JEGYZŐKÖNYV

Webes adatkezelő környezetek
Féléves feladat

Készítette: **Varga Eszter**
Neptunkód: **F9PSJA**
Dátum: **2025. december**

Miskolc, 2025

Tartalomjegyzék

A feladat leírása

A feladat célja egy egyetemi kurzusnyilvántartó rendszer elkészítése, amely az egyetem különböző adatait kezeli, például a hallgatók, oktatók, kurzusok, tanszékek és irodák adatait. A rendszerben minden ilyen elem összekapcsolódik egymással – például egy oktató egy tanszékhez tartozik, egy kurzust több hallgató is felvehet, vagy egy tanszék egy adott irodában működik.

Első lépésként egy ER modellt kellett készíteni, ami bemutatja az egyedeket, a köztük lévő kapcsolatokat (1:1, 1:N, M:N), és az ezekhez tartozó tulajdonságokat. Minden egyed legalább négy jellemzőt tartalmaz, a kapcsolatoknak pedig saját adataik is lehetnek.

A következő lépésben az ER modellt XDM modellé kellett alakítani, ami már az XML struktúráját írja le. Ez alapján készült el maga az XML dokumentum, benne valós, az oktatóval egyeztetett adatokkal. Az ismétlődő elemekből legalább két példány szerepel, és a fájlban megjegyzések is segítik az átláthatóságot.

Végül az XML-hez egy XML séma (XSD) is készült, ami meghatározza, hogy az adatok milyen típusúak lehetnek, mely mezők kötelezőek, és hogyan kell kinéznie a dokumentumnak. A végeredmény így egy jól strukturált, validálható rendszer, ami egyszerűen kezelhető és továbbfejleszthető.

1. Egyetemi kurzusnyilvántartó rendszer

1.1 Az adatbázis ER modell tervezése

A rendszer célja, hogy az egyetemi kurzusokhoz kapcsolódó alapvető adatokat nyilvántartsa, és megmutassa az egyedek közötti kapcsolatokat. A modellben öt fő egyed szerepel: Hallgató, Oktató, Kurzus, Tanszék és Iroda.
Minden egyednek vannak tulajdonságai, ezek a következők:

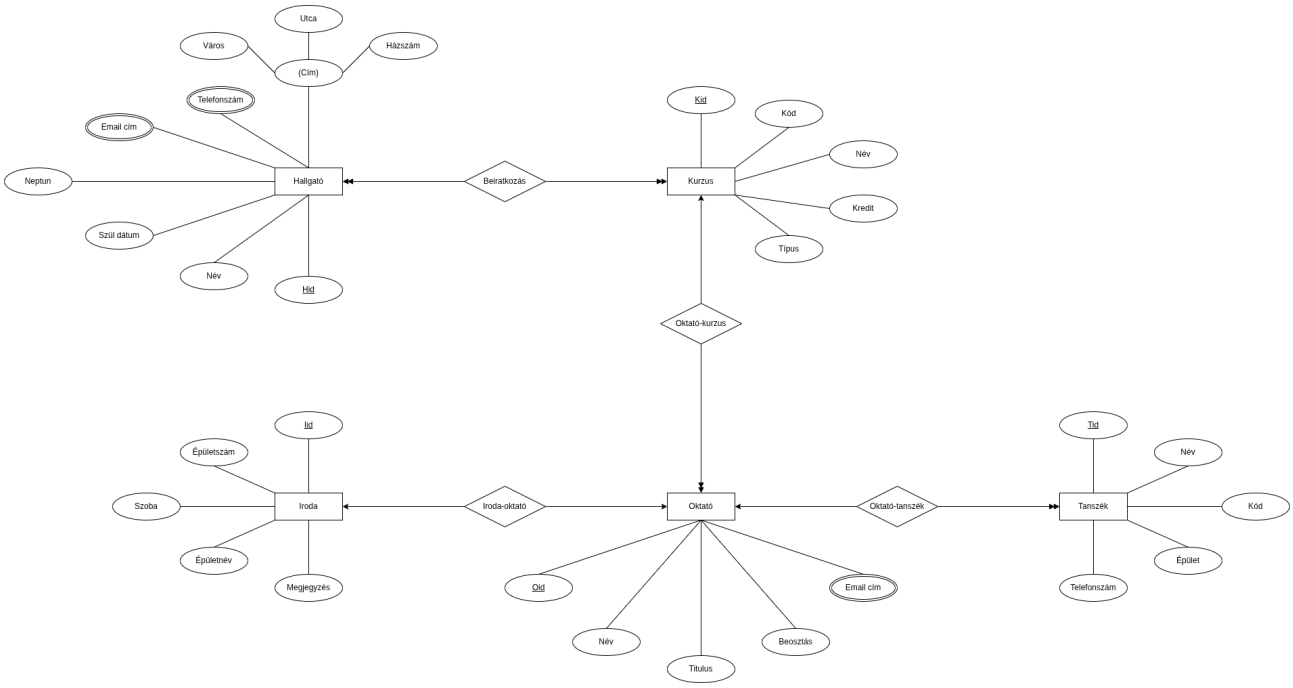
- hallgató:

    - név: a hallgató neve
    - születési dátum: a hallgató születésének dátuma
    - lakcím: összetett, a hallgató lakcímét tartalmazza, város+utca+házszám
    - email cím: a hallgató email címe
    - telefonszám: a hallgató telefonszáma
    - neptunkód: a hallgató neptun kódja
- oktató:
    - név: az oktató neve
    - titulus: az oktató titulusa
    - beosztás: az oktató beosztása
    - email cím: az oktató email címe
- kurzus:
    - kód: a kurzus kódja
    - név: a kurzus neve
    - kredit: a kurzus kreditértéke
    - típus: a kurzus típusa
- tanszék:
    - név: a tanszék neve
    - kód: a tanszék kódja
    - épület: a tanszék épülete
    - telefonszám: a tanszék telefonszáma
- iroda:
    - épületszám: az épület, amiben az iroda van
    - szoba: az iroda szobaszáma
    - épületnév: az iroda épületének neve
    - megjegyzés: kiegészítő információk az irodáról

Mindegyik egyednek van egy kulcs tulajdonsága, egy id-ja, ezen kívül vannak többértékű tulajdonságok(pl. egy oktatónak lehet több email címe is), és összetettek is(a lakcím több részből áll).

Az egyedek közt állnak fenn kapcsolatok is:

- hallgató-kurzus között a beiratkozás a kapcsolat, aminek vannak tulajdonságai is: félév, teljesítve, osztályzat, ez egy N:M kapcsolat, mert több hallgató felvehet több kurzust is
- kurzus és oktató közt 1:N kapcsolat, mivel egy oktatónak lehet több kurzusa is
- oktató és tanszék közt 1:N kapcsolat, mert egy tanszéken lehet több oktató is
- oktató és iroda közt 1:1 kapcsolat, hiszen 1 irodához 1 oktató tartozik, és fordítva is



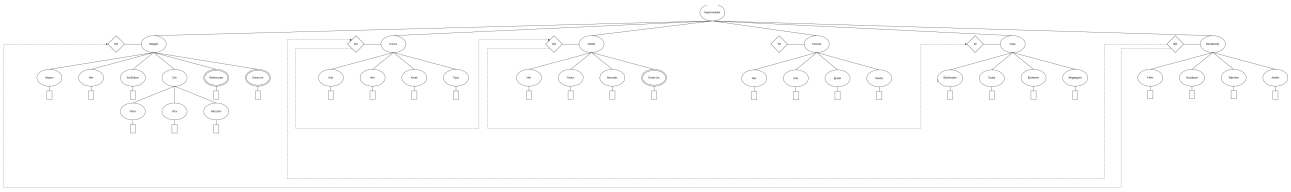
1. kép: Az ER diagram

1.2 Az adatbázis konvertálása XDM modellre

Az XDM modell létrehozásához felhasználtam a korábban elkészített ER modellemet, az alapján állapítottam meg, hogy a különböző elemek hogy helyezkedjenek el egymáshoz képest.

Az XDM modellem gyökéreleme a NeptunAdatok entitás, ebből ágazik le még 6 másik, a Hallgató, a Kurzus, az Oktató, a Tanszék, az Iroda és a Beiratkozás.

A hallgató entitás tartalmazza a hallgató nevét, Neptun kódját, születési dátumát, lakcímét, telefonszámát és email címét. A lakcímből még leágazik a város, utca, illetve a házszám. Minden hallgatónak egyedi azonosítója van, Hid néven. A hallgató entitás kapcsolatban áll a beiratkozással.
A kurzus tartalmazza a kurzus kódját, a nevét, hogy hány kreditet ér és a kurzus típusát. Minden kurzusnak van egy egyedi azonosítója, ennek a neve Kid, valamint kapcsolatban áll az oktató és a beiratkozás entitásokkal.
A következő entitásunk az oktató, amiből leágazik az oktató neve, titulusa, beosztása és email címe. Egyedi azonosítója az Oid, és kapcsolatban áll a kurzussal és az irodával.
Az oktatót a tanszék követi, ez tartalmazza a tanszék nevét, kódját, az épületet ahol elhelyezkedik, valamint a telefonszámát. Egyedi azonosítója a Tid.
Az iroda entitás tartalmazza az iroda épületszámát, a konkrét szobát, az épület nevét, valamint egy megjegyzést. Egyedi azonosítója az Iid, kapcsolatban áll az oktató entitással.
Végül a beiratkozáshoz tartozik a beiratkozás féléve, a kapott osztályzat, hogy teljesítve lett-e az adott kurzus, valamint a jelölés. Azonosítója a Bid, kapcsolatban áll a hallgatóval és az oktatóval.



2. kép: Az XDM model

1.3 Az XDM modell alapján XML dokumentum készítése

Az XML dokumentum célja az XDM modellben meghatározott adatszerkezet leírása. A fájl a hallgatók, oktatók, kurzusok, tanszékek, irodák és beiratkozások adatait tartalmazza strukturált, hierarchikus formában. A dokumentum gyökéreleme a `<NeptunAdatok>`, ami az összes entitást magába foglalja.

A hallgatók adatai a `<Hallgato>` elemekben jelennek meg, amik attribútumként egyedi azonosítót (`hid`) tartalmaznak. Minden hallgatóhoz meg vannak adva az alapvető személyes adatok: neptunkód, név, születési dátum, valamint egy összetett cím elem, ami a várost, utcát és házszámot tartalmazza. A hallgatóknál többértékű tulajdonságok is vannak, például több email címet és telefonszámot, ami az ismétlődő elemek segítségével valósul meg.

A kurzusokat a `<Kurzus>` elemek írják le, amik tartalmazzák a kurzus kódját, nevét, kreditértékét és típusát. Minden kurzushoz egy idegen kulcs (`TanarRef`) kapcsolódik, amely az adott kurzust oktató tanárra hivatkozik. Az oktatók adatai az `<Oktato>` elemekben vannak, ahol az azonosító (`oid`) mellett szerepel a név, titulus, beosztás és email cím. Az oktatókhoz 1:1 kapcsolatban tartozik egy iroda, amelyre az `IrodaRef` elem hivatkozik.

A tanszékek adatait a `<Tanszek>` elemek tárolják, amik megadják a tanszék nevét, kódját, épületét és telefonszámát. Az oktatók és tanszékek közötti kapcsolat idegen kulcsokon keresztül értelmezett. Az irodák külön `<Iroda>` elemekben szerepelnek, amelyek tartalmazzák az épületszámot, szobaszámot, épületnevet és egy opcionális megjegyzést.

A hallgatók és kurzusok közötti N:M kapcsolatot a `<Beiratkozas>` elemek valósítják meg, amelyek attribútumként hivatkoznak mind a hallgatóra (`hallgatoRef`), mind pedig a kurzusra (`kurzusRef`). A beiratkozásokhoz további információk is tartoznak, mint a félév, osztályzat, teljesítés státusza és jelölés.

Hallgató példakód:

```
<Hallgato hid="H2">
  <Neptun>MBXY34</Neptun>
  <Nev>Kovács Balázs</Nev>
  <SzuletesiDatum>1999-11-02</SzuletesiDatum>
  <Cim>
    <Varos>Budapest</Varos>
    <Utca>Fővám tér</Utca>
    <Hazszam>8</Hazszam>
  </Cim>
  <Email>balazs.kovacs@gmail.com</Email>
```

```
  <Telefonszam>+36-70-111-2222</Telefonszam>
</Hallgato>
```

Kurzus példakód:

```
<Kurzus kid="K2">
  <Kod>JAVA2</Kod>
  <Nev>Haladó Java</Nev>
  <Kredit>6</Kredit>
  <Tipus>Gyakorlat</Tipus>
  <TanarRef>TA2</TanarRef>
</Kurzus>
```

Oktató példakód:

```
<Oktato oid="TA2">
  <Nev>Prof. Nagy Erzsébet</Nev>
  <Titulus>egyetemi tanár</Titulus>
  <Beosztas>Adatbázisok</Beosztas>
  <Email>erzsebet.nagy@uni.hu</Email>
  <IrodaRef>I11</IrodaRef>
</Oktato>
```

Tanszék példakód:

```
<Tanszek tid="D2">
  <Nev>Matematika Tanszék</Nev>
  <Kod>MTA</Kod>
  <Epulet>B épület</Epulet>
  <Telefon>+36-52-444-100</Telefon>
</Tanszek>
```

Iroda példakód:

```
<Iroda iid="I11">
  <Epuletszam>B2</Epuletszam>
  <Szoba>101</Szoba>
  <EpuletNev>Matematika épület</EpuletNev>
  <Megjegyzes>Közös használatú iroda</Megjegyzes>
</Iroda>
```

Beiratkozás példakód:

```
<Beiratkozas bid="B2" hallgatoRef="H2" kurzusRef="K2">
  <Felev>2024/2</Felev>
  <Osztalyzat>5</Osztalyzat>
  <Teljesitve>Igen</Teljesitve>
  <Jeloles>Gyakorlat teljesítve</Jeloles>
</Beiratkozas>
```

1.4 Az XML dokumentum alapján XML schema készítése

Az XML séma célja az XML dokumentumban megadott adatszerkezet formális leírása, amely biztosítja az adatok szerkezeti és típusbeli helyességét. A séma a `<NeptunAdatok>` gyökérelemhez készült, és meghatározza az összes benne szereplő entitás (Hallgató, Kurzus, Oktató, Tanszék, Iroda, Beiratkozás) szerkezetét, azok attribútumait, elemeit, valamint az entitások közötti kapcsolatok érvényességét.

Hallgató elem:

A `Hallgato` elem egy összetett típusú (`complexType`) szerkezet, amely a hallgatók adatait tárolja.
A `maxOccurs="unbounded"` attribútum jelzi, hogy a dokumentumban több hallgató is szerepelhet.

A belső elemek:

- `<Neptun>` – a hallgató egyedi Neptun-kódja (`xs:string`).
- `<Nev>` – a hallgató teljes neve.
- `<SzuletesiDatum>` – a hallgató születési dátuma (`xs:date` típusban).
- `<Cim>` – összetett elem, amely a hallgató lakcímét tartalmazza:
  - `<Varos>` – a lakóhely városa
  - `<Utca>` – az utca neve
  - `<Hazszam>` – a házszám
- `<Email>` – többször megadható e-mail cím (`maxOccurs="unbounded"`), mivel egy hallgatónak több elérhetősége is lehet.
- `<Telefonszam>` – többször előforduló telefonszám mező.

Az elem attribútuma:

- `hid` – kötelező (`use="required"`) azonosító (`xs:ID`), amely egyedileg azonosítja a hallgatót.

Kód:

```
<xs:element name="Hallgato" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Neptun" type="xs:string"/>
      <xs:element name="Nev" type="xs:string"/>
      <xs:element name="SzuletesiDatum" type="xs:date"/>
      <xs:element name="Cim">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Varos" type="xs:string"/>
            <xs:element name="Utca" type="xs:string"/>
            <xs:element name="Hazszam" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Email" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="Telefonszam" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="hid" type="xs:ID" use="required"/>
  </xs:complexType>
</xs:element>
```

Kurzus elem:

A `Kurzus` elem az egyes egyetemi kurzusok adatait tárolja. A `maxOccurs="unbounded"` attribútum lehetővé teszi, hogy több kurzus is szerepeljen az XML dokumentumban.

A belső elemek:

- `<Kod>` – a kurzus egyedi kódja (`xs:string`).
- `<Nev>` – a kurzus neve (pl. „Haladó Java”).
- `<Kredit>` – a kurzushoz tartozó kreditérték (`xs:positiveInteger`).
- `<Tipus>` – a kurzus típusa, például „Előadás” vagy „Gyakorlat”.

- `<TanarRef>` – idegen kulcs (`xs:IDREF`), amely hivatkozik arra az oktatóra, aki a kurzust tartja.

Attribútum:

- `kid` – kötelező (`xs:ID`) azonosító, amely egyedileg azonosítja a kurzust.

Kód:

```
<xs:element name="Kurzus" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Kod" type="xs:string"/>
      <xs:element name="Nev" type="xs:string"/>
      <xs:element name="Kredit" type="xs:positiveInteger"/>
      <xs:element name="Tipus" type="xs:string"/>
      <xs:element name="TanarRef" type="xs:IDREF"/>
    </xs:sequence>
    <xs:attribute name="kid" type="xs:ID" use="required"/>
  </xs:complexType>
</xs:element>
```

Oktató elem:

Az `Oktato` elem az egyetemi oktatók adatait írja le. Többször előfordulhat a dokumentumban (`maxOccurs="unbounded"`), így több oktató is megadható.

A belső elemek:

- `<Nev>` – az oktató teljes neve.
- `<Titulus>` – az oktató tudományos fokozata vagy beosztása (pl. „egyetemi tanár”).
- `<Beosztas>` – az oktató szakmai szerepköre vagy tanszéki beosztása.
- `<Email>` – az oktató hivatalos e-mail címe.
- `<IrodaRef>` – idegen kulcs (`xs:IDREF`), amely az oktatóhoz tartozó iroda azonosítójára mutat.

Attribútum:

- `oid` – kötelező (`xs:ID`) azonosító, amely minden oktatót egyedileg jelöl.

Kód:

```
<xs:element name="Oktato" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Nev" type="xs:string"/>
      <xs:element name="Titulus" type="xs:string"/>
      <xs:element name="Beosztas" type="xs:string"/>
      <xs:element name="Email" type="xs:string"/>
      <xs:element name="IrodaRef" type="xs:IDREF"/>
    </xs:sequence>
    <xs:attribute name="oid" type="xs:ID" use="required"/>
  </xs:complexType>
</xs:element>
```

Tanszék elem:

A `Tanszek` elem az egyetemi tanszékek adatait tartalmazza. A `maxOccurs="unbounded"` érték miatt több tanszék is leírható egy XML dokumentumban.

A belső elemek:

- `<Nev>` – a tanszék neve (pl. „Informatikai Tanszék”).
- `<Kod>` – a tanszék azonosító kódja (`xs:string`).
- `<Epulet>` – az épület megnevezése, ahol a tanszék található.
- `<Telefon>` – a tanszék központi telefonszáma.

Attribútum:

- `tid` – kötelező (`xs:ID`) attribútum, amely a tanszéket egyedileg azonosítja.

Kód:

```
<xs:element name="Tanszek" maxOccurs="unbounded">
  <xs:complexType>
```

```
    <xs:sequence>
      <xs:element name="Nev" type="xs:string"/>
      <xs:element name="Kod" type="xs:string"/>
      <xs:element name="Epulet" type="xs:string"/>
      <xs:element name="Telefon" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="tid" type="xs:ID" use="required"/>
  </xs:complexType>
</xs:element>
```

Iroda elem:

Az `Iroda` elem az oktatókhoz kapcsolódó irodák adatait írja le. Több iroda is szerepelhet az XML-ben (`maxOccurs="unbounded"`).

A belső elemek:

- `<Epuletszam>` – az épület száma vagy kódja.
- `<Szoba>` – az iroda szobaszáma.
- `<EpuletNev>` – az épület neve, ahol az iroda található.
- `<Megjegyzes>` – opcionális (`minOccurs="0"`) leírás vagy megjegyzés az irodáról (pl. „Közös használatú iroda”).

Attribútum:

- `iid` – kötelező (`xs:ID`) azonosító, amely minden irodát egyedileg azonosít.

Kód:

```
<xs:element name="Iroda" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Epuletszam" type="xs:string"/>
      <xs:element name="Szoba" type="xs:string"/>
      <xs:element name="EpuletNev" type="xs:string"/>
      <xs:element name="Megjegyzes" type="xs:string" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="iid" type="xs:ID" use="required"/>
  </xs:complexType>
</xs:element>
```

Beiratkozás elem:

A `Beiratkozas` elem a hallgatók és kurzusok közötti kapcsolatot (N:M kapcsolat) írja le.
Minden beiratkozás egy konkrét hallgatóhoz és kurzushoz tartozik.

A belső elemek:

- `<Felev>` – a tanulmányi félév azonosítója (pl. „2024/2”).
- `<Osztalyzat>` – a hallgató által elért érdemjegy (`xs:integer`).
- `<Teljesitve>` – a kurzus teljesítésének státusza („Igen” / „Nem”).
- `<Jeloles>` – szöveges megjegyzés vagy státuszjelzés (pl. „Gyakorlat teljesítve”).

Attribútumok:

- `bid` – kötelező (`xs:ID`) azonosító, amely a beiratkozást egyedileg azonosítja.
- `hallgatoRef` – idegen kulcs (`xs:IDREF`), amely a `Hallgato` elem `hid` attribútumára hivatkozik.
- `kurzusRef` – idegen kulcs (`xs:IDREF`), amely a `Kurzus` elem `kid` attribútumára hivatkozik.

Kód:

```
<xs:element name="Beiratkozas" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Felev" type="xs:string"/>
      <xs:element name="Osztalyzat" type="xs:integer"/>
      <xs:element name="Teljesitve" type="xs:string"/>
      <xs:element name="Jeloles" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="bid" type="xs:ID" use="required"/>
    <xs:attribute name="hallgatoRef" type="xs:IDREF" use="required"/>
    <xs:attribute name="kurzusRef" type="xs:IDREF" use="required"/>
  </xs:complexType>
</xs:element>
```

2. DOM feladat

Project name: F9PSJADomParse

Package: hu.domparse.f9psja

Class names: (F9PSJADomRead, F9PSJADomQuery, F9PSJADomModify)

2.1 Adatolvasás

A program felépítése és működése

A fő osztály:

```
public class F9PSJADOMRead {
    public static void main(String[] args) {
        // XML fájl beolvasása DOM objektummá
        File xmlFile = new File("F9PSJA_XML.xml");
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse(xmlFile);
        doc.getDocumentElement().normalize();
```

Magyarázat:

A `DocumentBuilderFactory` és `DocumentBuilder` osztályok segítségével történik az XML dokumentum beolvasása. A `normalize()` metódus gondoskodik arról, hogy az XML-ben lévő whitespace karakterek egységesen legyenek kezelve.

Fő feldolgozó logika:

A gyökérelem kiírása és az elemek bejárása:

```
String rootName = doc.getDocumentElement().getNodeName();
printlnBoth("Gyökérelem: " + rootName, writer);

NodeList topChildren = doc.getDocumentElement().getChildNodes();
for (int i = 0; i < topChildren.getLength(); i++) {
    Node n = topChildren.item(i);
    if (n.getNodeType() != Node.ELEMENT_NODE) continue;
    Element e = (Element) n;
    processElement(e, writer, 0);
}
```

Magyarázat:

A gyökérelem (`NeptunAdatok`) azonosítása után a program végigiterál a gyermek

elemeken. Minden egyes elem feldolgozását a `processElement()` metódus végzi.

Elemfeldolgozás:

```
private static void processElement(Element elem, PrintWriter
writer, int indent) {
    String indentStr = " ".repeat(indent);
    NamedNodeMap attrs = elem.getAttributes();
    if (attrs.getLength() > 0) {
        // Attribútumok kiírása
        StringBuilder sb = new StringBuilder();
        sb.append(indentStr).append("Attribútumok: ");
        for (int i = 0; i < attrs.getLength(); i++) {
            Node a = attrs.item(i);

sb.append(a.getNodeName()).append("=").append(a.getNodeValue());
            if (i < attrs.getLength() - 1) sb.append(", ");
        }
        printlnBoth(sb.toString(), writer);
    }
```

Magyarázat:

Ez a metódus rekurzívan bejárja az XML-fa elemeit. Az attribútumok kiírását kezeli, majd a gyermek elemeket külön csoportosítja és újra meghívja önmagát, így biztosítva a hierarchikus szerkezet megjelenítését.

Segédmetódusok:

A kód tartalmaz több kisegítő metódust:

- `printlnBoth()` – egyszerre ír ki a konzolra és a fájlba.
- `hasOnlyText()` – eldönti, hogy egy elem csak szöveget tartalmaz-e.
- `processGroup()` – azonos típusú elemek (pl. több `Hallgato`) blokkosított kiírását végzi.

Ezek a metódusok segítik az XML átlátható, strukturált megjelenítését.

Kimenet formája:

A program a teljes XML tartalmát a konzolra és a `F9PSJA_DOMRead_output.txt` fájlba írja ki, a következő formában:

=== HALLGATO (összesen: 3) ===
--- Hallgato #1 ---
Attribútumok: hid=H1
Nev: Kiss Anna
SzuletesiDatum: 2000-04-12
Email: anna.kiss@example.com
Telefonszam: +36201234567

Tervezés és megvalósítás összefoglalása

A program moduláris felépítésű:

- A DOM modell lehetővé teszi az XML struktúra teljes bejárását.
- A rekurzív feldolgozás biztosítja, hogy minden szintű elem és attribútum megjelenjen.
- A blokkos kiírás formázott, emberileg olvasható kimenetet biztosít.
- A megjegyzések segítik a kód áttekinthetőségét és a logikai egységek elkülönítését.

Link

2.2 Adat-lekérdezés

A program működése

A program elején az XML dokumentum beolvasása és normalizálása történik:

```
Document doc = DocumentBuilderFactory.newInstance()
        .newDocumentBuilder()
        .parse(new File("F9PSJA_XML.xml"));
doc.getDocumentElement().normalize();
System.out.println("===== LEKÉRDEZÉSEK =====");
```

Magyarázat:

A `DocumentBuilderFactory` és `DocumentBuilder` osztályok használatával a program betölti az XML fájlt memóriába DOM struktúraként, majd a

`normalize()` segítségével egységesíti a szöveges csomópontokat. Ezután jöhetnek a lekérdezések.

1. Lekérdezés – Minden hallgató neve:

```
NodeList hallgatok = doc.getElementsByTagName("Hallgato");
System.out.println("1. Minden hallgató neve:");
for (int i = 0; i < hallgatok.getLength(); i++) {
    Element h = (Element) hallgatok.item(i);
    System.out.println("- " +
h.getElementsByTagName("Nev").item(0).getTextContent());
}
```

Magyarázat:

A `getElementsByTagName("Hallgato")` metódus az összes `<Hallgato>` elemet kiválasztja.

Ezután a `Nev` al-elemből kiolvassa a szöveget, és a konzolra írja a hallgatók nevét.

2. Lekérdezés – 5 kredites kurzusok:

```
NodeList kurzusok = doc.getElementsByTagName("Kurzus");
System.out.println("\n2. 5 kredites kurzusok:");
for (int i = 0; i < kurzusok.getLength(); i++) {
    Element k = (Element) kurzusok.item(i);
    if
(Integer.parseInt(k.getElementsByTagName("Kredit").item(0).getText
Content()) == 5) {
        System.out.println("- " +
k.getElementsByTagName("Nev").item(0).getTextContent());
    }
}
```

Magyarázat:

A `Kurzus` elemek közül csak azokat írja ki, ahol a `Kredit` értéke 5.

A feltételes ellenőrzés a `parseInt()` segítségével történik.

3. Lekérdezés – Oktatók email címei:

```
NodeList oktatok = doc.getElementsByTagName("Oktato");
System.out.println("\n3. Oktatók email címei:");
for (int i = 0; i < oktatok.getLength(); i++) {
    Element o = (Element) oktatok.item(i);
```

```
System.out.println(o.getElementsByTagName("Nev").item(0).getTextCo
ntent() + ": "
            +
o.getElementsByTagName("Email").item(0).getTextContent());
}
```

Magyarázat:

A lekérdezés minden oktató nevét és e-mail címét listázza ki.

A DOM lehetőséget ad arra, hogy könnyen hozzáférjünk a beágyazott elemekhez (`Nev`, `Email`).

4. Lekérdezés – Sikertelen teljesítések:

```
NodeList beiratkozasok = doc.getElementsByTagName("Beiratkozas");
System.out.println("\n4. Sikertelen teljesítések:");
for (int i = 0; i < beiratkozasok.getLength(); i++) {
    Element b = (Element) beiratkozasok.item(i);
    if
(b.getElementsByTagName("Teljesitve").item(0).getTextContent().equ
als("Nem")) {
        System.out.println("- " + b.getAttribute("bid"));
    }
}
```

Magyarázat:

Ez a lekérdezés csak azokat a beiratkozásokat listázza, ahol a `Teljesitve` elem értéke „Nem”.

A program a beiratkozás azonosítóját (`bid` attribútum) jeleníti meg.

Kimenet mintája

```
===== LEKÉRDEZÉSEK =====
1. Minden hallgató neve:
- Kiss Anna
- Tóth Péter
- Nagy Eszter

2. 5 kredites kurzusok:
- Programozás I
- Adatbázisok

3. Oktatók email címei:
Dr. Kovács Béla: bela.kovacs@uni.hu
Dr. Horváth Anna: anna.horvath@uni.hu
```

```
4. Sikertelen teljesítések:
- B3
```

Tervezés és megvalósítás összefoglalása

A `NeptunkodDOMQuery.java` program a DOM-modell használatával egyszerű, de hatékony módon hajt végre strukturált lekérdezéseket XML dokumentumon:

- Az XML adatokat teljesen beolvassa memóriába.
- Egyszerű metódusokkal (`getElementsByTagName`) lekérdezi az adott elemeket.
- Feltételek alapján szűri és jeleníti meg az eredményt.
- Könnyen bővíthető további lekérdezésekkel.

Link

2.3 Adatmódosítás

A program működése

A program célja az `F9PSJA_XML.xml` állomány módosítása DOM (Document Object Model) segítségével, majd az új, frissített dokumentum mentése `F9PSJA_XML_modified.xml` néven.

XML beolvasása és normalizálása:

```
File xmlFile = new File("F9PSJA_XML.xml");
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(xmlFile);
doc.getDocumentElement().normalize();
```

Magyarázat:
A program beolvassa az `F9PSJA_XML.xml` fájlt DOM struktúraként a memóriába. A `normalize()` metódus egységesíti a szöveges csomópontokat és eltávolítja a fölösleges whitespace karaktereket, így biztosítva a későbbi módosítások helyes működését.

1. Módosítás – Első hallgató nevének megváltoztatása:

```
Element firstHallgato = (Element)
doc.getElementsByTagName("Hallgato").item(0);
String oldName =
firstHallgato.getElementsByTagName("Nev").item(0).getTextContent()
;
firstHallgato.getElementsByTagName("Nev").item(0).setTextContent("
Horváth Anna Mária");
System.out.println("Hallgató neve módosítva: " + oldName + " →
Horváth Anna Mária");
```

Magyarázat:
A `getElementsByTagName("Hallgato")` az összes hallgatót listázza, majd az elsőt (`item(0)`) kiválasztja.
A `Nev` al-elemet eléri, és a szöveges tartalmát módosítja egy új névre.
A konzolon kiírja a régi és új értéket.

2. Módosítás – Új kurzus hozzáadása:

```
Node lastKurzus = doc.getElementsByTagName("Kurzus")
        .item(doc.getElementsByTagName("Kurzus").getLength() - 1);

Element newKurzus = doc.createElement("Kurzus");
newKurzus.setAttribute("kid", "K3");
```

```
Element kod = doc.createElement("Kod");
kod.setTextContent("WEB3");
Element nev = doc.createElement("Nev");
nev.setTextContent("Webprogramozás alapjai");
Element kredit = doc.createElement("Kredit");
kredit.setTextContent("4");
Element tipus = doc.createElement("Tipus");
tipus.setTextContent("Előadás");
Element tanarRef = doc.createElement("TanarRef");
tanarRef.setTextContent("TA1");

newKurzus.appendChild(kod);
newKurzus.appendChild(nev);
newKurzus.appendChild(kredit);
newKurzus.appendChild(tipus);
newKurzus.appendChild(tanarRef);

root.insertBefore(newKurzus, lastKurzus.getNextSibling());
System.out.println("Új kurzus hozzáadva: WEB3");
```

Magyarázat:
Egy új `<Kurzus>` elem jön létre `K3` azonosítóval.
Az elemhez al-elemeket hozunk létre (Kod, Nev, Kredit, Tipus, TanarRef), majd beszúrjuk a dokumentum végére a többi kurzus után.
A DOM segítségével a program teljes értékű új elemet tud generálni és beilleszteni.

3. Módosítás – Első beiratkozás törlése:

```
Node firstBeiratkozas =
doc.getElementsByTagName("Beiratkozas").item(0);
root.removeChild(firstBeiratkozas);
System.out.println("Első beiratkozás törölve.");
```

Magyarázat:
A program kiválasztja az első `<Beiratkozas>` elemet, majd a gyökérelemből (`root`) eltávolítja azt.
Ez a DOM `removeChild()` metódusával történik, amely az adott elem és annak teljes alstruktúráját törli.

4. Módosítás – Oktató e-mail címének frissítése:

```
Element oktato = (Element)
doc.getElementsByTagName("Oktato").item(0);
Node emailNode = oktato.getElementsByTagName("Email").item(0);
String oldEmail = emailNode.getTextContent();
emailNode.setTextContent("szabo.gabor@inf.unideb.hu");
System.out.println("Oktató e-mail módosítva: " + oldEmail + " →
szabo.gabor@inf.unideb.hu");
```

Magyarázat:
Az első oktató `<Email>` elemét eléri és a szöveges értékét módosítja. Így az XML-ben az oktató e-mail címe frissül az új címre.

5. Módosítások mentése új fájlba:

```
Transformer transformer =
TransformerFactory.newInstance().newTransformer();
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-
amount", "2");
transformer.transform(new DOMSource(doc), new StreamResult(new
File("F9PSJA_XML_modified.xml")));
```

Magyarázat:
A `Transformer` segítségével a módosított DOM dokumentumot visszaírja fájlba. Az `INDENT` beállítás gondoskodik az olvasható, tagolt formátumról. A végeredmény `F9PSJA_XML_modified.xml` néven jön létre.

Kimenet mintája:

```
Hallgató neve módosítva: Kiss Anna → Horváth Anna Mária
Új kurzus hozzáadva: WEB3
Első beiratkozás törölve.
Oktató e-mail módosítva: kovacs.bela@uni.hu →
szabo.gabor@inf.unideb.hu
```

Tervezés és megvalósítás összefoglalása

A F9PSJADOMModify.java program a DOM-modell módosítási lehetőségeit demonstrálja.
A program:

- Beolvassa az XML dokumentumot a memóriába.
- Elem- és attribútumszinten képes adatokat módosítani.
- Új elemeket hoz létre és illeszt be.
- Meglévő elemeket töröl.
- Az eredményt új, jól formázott XML fájlba menti.

A megoldás jól szemlélteti a DOM API gyakorlati alkalmazását XML dokumentumok szerkesztésére és karbantartására.

Link