JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

Ételrendelések

Készítette: **Uri Alexandra Nikoletta**

Neptunkód: **OE8BW9**

Dátum: **2025. december**

Miskolc, 2025

Tartalomjegyzék

Bevezetés

A jelen jegyzőkönyv célja egy, az ételrendelések kapcsolódó, XML-alapú adatrendszer megtervezése, valamint az ehhez szükséges adatmodellek elkészítése ER és XDM formában. A dokumentum bemutatja a modellek alapján létrehozott XML állományt és a hozzá tartozó XML Schema leírást is. A második rész a DOM szabvány szerinti feldolgozási műveletekre fókuszál, amelyek Java nyelven valósulnak meg: az adatbeolvasásra, a lekérdezések végrehajtására és az adatmódosításokra.

Az ER modell a fogalmi tervezési szintet képviseli: rögzíti az egyedeket, azok kapcsolatait, valamint az egyedek attribútumait. A koncepcionális tervből az XDM irányába haladva a rendszer hierarchikus, fastruktúrájú leírása készül el, amely meghatározza a gyökérelemet, az elemek egymásra hivatkozásait és az ismétlődő struktúrákat. Az XML dokumentum a modellt valós adatokkal szemlélteti, míg az XSD séma a szerkezetet, adattípusokat és integritási szabályokat formálisan rögzíti.

A DOM-alapú feldolgozás célja, hogy az OE8BW9_XML.xml tartalma programozottan is kezelhető legyen. Ennek fő lépései:

- **olvasás**: a teljes dokumentum strukturált, blokkos megjelenítése,
- **lekérdezés**: legalább négy releváns információ kigyűjtése,
- **módosítás**: legalább négy, az adatintegritást megtartó változtatás végrehajtása, a módosított elemek blokkos kiírásával.

A dokumentum tartalmazza a feladat által előírt formai követelményeket is: megfelelő címszintek, sorkizárt szövegezés, a képek számozása és hivatkozása, valamint a kódrészletek elkülönített betűtípussal történő formázása. A beillesztett ábrák szabványos jelölésrendszerrel, számozással és hivatkozásokkal szerepelnek.

1.Forrás fileok

Forgalmi és logikai tervezés, majd XML és XSD létrehozása

A feladat leírása

A feladat célja, hogy egy hétköznapi étteremhez bejövő, telefonos hívásokon keresztül leadott rendeléseket kezelje.
A rendszer rögzíti a rendelés leadásának időpontját, a vevő által megadott kiszállítási címet, valamint a rendelt ételek teljes, fizetendő összegét.

Az adatbázis tartalmazza mind a megrendelhető, mind a már megrendelt ételeket. Az ételek adatai között szerepel a nevük, áruk, típusuk (például: pizzák, húsételek, saláták stb.), valamint az esetleges allergének, amelyeket tartalmazhatnak az ételek.

A rendelés és az ételek kapcsolatának áttekinthetősége érdekében az adatbázisban feltüntetjük, ha egy adott ételből több darabot is rendeltek.

Tekintettel arra, hogy bizonyos időszakokban egyes ételek az étterem kínálatában akciós áron kaphatók, hogy vonzóbbá tegyék az ajánlatot a vásárlók számára. Az ilyen kedvezmények esetén eltároljuk az akció kezdetének és végének időpontját, valamint a leárazás mértékét százalékban megadva.

Az adatbázisban nyilvántartjuk a megrendelést leadó vendéget is, mint megrendelőt. Itt elérhetők az ügyfél adatait, a nevét és a telefonszáma, valamint információ arról, hogy korábban vásárolt-e már az étteremnél.

A sikeres kiszállítást követően a fizetés adatai is bekerülnek az adatbázisba. Ide tartozik az átvétel időpontja, a fizetés módja (bankkártya, készpénz vagy egyéb fizetési forma), valamint annak rögzítése, hogy a vevő használt-e az étterem által kiadott kupont a fizetés során.

1.Feladat

1.1 Az Adatbázis ER modell tervezése

A feladat egy ételrendelési rendszer fogalmi modelljének megalkotása: rendelések, megrendelők, fizetések, ételek, akciók, valamint a rendelési és akció-adatok egységes kezelésével. A modellnek támogatnia kell a több-több kapcsolatot (pl. egy rendelésben több étel, egy ételhez több akció tartozhat), és biztosítania kell az adatintegritást elsődleges és idegen kulcsokkal.

Egyedek és tulajdonságok:

• Rendelés

- RID: elsődleges kulcs
- MID: idegen kulcs a megrendelőre mutat
- FID: idegen kulcs a fizetésre mutat
- Dátum: a rendelés dátuma
- Kiszállítási_cím: cím string formátumban
- Összeg: teljes rendelési érték

• Megrendelő

- MID: elsődleges kulcs
- Visszatérő_vendég: enumerált érték (igen/nem)
- Telefonszám: string
- Név: a megrendelő neve
- Fizetés
- FID: elsődleges kulcs
- Átvitel_ideje: tranzakció dátuma
- Kupon: enumerált érték (igen/nem)
- Fizetés_módja: enumerált érték (bankkártya/készpénz/szép-kártya)

• Rendelt (kapcsoló egyed Rendelés és Étel között)

- RID: idegen kulcs a rendelésre mutat
- EID: idegen kulcs az ételre mutat
- Mennyiség: rendelt mennyiség
- Tétel_ár: egységár
- Megjegyzés: opcionális szöveges megjegyzés

• Étel

- EID: elsődleges kulcs
- Név: az étel neve

- Ár: alapár
- Allergén: többértékű tulajdonság (glutén, tej, stb.)
- Típus: kategória (pl. pizza, saláta)

• Akció

- AID: elsődleges kulcs
- EID: idegen kulcs az ételre mutat
- Időtartam: összetett tulajdonság
  - Kezdete: akció kezdő dátuma
  - Vége: akció záró dátuma
- Leárazás: százalékos érték

Kapcsolatok:

- **Megrendelő – Rendelés: 1:N**

  Egy megrendelő több rendelést leadhat, de egy rendelés pontosan egy megrendelőhöz tartozik.

- **Rendelés – Fizetés: 1:1**

  Egy rendeléshez egy fizetési tranzakció társul, és egy fizetés csak egy rendelésre vonatkozik.

- **Rendelés – Étel – Rendelt: M:N (Rendelt kapcsoló egyed)**

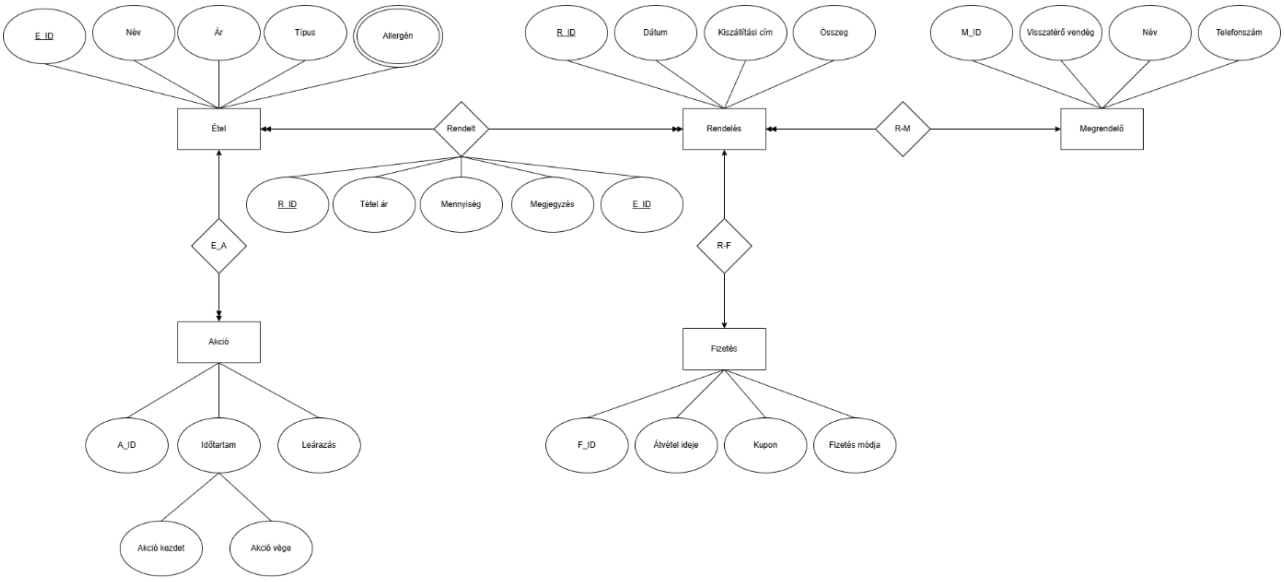
  Egy rendelésben több étel szerepelhet különböző mennyiségekben/árakkal; egy étel több rendelésben is megjelenhet.

- **Étel – Akció: 1:N**

  Egy ételhez több akció is tartozhat különböző időtartamokban/leárazásokkal; egy akció csak egy konkrét ételre vonatkozik.

A kapcsolatok biztosítása:

- Minden kapcsolat elsődleges és idegen kulcsokkal van implementálva
- A kapcsoló entitások (Rendelt) további attribútumokat tartalmaznak (mennyiség, ár, megjegyzés)
- Az 1:1 kapcsolat unique megszorítással van biztosítva
- A többértékű attribútumok (allergének) a séma struktúrájában vannak kezelve



ER modell

1.2 Az adatbázis konvertálása XDM modellre

XDM modell felépítése a megadott jelölésekkel

A XDM modell az XML dokumentum szerkezetét reprezentálja a következő jelölésekkel:

- **Ellipszis**: Elemek (minden egyedből és tulajdonságból, kivéve kulcsok)
- **Rombusz**: Attribútumok (kulcs tulajdonságokból)
- **Téglalap**: Szöveges tartalom (XML dokumentumban megjelenő értékek)
- **Dupla szürke ellipszis**: Többször előforduló elemek
- **Szaggatott vonal**: Idegen kulcs kapcsolatok

XDM modell elemek és attribútumok:

Gyökérelem: ételrendelések (téglalap)

1. Rendelés (dupla szürke ellipszis - többször előfordul)

   - RID (rombusz) - elsődleges kulcs
   - MID (rombusz) - idegen kulcs → megrendelő
   - FID (rombusz) - idegen kulcs → fizetés
   - dátum (ellipszis) → téglalap: dátum érték
   - kiszállítási_cím (ellipszis) → téglalap: cím szöveg
   - összeg (ellipszis) → téglalap: számérték

2. Megrendelő (dupla szürke ellipszis - többször előfordulhat)

   - MID (rombusz) - elsődleges kulcs
   - visszatérő_vendég (ellipszis) → téglalap: "igen"/"nem"
   - telefonszám (ellipszis) → téglalap: telefonszám
   - név (ellipszis) → téglalap: név szöveg

3. Fizetés (dupla szürke ellipszis - többször előfordulhat)

- FID (rombusz) - elsődleges kulcs
- átvitel_ideje (ellipszis) → téglalap: dátum érték
- kupon (ellipszis) → téglalap: "igen"/"nem"
- fizetés_módja (ellipszis) → téglalap: "bankkártya"/"készpénz"/"szép-kártya"

4. Rendelt (dupla szürke ellipszis - többször előfordulhat, kapcsoló elem)

- RID (rombusz) - idegen kulcs → rendelés
- EID (rombusz) - idegen kulcs → étel
- mennyiség (ellipszis) → téglalap: számérték
- tétel_ár (ellipszis) → téglalap: számérték
- megjegyzés (ellipszis) → téglalap: szöveges megjegyzés (opcionális)

5. Étel (dupla szürke ellipszis - többször előfordulhat)

- EID (rombusz) - elsődleges kulcs
- név (ellipszis) → téglalap: étel neve
- ár (ellipszis) → téglalap: alapár
- allergén (dupla szürke ellipszis) → téglalap: allergén neve (többértékű)
- típus (ellipszis) → téglalap: étel típusa

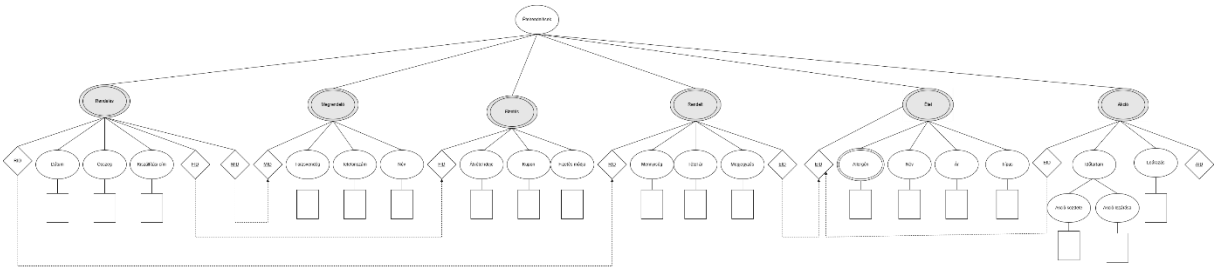
6. Akció (dupla szürke ellipszis - többször előfordulhat)

- AID (rombusz) - elsődleges kulcs
- EID (rombusz) - idegen kulcs → étel
- időtartam (ellipszis) - összetett elem
- kezdete (ellipszis) → téglalap: dátum érték
- vége (ellipszis) → téglalap: dátum érték
- leárazás (ellipszis) → téglalap: százalékos érték

Kapcsolatok szaggatott vonallal:

- rendelés MID → megrendelő MID
- rendelés FID → fizetés FID
- rendelt RID → rendelés RID
- rendelt EID → étel EID
- akció EID → étel EID

A modell biztosítja az adatintegritást és követi a séma definíciót, miközben vizuálisan reprezentálja az XML dokumentum hierarchikus szerkezetét.



XDM*modell*

1.3 Az XDM modell alapján XML dokumentum készítése

Az XDM modell (amely az XML dokumentumokat csomópontfák hierarchiájaként kezeli) alapján az XML dokumentumot a `ételrendelések` gyökérelem létrehozásával kezdtem. A dokumentum célja egy ételrendelési rendszer adatainak tárolása.

Az adatmodell az adatbázis-tervezés logikáját követi: minden fő entitáshoz (`rendelés`, `megrendelő`, `fizetés`, `étel`, `akció`) külön szekciót hoztam létre a gyökér alatt. Az azonosítók és hivatkozások kezelésére az attribútumokat (`RID`, `MID`, `FID`, `EID`, `AID`) használtam elsődleges és idegen kulcsokként.

Az XDM hierarchia szemléltetése érdekében minden többszörös előfordulású főkategóriából (pl. `rendelés`, `étel`) legalább négy-négy példányt készítettem. A kapcsolótáblát reprezentáló `rendelt` elemekben bemutattam a több a többhöz kapcsolatot (pl. egy rendeléshez több tétel tartozik). Végül, ahol többértékű tulajdonság szerepelt (pl. egy ételhez több `allergén` tartozhat), legalább két előfordulást adtam meg. Az összetett tulajdonságokat (pl. `akció` elemen belüli `időtartam`) további al-elemekkel (`kezdete`, `vége`) bontottam ki.

XML-részlet:

```
<ételrendelések><!—root element→
  <!-- rendelések -->
  <rendelés RID="1" MID="1" FID="1">
    <dátum>2025-01-01</dátum>
    <kiszállítási_cím>Arany János utca 1.</kiszállítási_cím>
    <összeg>6750</összeg>
  </rendelés>

  <rendelés RID="2" MID="2" FID="2">
    <dátum>2025-02-02</dátum>
    <kiszállítási_cím>Irinyi János utca 10.</kiszállítási_cím>
    <összeg>6060</összeg>
  </rendelés>

  <rendelés RID="3" MID="1" FID="3">
    <dátum>2025-10-10</dátum>
    <kiszállítási_cím>Petőfi Sándor utca 1.</kiszállítási_cím>
    <összeg>10880</összeg>
```

```
</rendelés>

<rendelés RID="4" MID="3" FID="4">
  <dátum>2025-11-04</dátum>
  <kiszállítási_cím>Kuruc József utca 23.</kiszállítási_cím>
  <összeg>9470</összeg>
</rendelés>
```

1.4 Az XML dokumentum alapján XMLSchema készítése

Az XML dokumentum szerkezetét és adatintegritását egy külön XSD séma rögzíti (XMLSchemaOE8BW9.xsd). A séma a relációs adatbázis-tervezési elveket követi: minden entitáshoz (rendelés, megrendelő, fizetés, étel, stb.) egyedi azonosítót (elsődleges kulcsot) rendel, és a hivatkozásokat idegen kulcsokkal biztosítja.

A séma gyökéreleme az ételrendelések , amely alatt kollekcióként jelennek meg a fő gyerekelemek. Ezek mind globálisan definiált complexType -ra hivatkoznak. Ahol a gyermekelemek többször fordulhatnak elő (pl. allergén egy ételen belül, vagy maga a rendelés a gyökérben), ott maxOccurs="unbounded" van beállítva.

Az adatintegritás kulcs- és hivatkozási szabályokkal (xs:key , xs:keyref) történik, amelyeket a gyökérelemben definiáltunk az egyedi azonosítókra (pl. RID , MID). A xs:ref attribútumot globális attribútumok definiálásával használtuk a sémában a kód ismétlődésének elkerülése és a jobb karbantarthatóság érdekében.

Az adattípusok és kényszerek a sémában külön simpleType -pal vannak megerősítve:

- azonositoTipus pozitív egész számot követel meg.
- penzosszegTipus nem negatív egész számot vár el.
- telefonszamTipus reguláris kifejezéssel (regex) ellenőrzi a formátumot (06 előtag és a számjegyek száma).
- igenNemTipus és fizetesModjaTipus enumerációval (felsorolással) korlátozza a lehetséges értékeket.

Tervezés és Megvalósítás Menete

A séma megvalósítása a következő lépésekben történt:

Saját Egyszerű Típusok Létrehozása: Az adatok tartalmának szigorítása érdekében egyedi típusokat hoztunk létre. Például az azonositoTipus kizárólag pozitív egész számot engedélyez, a telefonszamTipus reguláris kifejezéssel (regex) ellenőrzi a formátumot, a fizetesModjaTipus pedig xs:enumeration -nel korlátozza a választható értékeket.

Globális Attribútumok és ref Használata: Az ismétlődő azonosítókat (RID , MID , EID , FID , AID) globális attribútumként definiáltuk. Ez lehetővé teszi

a `ref` kulcsszó használatát a komplex típusokon belül, ami csökkenti a redundanciát és növeli az áttekinthetőséget.

```
<!-- GLOBÁLIS ATTRIBÚTUMOK DEFINIÁLÁSA (ref használatához) -->
<xs:attribute name="RID" type="azonositoTipus"/>
<xs:attribute name="MID" type="azonositoTipus"/>
<xs:attribute name="FID" type="azonositoTipus"/>
<xs:attribute name="EID" type="azonositoTipus"/>
<xs:attribute name="AID" type="azonositoTipus"/>
<!-- Példa a hivatkozásra egy komplex típuson belül -->
<xs:complexType name="rendelesTipus">
  <!-- ... sequence elemek ... -->
  <xs:attribute ref="RID" use="required"/> <!-- Itt használjuk a ref-et -->
  <xs:attribute ref="MID" use="required"/>
  <xs:attribute ref="FID" use="required"/>
</xs:complexType>
```

Egyedi, megszorított egyszerű típusok (`simpleType`)

Az adatok validitásának növelése érdekében a szabványos típusokat megszorítjuk (`restriction`) a valós üzleti logikának megfelelően.

```
xml
<!-- Telefonszám típus: 06-al kezdődik, 9-11 számjegyű, regex használatával -->
<xs:simpleType name="telefonszamTipus">
  <xs:restriction base="xs:string">
    <xs:pattern value="06[0-9]{9,11}"/>
  </xs:restriction>
</xs:simpleType>
<!-- Fizetés módja típus (enumeráció) -->
<xs:simpleType name="fizetesModjaTipus">
  <xs:restriction base="xs:string">
    <xs:enumeration value="készpénz"/>
    <xs:enumeration value="bankkártya"/>
    <xs:enumeration value="szép-kártya"/>
  </xs:restriction>
</xs:simpleType>
```

Komplex Típusok: Minden fő adatszerkezethez (pl. `rendelesTipus` , `etelTipus`) komplex típust rendeltünk. Ezek határozzák meg a gyermekelemek sorrendjét (`xs:sequence`), előfordulásukat (`minOccurs` , `maxOccurs`) és a hozzájuk tartozó attribútumokat.

```
<!-- Ételtípus komplex típusa -->
<xs:complexType name="etelTipus">
  <xs:sequence>
    <xs:element name="név" type="xs:string"/>
```

```
        <xs:element name="ár" type="penzosszegTipus"/>
        <!-- Egy ételnek több allergénje is lehet, akár 0 is -->
        <xs:element name="allergén" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="típus" type="xs:string"/>
      </xs:sequence>
      <xs:attribute ref="EID" use="required"/>
    </xs:complexType>
```

PK (Primary Key) és FK (Foreign Key) Definiálása: A relációs adatbázis-logika átültetésre került XSD-be az `xs:key` és `xs:keyref` elemekkel. A gyökérelemen belül definiáltuk ezeket a megszorításokat a hivatkozások érvényesítésére. A `key` garantálja az egyediséget, a `keyref` pedig a hivatkozott elem létezését.

Adatintegritás biztosítása (`key`, `keyref`, `unique`)

A relációs adatbázis-logika átültetése XML sémába a gyökérelemen belül történik az XSD 1.0/1.1 speciális elemeivel.

```
xml
  <xs:element name="ételrendelések">
    <!-- ... complexType, sequence elemek ... -->

    <!-- ELSŐDLEGES KULCSOK (Primary Keys) -->
    <xs:key name="rendelesKey">
      <xs:selector xpath="rendelés"/> <!-- Az útvonal, ahol a kulcsot keressük -->
      <xs:field xpath="@RID"/>      <!-- Az attribútum, ami a kulcs lesz -->
    </xs:key>

    <!-- IDEGEN KULCSOK (Foreign Keys) -->
    <xs:keyref name="rendelesMegrendeloFK" refer="megrendeloKey">
      <xs:selector xpath="rendelés"/> <!-- Az elem, amiben az FK van -->
      <xs:field xpath="@MID"/>      <!-- Az FK attribútum -->
    </xs:keyref>

    <!-- 1:1 kapcsolat a rendelés és fizetés között (FID) -->
    <xs:unique name="rendelesFizetesEgy">
      <xs:selector xpath="rendelés"/>
      <xs:field xpath="@FID"/>
    </xs:unique>
  </xs:element>
```

Speciális Elemek (`unique`): A rendelés és fizetés közötti 1:1 kapcsolat kikényszerítésére az `xs:unique` elemet használtuk.

2.Az Ételrendelések
Projekt name: OE8BW9DomParse

Package: oe8bw9.domparse.hu

Class names: (OE8BW9DomRead, OE8BW9DomModify, OE8BW9DomQuery)

2.1 Adatolvasás

A fejezet célja a megadott XML dokumentum (fájlnév: `OE8BW9_XML.xml`) adatainak beolvasása és feldolgozása Java nyelven, a DOM (Document Object Model) API használatával. A feldolgozás eredményét a program kiírja a konzolra és egy szöveges fájlba (`OE8BW9DomRead_output.txt`) is, "blokk formában". A feladathoz tartozó Java forráskód fájlneve: `OE8BW9DomRead.java` (a csomag neve: `oe8bw9.domparse.hu`).

A tervezés és megvalósítás leírása

A program struktúrája a DOM API szabványos megközelítését követi:

1. **DOM inicializálás:** A `javax.xml.parsers` csomag osztályainak használatával (különösen a `DocumentBuilderFactory` és `DocumentBuilder`) egy "parser" (elemző) jön létre.

2. **XML elemzése:** A parser beolvassa az XML fájlt, és a memória egy `Document` objektumában (DOM-fa) tárolja a teljes hierarchikus adatstruktúrát.

3. **Adatkinyerés:** A `doc.getElementsByTagName()` metódussal történik az egyes főbb XML elemek (pl. `<rendelés>`, `<étel>`, `<akció>`) kiválasztása. Egy `for` ciklus segítségével iterálunk a kapott `NodeList` elemein.

4. **Adatok elérése:** Az egyes `Node`-okat `Element`-té kasztolva érjük el az attribútumaikat (`getAttribute()`) és a gyermekelemeik tartalmát (`getTextContent()`).

5. **Kimenet kezelése:** A kód egy `printlnBoth(PrintWriter pw, String line)` nevű segédmetódust alkalmaz a szinkronizált írás érdekében, amely egyszerre ír a konzolra és a kimeneti fájlba.

6. **Hibakezelés:** A program a fő metódus szintjén kezeli a szükséges kivételeket (`SAXException`, `IOException`, `ParserConfigurationException`).

Lényeges kód kiemelése és magyarázata

1. DOM Document inicializálása és az XML beolvasása

Ez a kódrészlet felelős a fájl beolvasásáért és a DOM-fa memóriában való felépítéséért.

```
java
// Létrehozzuk a DocumentBuilderFactory és DocumentBuilder példányokat
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = factory.newDocumentBuilder();
```

```
// XML beolvasása DOM-ba
Document doc = dBuilder.parse(xmlFile);
doc.getDocumentElement().normalize();
```

Magyarázat: A `DocumentBuilder.parse()` hívása elemzi az XML fájlt, és a teljes struktúrát egy `Document` objektumba tölti, amely a DOM-fa gyökere. A `normalize()` segít a strukturális integritás biztosításában.

2. Adatok kinyerése attribútumokból és gyermekelemekből

Az adatok elérését a segédmetódusok és az attribútumlekérdezés mutatja be.

```
java
if (n.getNodeType() == Node.ELEMENT_NODE) {
   Element e = (Element) n;
   String rid = e.getAttribute("RID"); // Attribútum kiolvasása
   String datum = getText(e, "dátum"); // Gyermek elem szövege (segédmetódussal)
   // ...
}
```

Magyarázat: A kódban az `e.getAttribute("RID")` közvetlenül lekéri az attribútum értékét. A `getText(e, "dátum")` egy segédmetódust hív, amely biztonságosan kinyeri a megadott nevű első gyermekelem szöveges tartalmát.

3. Többszörösen beágyazott elemek kezelése (Akciók)

Az akcióknál látható az összetettebb hierarchia kezelése (`<időtartam>` elemen belül a `<kezdete>` és `<vége>`).

```
java
// A 'kezdete' és 'vége' az 'időtartam' gyerekei
Node idoNode = e.getElementsByTagName("időtartam").item(0);
if (idoNode != null) {
    // A segédfüggvényt meghívjuk a belső elemen (idoNode)
    printlnBoth(pw, "Kezdete: " + getText((Element) idoNode, "kezdete"));
    printlnBoth(pw, "Vége: " + getText((Element) idoNode, "vége"));
}
```

Először lekérjük a szülő `<akció>` elemből az `<időtartam>` gyermekét, majd ezen a belső elemen (`idoNode`) hívjuk meg a `getText` segédmetódust a még mélyebben fekvő adatok eléréséhez.

4. A `getText` és `printlnBoth` segédfüggvények

A kód alján elhelyezkedő segédmetódusok biztosítják a kód modularitását és a kettős kimenetet.

```java
/** Első tagName gyermek szövegének visszaadása, ha nincs: üres string. */
private static String getText(Element parent, String tagName) {
   // ... null ellenőrzések a biztonságos működésért ...
   return nl.item(0).getTextContent().trim();
}

/** Ugyanazt a sort kiírja a konzolra és a fájlba is. */
private static void printlnBoth(PrintWriter pw, String line) {
   System.out.println(line); // Kiírás konzolra
   pw.println(line);       // Kiírás fájlba
}
```

2.2 Adat-lekérdezés

A fejezet célja az `OE8BW9_XML.xml` XML dokumentumban tárolt adatok lekérdezése Java DOM API segítségével, kifejezetten XPath kifejezések használata nélkül. A lekérdezések eredményét a program kiírja a konzolra. A feladathoz tartozó Java forráskód fájlneve: `OE8BW9DomQuery.java`.

A tervezés és megvalósítás leírása

Tervezés:

1. **DOM felépítése:** Az XML fájl beolvasása a memóriába, akárcsak az olvasás feladatban.
2. **Lekérdezési logika:** Mivel XPath nem használható, a lekérdezések a `doc.getElementsByTagName()` metóduson és a Java nyelvi szerkezetein (feltételes elágazások, ciklusok, `List`, `Map` adatszerkezetek) alapulnak.
3. **Konzol kimenet:** A program a lekérdezések eredményét strukturált formában, blokkokba rendezve írja ki a konzolra.
4. **Komplex lekérdezés:** Egy összetettebb lekérdezés megvalósításához "manuális join"-t kell végrehajtani a különböző adatelemek (például rendelés és megrendelő neve) között, ehhez `LinkedHashMap` adatszerkezeteket használunk.
5. **Segédmetódusok:** A kód olvashatóságának érdekében segédmetódusok kerültek bevezetésre (`getText`, `startBlock`, `endBlock`, `asList`).

Lényeges kód kiemelése és magyarázata

1. Lekérdezés feltétel alapján (Pizza típusú ételek)

Ez a lekérdezés bemutatja, hogyan lehet egy adott feltételnek megfelelő elemeket kiszűrni és listába gyűjteni.

```java
// ===== 1) Ételek nevei listában (Pizza típusúak) =====
List<String> pizzaNevek = new ArrayList<>();
NodeList etelek = doc.getElementsByTagName("étel");
for (int i = 0; i < etelek.getLength(); i++) {
    Node n = etelek.item(i);
    if (n.getNodeType() == Node.ELEMENT_NODE) {
        Element e = (Element) n;
        String tipus = getText(e, "típus");
        if ("Pizza".equals(tipus)) { // Feltétel ellenőrzése
            String nev = getText(e, "név");
            if (!nev.isEmpty())
                pizzaNevek.add(nev);
        }
    }
}
```

Végigiterálunk az összes `<étel>` elemen, lekérjük a "típus" gyermekelem tartalmát, és ha az pontosan "Pizza", akkor a "név" értéket hozzáadjuk egy Java `ArrayList`-hez.

2. Összegző lekérdezés (Készpénzes fizetések száma)

Ez a kód egy egyszerű összesítést végez egy számláló segítségével.

```java
// ===== 3) Készpénzes fizetések száma =====
NodeList fizetesek = doc.getElementsByTagName("fizetés");
int kpSzam = 0;
for (int i = 0; i < fizetesek.getLength(); i++) {
    Node n = fizetesek.item(i);
    if (n.getNodeType() == Node.ELEMENT_NODE) {
        Element e = (Element) n;
        String modja = getText(e, "fizetés_módja");
        if ("készpénz".equals(modja)) {
            kpSzam++; // Számláló növelése
        }
    }
}
```

A program az összes `<fizetés>` elemet megvizsgálja. Ha a "fizetés_módja" elem értéke "készpénz", a `kpSzam` változó értékét növeli.

3. Összetett lekérdezés "manuális join"-nal

A legbonyolultabb lekérdezés, amely több különböző XML-entitásból (Rendelés, Megrendelő, Fizetés) származó adatokat kapcsol össze attribútum-azonosítók alapján.

```
java
// Előkészítünk "join" segéd Map-eket
Map<String, String> megrendeloNevById = new LinkedHashMap<>();
// ... feltöltés a megrendelők nevével és MID attribútumával ...

Map<String, String> fizetesModById = new LinkedHashMap<>();
// ... feltöltés a fizetési móddal és FID attribútumával ...

// Fő ciklus a rendeléseken
NodeList rendelesek = doc.getElementsByTagName("rendelés");
for (int i = 0; i < rendelesek.getLength(); i++) {
   Element e = (Element) rendelesek.item(i);
   String mId = e.getAttribute("MID");
   String fId = e.getAttribute("FID");
   // ... egyéb adatok kinyerése ...

   // Összekapcsolás a Map-ek segítségével
   String megrendeloNev = megrendeloNevById.getOrDefault(mId, "(ismeretlen
megrendelő)");
   String fizetesMod = fizetesModById.getOrDefault(fId, "(ismeretlen fizetés)");

   System.out.println("- " + datum + " | Cím: " + cim + " | Megrendelő: " + megrendeloNev + "
| Fizetés módja: " + fizetesMod);
}
```

A lekérdezéshez szükséges volt először két `Map` adatszerkezetet létrehozni, amelyek kulcsként az azonosítókat (MID, FID), értékként a hozzájuk tartozó neveket/módokat tárolják. Ezután a fő `<rendelés>` cikluson belül ezekből a Map-ekből kérjük le a kapcsolódó adatokat az attribútumok (`mId` , `fId`) alapján, szimulálva egy adatbázis-szerű JOIN műveletet.

2.3 Adatmódosítás

A fejezet célja a meglévő XML dokumentum (`OE8BW9_XML.xml`) adatainak módosítása, új adatok hozzáadása, valamint a módosított DOM-fa elmentése egy új XML fájlba (`MOD_NeptunkodXML.xml`) Java DOM API használatával. A feladathoz tartozó Java forráskód fájlneve: `OE8BW9DomModify.java` .

A tervezés és megvalósítás leírása

A program a DOM-fa memóriában történő felépítését követően végrehajt négy különböző adatmanipulációs műveletet:

1. **DOM inicializálás:** Az XML fájl beolvasása a memóriába `Document` objektumként.

2. **Elemek keresése:** A módosítandó elemeket attribútumérték alapján azonosítjuk egy `byAttr()` segédmetódus segítségével, elkerülve az XPath használatát.

3. **Adatok módosítása:**

- Létező gyermekelemek tartalmának frissítése (`setText()`).

- Új gyermekelem hozzáadása, ha az adott tartalom még nem létezik (`addChildIfMissingWithText()`).

4. **Mentés:** A módosított DOM-fát a `javax.xml.transform` API segítségével írjuk ki egy új XML fájlba, biztosítva a szép formázást (behúzás, UTF-8 kódolás).

5. **Konzol kimenet:** A módosítások előtti és utáni állapotot a konzolra írjuk az ellenőrzés megkönnyítése érdekében.

Lényeges kód kiemelése és magyarázata
1. Elemek megkeresése attribútum alapján (`byAttr` segédmetódus)

Mivel XPath nem használható, egy saját segédmetódusra volt szükség a konkrét elemek gyors megtalálásához.

```
java
/** Megkeres egy elemet adott attribútum név és érték alapján. */
private static Element byAttr(Document doc, String tag, String attr, String val) {
   NodeList nl = doc.getElementsByTagName(tag);
   for (int i = 0; i < nl.getLength(); i++) {
      Node n = nl.item(i);
      if (n.getNodeType() == Node.ELEMENT_NODE) {
         Element e = (Element) n;
         if (val.equals(e.getAttribute(attr))) // Attribútum értékének ellenőrzése
            return e; // A megfelelő elem megtalálva
      }
   }
   return null;
}
```

A függvény végignézi az összes megadott nevű tag-et (`tag`), ellenőrzi, hogy az `attr` nevű attribútumának értéke megegyezik-e a keresett `val` értékkel. Ha igen, visszaadja az `Element` objektumot.

2. Elem tartalmának módosítása és új elem hozzáadása, ha szükséges (`setText`)

Ez a metódus kezeli a leggyakoribb módosítási esetet: egy gyermekelem tartalmának beállítását.

```
java
/** Egy gyermekelem szöveges tartalmát állítja be/módosítja. */
private static void setText(Element parent, String tag, String value) {
   NodeList nl = parent.getElementsByTagName(tag);
   if (nl.getLength() == 0) {
      // Ha a gyermekelem nem létezik, létrehozzuk
      Element child = parent.getOwnerDocument().createElement(tag);
      child.setTextContent(value);
      parent.appendChild(child);
   } else {
      nl.item(0).setTextContent(value); // Módosítás
   }
}
```

A metódus nemcsak módosítja a létező elemet, hanem robusztusan hozzá is adja azt, ha korábban nem létezett az adott szülő alatt.

3. Étel árának módosítása (1. lekérdezés)

Konkrét példa a módosításra, ahol az adatokat int-té alakítjuk a matematikai művelethez.

```
java
// 1) Étel (EID=1 - Hawaii pizza): ár frissítése (növelése 500 Ft-tal)
Element etel1 = byAttr(doc, "étel", "EID", "1");
if (etel1 != null) {
   int currentPrice = parseInt(getText(etel1, "ár"), 0);
   int newPrice = currentPrice + 500;
   setText(etel1, "ár", String.valueOf(newPrice)); // Az új ár beállítása
   printEtel(etel1);
}
```

Megkeressük az EID="1" attribútumú elemet, kiolvassuk az aktuális árat, 500-at adunk hozzá, majd a `setText` segédmetódussal beírjuk az új értéket.

4. A módosított DOM-fa mentése

A végleges lépés a memóriában lévő adatok kiírása az új XML fájlba.

```
java
// Mentés új fájlba
saveXml(doc, "MOD_NeptunkodXML.xml");
```

```
// ...
private static void saveXml(Document doc, String filename) throws Exception {
    TransformerFactory transformerFactory = TransformerFactory.newInstance();
    Transformer transformer = transformerFactory.newTransformer();
    transformer.setOutputProperty(OutputKeys.INDENT, "yes"); // Szép formázás
    transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
    DOMSource source = new DOMSource(doc);
    StreamResult result = new StreamResult(new File(filename));
    transformer.transform(source, result);
    System.out.println("\nMentve: " + filename);
}
```

A Java XSLT Transzformátor API-ját használjuk a mentéshez. Az `OutputKeys.INDENT, "yes"` opció biztosítja, hogy a kimeneti XML fájl olvasható, behúzott formátumú legyen.