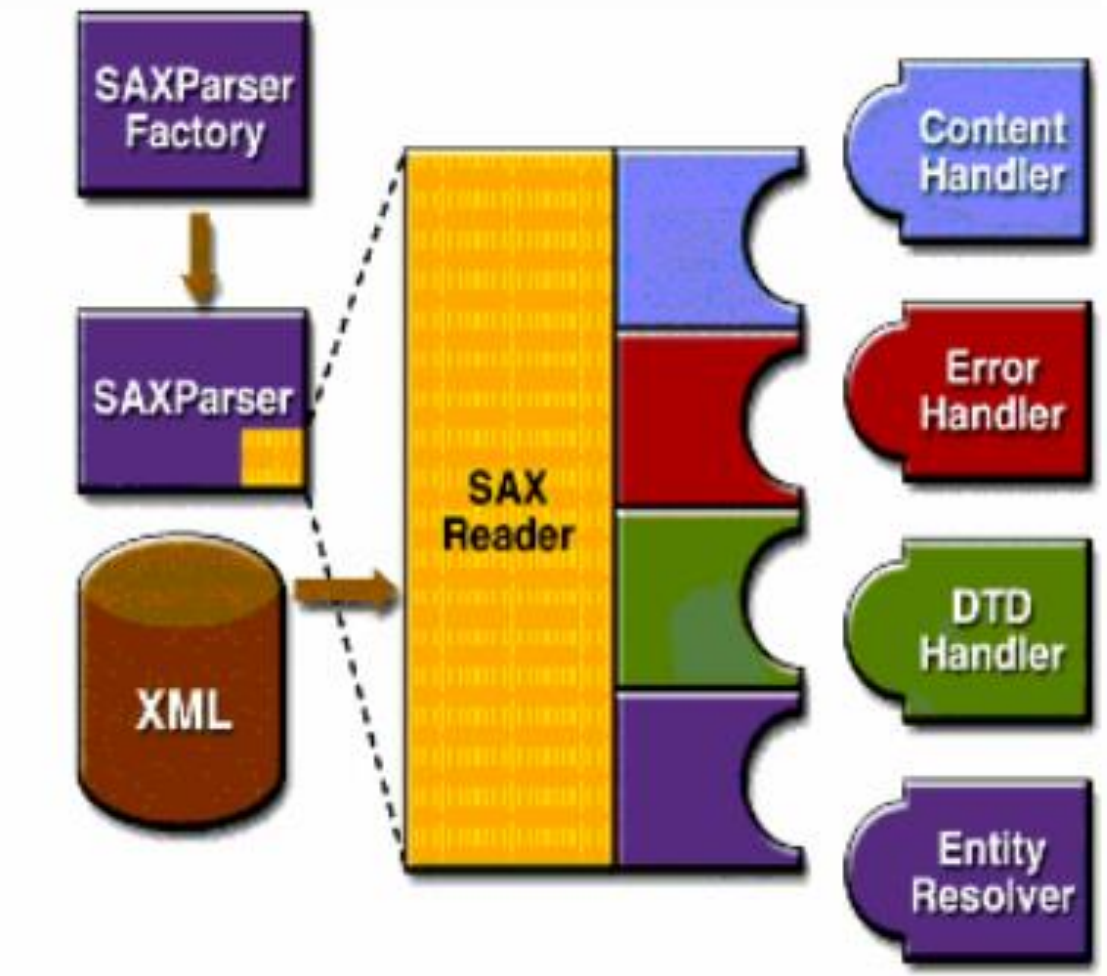
https://docs.oracle.com/javase/7/docs/api/org/xml/sax/package-summary.html

# A SAX Java API elemei

`SAXParserFactory` osztály feladata: *XML-feldolgozó létrehozása,*

amely a `javax.xml.parsers` `SAXParserFactory` csomagban található

Lásd:




https://docs.oracle.com/cd/E11882_01/appdev.112/e10769/javax/xml/parsers/SAXParserFactory.html

# A SAX Java API elemei

`SAXParser`: ez reprezentálja magá

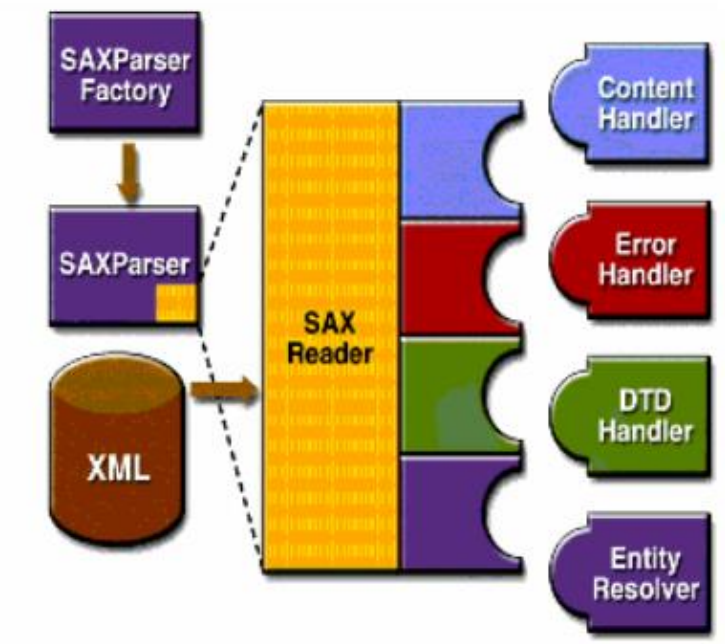
*absztrakt osztályt.*

*Legfontosabb metódusai*: a `parse` m

képesek (`File`, `InputStream`, `Inpu`

`String`) az XML dokumentumok feld

Valójában a feldolgozás folyamatát hat



# A SAX Java API elemei

`SAXReader`: A *SAXReader-nek* kell kapcsolatot létrehozni *a SAX eseménykezelővel*.

Itt adhatók hozzá az *interfészeket megvalósító objektumok* a *feldolgozást végző kódhoz* - metódus segítségével (pl.: a void setContentHandler( ContentHandler handler );

# A SAX Java API elemei

`ContentHandler:` Ez a dokumentum *logikai tartalmáról, főbb elemzési eseményeiről* értesítést kapó interfész.

Ennek metódusait hívja majd a SAXParser a feldolgozás során.

**Főbb metódusai:**

- startDocument (a dokumentum kezdetéről kap értesítést),
- endDocument (a dokumentum végének olvasásakor kerül meghívásra),

# A SAX Java API elemei

- startElement (egy elem kezdetekor hívódik meg),
- endElement (egy zárócímke beolvasásakor kerül meghívásra),
- characters(), amely szöveges adat olvasására reagálva aktivizálódik.

*ErrorHandler* interface metódusai*:* az *error, fatalError,* és a *warning* akkor kerülnek meghívásra, ha elemzési hibák következnek be.

# A SAX Java API elemei

- *DTDHandler:* a *dokumentumtípus-deklarációkhoz* kapcsolódó *bizonyos események* (jelölések és elemzetlen egyedek deklarációja) esetén kerülnek meghívásra a metódusai.
- *EntityResolver:* metódusai a külső egyedek feldolgozása esetén hívódik meg.

# A SAX Java API elemei – egyszerű példa

Adott a következő XML kód egy része: a kurzusnev elemből nyerjük ki a tartalom elemet.

```
<kurzus id="GEIAL332-B">
      <kurzusnev>Adatkezelés XML kornyezetben</kurzusnev>
      <kredit>5</kredit>
      <hely>XXXII.</hely>
      <idopont>Kedd 12:00-13:30</idopont>
      <oktato>Dr. Kiss Ferenc </oktato>
</kurzus>
```

# A SAX Java API elemei – egyszerű példa

## XML SAX Handler

```
startElement(paraméterek) {
        if (name == "kurzusnev")
                kurzusnevElement = true
}

characters(paraméterek) {
        if (kurzusnevElement)
                result = new String(paraméterek)
}

endElement(paraméterek) {
        if (name == "kurzusnev")
                kurzusnevElement = false
}
```

# A SAX Java API elemei

Ezt a *négy interfészt* egy `DefaultHandler` objektum fogja össze.

Tehát, az **elemző:**

- egy *XML adatforrást* és
- egy *DefaultHandler* kap,

majd ezután feldolgozza az XML *dokumentumkezelő objektumot*, miközben meghívja a *kezelőobjektum megfelelő metódusait*.

https://docs.oracle.com/javase/7/docs/api/org/xml/sax/helpers/DefaultHandler.html

# A SAX Java API elemei

`DefaultHandler`

Az `org.xml.sax.helpers.DefaultHandler` osztály implementálja a `ContentHandler, ErrorHandler, DTDHandler, és EntityResolver` ***interfészeket,*** így elég ezt az *egy osztály kiterjeszteni*, *metódusait implementálni*, és az osztály egy példánya kerül majd paraméterátadásra a feldolgozó `parse` metódusa számára.

# A SAX Java API elemei - példa

Készítsen egy programot, amely kiírja a szabvány kimenetre az észlelt eseményeket.

*Az XML dokumentum feldolgozása a következő lépésekből áll:*

**1**. dokumentum olvasó létrehozása,

**2**. tartalomkezelő keret létrehozása,

**3**. az eseménykezelő metódusok létrehozása,

**4**. hibakezelő metódusok létrehozása.

# A SAX Java API elemei

**Végrehajtott parancsok -** *Dokumentumolvasó létrehozása*

A *try blokk első utasítás* példányosítja a SAXParser értelmezőt.

```
XMLReader parser = new SAXParser();
```

A *második utasítás* a *saját eseménykezelő objektumunkat* hozzunk létre.

```
SAXproba1 Shand = new SAXproba1();
```

# A SAX Java API elemei

A *harmadik utasítás*: *a SAX feldolgozó* modulnak **átadja** a *saját eseménykezelő objektum címet, elérését*.

```
parser.setContentHandler(Shand);
```

Ez a SAX modul `SetContentHandler` metódusán keresztül valósul meg.

# A SAX Java API elemei

A *negyedik utasítás* elindítja az XML dokumentum

feldolgozását, ahol az XML dokumentum elérési adata az `args` változóban tárolt.

```
parser.parse(args[0]);
```

A `catch` részben az *hibák lekezelése* megy végbe.

Amikor a vezérlés visszatér a `parse metódus` hívásból, akkor már megtörtént a dokumentum átolvasása és feldolgozása.

# A SAX Java API elemei

A *feldolgozás magja* a *saját eseménykezelő osztály* kódjában helyezkedik el.

Az *eseménykezelő osztálynak* illeszkednie kell a *SAX szabványban definiált interfészre.*

Így a *saját eseménykezelő osztálynak* egy megadott sémát kell implementálnia.

# A SAX Java API elemei

Az osztály definíciót megadó utasítás váza:

```
public class Skezelo implements ContentHandler {
private Locator loc;
...}
```

A SAX modulban megadott `ContentHandler` interfész megadott *nevű és szignatúrájú metódusokat* tartalmaz.

A `loc` változó az a `Locator` objektum, amelyen keresztül megkapja a gazdaprogram az események helyének azonosítását.

# A SAX Java API elemei

Az `interfész` a következő *`metódusokat`*, és rajtuk keresztül *eseményeket értelmezi*:

```
public void startDocument() throws SAXException
{
...
}       //startDocument() : dokumentum feldolgozás kezdete
public void endDocument() throws SAXException {
...
}       //endDocument() : dokumentum feldolgozás vége
```

# A SAX Java API elemei

```
public void startPrefixMapping(String prefix,
String uri) throws SAXException {
...}     // startPrexMapping(String prex, String uri) :
```

- `startPrexMapping`: névtér alias definiálása,
- a `prex` az alias érték,
- az `uri` a névtér leírás.

# A SAX Java API elemei

```
public void endPrefixMapping(String prefix)
throws SAXException {
...
}
```

//endPrexMapping(String prex): névtér alias hatáskör vége.

# A SAX Java API elemei

```
public void processingInstruction (String target,
String data) throws SAXException {
...}
```
//processingInstruction (String target, String data) :

- `processingInstruction`: feldolgozási direktíva,
- `target`: feldolgozó program azonosítása,
- `data`: utasítás szövege.

# A SAX Java API elemei - startElement

```
public void startElement (String namespaceuri, String
localname, String rawname, throws SAXException {
```

`...` `}` `//` startElement (String namespaceuri, String localname, String rawname, Attributes atts):

`startElement`: elem kezdete,

- `namespaceuri`: névtér,
- `localname`: elem neve,
- `rawname`: belső név,
- `atts`: befoglalt *elemjellemzők* listája.

# A SAX Java API elemei

```
public void endElement (String namespaceuri, String
localname, String rawname) {...
```

```
}
```
//endElement (String namespaceuri, String localname, String raw name): elem vége

```
public void characters (char[] ch, int start, int
length) throws SAXException {
```

```
... }
```
//characters (char[] ch, int start, int length) : szövegrész

# A SAX Java API elemei

```
public void ignorableWhitespace(char[] ch, int
start, int length) throws SAXException {
  ... }
```
//ignorableWhitespace(char[] ch, int start, int length): figyelmen kívül szövegrész

```
public void skippedEntity( String name) throws
SAXException {
...
}
```
//skippedEntity( String name) : szimbólum

# A SAX Java API elemei – Locator osztály

Az osztályok legfontosabb `metódusai` és `adattagjai`.

*`Locator:`* a két legfontosabb *metódusa* az esemény forráskód-beli helyet adja meg:

- `int getLineNumber()` : az eseményhez tartozó rész sorának sorszáma,
- `int getColumnNumber()` : az eseményhez tartozó rész karakterpozíciója a soron belül.

# A SAX Java API elemei

*`Attributes:`* a fontosabb metódusok az *elemjellemzők* adatainak elérésére:

- int getLength() : az elemjellemzők darabszáma
- String getLocalName(int poz) : az i. elemjellemző lokális neve
- String getValue(int poz) : az i. elemjellemző értéke
- String getType(int poz) : az i. elemjellemző típusa
- String getURI(int poz) : az i. elemjellemző névtere

# A SAX Java API elemei

*XMLReader:* metódusok a SAX feldolgozó objektumhoz

kapcsolódóan:

- `void parse(String fnev)` : a megadott XML dokumentum feldolgozásának elindítása,
- `void setFeature(String nev, boolean val)`: feldolgozás módjának beállítása, ahol a név egy megadott funkciót azonosít,
- `void setContentHandler(String fnev)`: eseménykezelő beállítása,
- `void setErrorHandler(String fnev)`: hibakezelő beállítása

# A SAX Java API elemei

*ErrorReader:* a hibakezelő objektumhoz kapcsolódó metódusok:

- `void error(SAXParseException fnev)` : a megadott hiba kiváltása,
- `void fatalError(SAXParseException fnev)` : a megadott végzetes hiba kiváltása,
- `void warning(SAXParseException fnev)` : a megadott figyelmeztetés kiváltása.

# SAX 2 fontosabb osztályai - összefoglalás

*Fontosabb osztályok*

`SAXParserFactory, SAXParser, DefaultHandler`

*Dokumentumolvasó létrehozása*

```
SAXParserFactory factory =
SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
SaxHandler handler = new SaxHandler();
```

# SAX fontosabb osztályai - összefoglalás

*A tartalomkezelő keret, valamint az esemény- és hibakezelő metódusok definiálása:*

a `DefaultHandler`–ből való leszármaztatással és a megfelelő metódusainak felüldefiniálásával történik.

A `DefaultHandler` fontosabb metódusai:

`startElement, endElement, characters`, stb.

# SAX fontosabb osztályai - összefoglalás

*A dokumentum értelmezési folyamatának elindítása*:

```
parser.parse(new File("XML dokumentum"), new
ContentHandler());
```

# SAX szabvány - mintapélda

Készítsen egy SAX programot, amely egy adott XML (kurzusfelvetel.xml) dokumentum teljes feldolgozását végzi el, megjeleníti a konzolon  a dokumentum struktúráját is.

# SAX szabvány - mintapélda

- *Projekt név*: SaxNEPTUNKOD
- *Package: hu.*saxneptunkod
- *File name:* SaxNEPTUNKOD.java
- Class name: SaxNEPTUNKOD
- *XML name*: Neptunkod_kurzusfelvetel.xml

# SAX szabvány - mintapélda

A kód lehetséges felépítése:

**a)** Tartalomkezelő keret létrehozása definiálása.

Itt kell létrehozni az eredmény kiírásban szereplő minden nyitó jelölőelem után *start* és záró jelölőelem után *end*, *{}, : , ;* etc.

# SAX szabvány - mintapélda

**b)** Eseménykezelő metódusok létrehozása.

A következő feltételnek kell teljesülni:

- minden **nyitóelem** után **start** kiírása, - majd újsor
- az elemek között lévő **tartalom** kiírása , - majd újsor
- minden zárelem után **end** kiírása, - majd újsor
- elemjellemzők értékét{id=1} közé írja ki – majd újsor

# SAX szabvány - mintapélda

Három eseménykezelő metódust javaslok a feladat megoldásához:

- *startElement():* mindegyik XML elem elején hívódik.
- *endElement():* az XML elemek végén hívódik.
- *characters():* két tag közötti szövegre hívódik.

**c)** A kód fő programja: try { }catch között legyen és tartalmazza a hibakezelést is.

# A SAX Java API - példa

A teljes kód a SAX API bemutatása  keretében a jegyzetben található.

A program kiírja a  szabvány kimenetre  az észlelt  eseményeket.

# Az XML dokumentum érvényességének ellenőrzése

Az XML dokumentumok ellenőrizhetők Java kódból *jólformáltság* és *érvényesség* szerint, azaz hogy egy előre meghatározott szerkezeti felépítést követnek-e.

Két lehetőség van validálásra:

**1.** Ha ellenőrizni szeretnénk az XML dokumentum *érvényességét*, akkor ezt a `setValidating` metódus segítségével kell a SAX elemzőt előállító gyárral közölnünk.

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

# Az XML dokumentum érvényességének ellenőrzése

- Ekkor a *gyár egy validáló elemzőt fog* legyártani, amennyiben az lehetséges.
- Hogy az *elemző validáló* legyen, explicit módon kell kérnünk, mert alapértelmezés szerint az előállított feldolgozók nem validálnak.

# Az XML dokumentum érvényességének ellenőrzése

**2.** Ha a *validálást XML Schema* ellenében végezzük, a előbbin kívül el kell végeznünk még néhány beállítást.

Elsőként a *gyár állapotának módosításáva*l hatást gyakorlunk a gyártás folyamatára:

- névtérképes és érvényesítő feldolgozót legyártása.
- beállítjuk a legyártott SAX elemzővel kapcsolatban elvárt tulajdonságokat.

Ilyen tulajdonság segítségével jelezzük az XML-feldolgozónak, hogy sémanyelvként az XML Schema nyelvet szeretnénk használni.”

# Felhasznált irodalom


- Kovács László: Adatkezelés XML környezetbe.
- Kollár Lajos, Sterbinszky Nóra: Programozási technológiák – Jegyzet, DE, 2014.
- Dr. Adamkó Attila: Fejlett Adatbázis Technológiák - Jegyzet, 2013.
- Jeszenszky Péter: XML, DE, 2019