JEGYZŐKÖNYV

ADATKEZELÉS XML-BEN

FÉLÉVES FELADAT

RAKTÁR ALKALMAZÁS

Készítette : Nagy Boldizsár

Neptunkód : IKXS9J

Dátum : 2024.12.01.

Tartalomjegyzék :

Bevezetés :

Az XML (Extensible Markup Language) egy széles körben használt adatmegjelenítési és -tárolási szabvány, amely lehetőséget ad az adatok hierarchikus és strukturált formában történő kezelésére. A feladat célja egy XML adatkezelő rendszer létrehozása, amely lehetőséget biztosít adatok strukturált tárolására, kezelésére és módosítására. A rendszerhez kapcsolódóan egy ER (Entitás-Kapcsolat) modell és az abból származtatott XML dokumentummodell (XDM) készült, amely megalapozza az adatok validálását és adminisztrációját.

A DOM (Document Object Model) API segítségével Java nyelven készült programok lehetővé teszik az XML dokumentum adatainak kiolvasását, lekérdezését, módosítását és kiírását. A feladat gyakorlati betekintést nyújt az XML alapú rendszerek fejlesztésébe, miközben megismerteti a modern adatkezelési technológiák alapvető eszközeit. A projekt keretében létrejött rendszer egy komplex, de jól strukturált adatkezelési feladat megoldását mutatja be, amely hozzájárul a webes és adatkezelési ismeretek bővítéséhez.

A feladat Leírása :

A feladat során egy XML alapú adatkezelő rendszert kellett létrehozni, amelyhez egy jól strukturált ER modellt dolgoztam ki. Ez tartalmazott legalább öt entitást (pl. *Felhasználók*, *Raktárak*, *Események*), amelyekhez különböző kapcsolatokat (1:1, 1:N, N:M) és attribútumokat (normál, kulcs, összetett, többértékű) rendeltem. Az ER modellből XDM modellt készítettem, amely az XML adatstruktúrákhoz igazodik, és biztosítja az adatok validációját. Az XDM modell alapján egy validált XML dokumentumot (*XML_IKXS9J.xml*) hoztam létre, amely az entitások ismétlődő elemeiből legalább három példányt tartalmazott, részletes megjegyzésekkel kiegészítve.

A második részfeladat során a DOM API segítségével Java programokat készítettem, amelyek az XML dokumentum adminisztrációs feladatait valósítják meg. Négy osztály került megvalósításra:

- **DOMReadIKXS9J**: Az XML dokumentum adatait olvassa be és jeleníti meg.
- **DOMQueryIKXS9J**: Lehetővé teszi az XML adatok célzott lekérdezését.
- **DOMModifyIKXS9J**: Az XML dokumentum tartalmának módosítását végzi (pl. attribútumok és elemek frissítése).
- **DOMWriteIKXS9J :** Az XML dokumentum hierarchikus struktúráját jeleníti meg a konzolon, és egy új fájlba is kiírja.

A feladat elkészítése során figyelmet fordítottam az XML dokumentum helyes validációjára és a Java kódok olvashatóságára. Az elkészült projekt átfogó megoldást nyújtott az XML alapú adatkezelési feladatok végrehajtására, beleértve az adatok kezelését és adminisztrációját is. Ez a feladat lehetővé tette az XML kezelésével kapcsolatos ismeretek gyakorlati alkalmazását, amely kulcsfontosságú a webes fejlesztések terén.

1. Feladat : Raktár Alkalmazás létrehozása

1.1 ER Modell Megvalósítása :

Az ER modell megtervezése során a táblák (entitások) a rendszer legfontosabb objektumait képviselik, amelyek között kapcsolatok (relációk) jönnek létre. A struktúra az alábbiak szerint épült fel:

1. Egyedek (Entitások)

Az adatbázis négy fő entitást tartalmaz:

- **Felhasználók:** A rendszer felhasználói adatait tárolja, mint például név, email, nem, és jelszó.

- **Raktárak:** A raktárak adatait rögzíti, beleértve az árakat (bérlés és vétel), címet, és tulajdonosokat.
- **Események:** A rendszer eseményeit tartalmazza, mint a licitálások, kiárusítások vagy bolhapiacok időpontja és helyszíne.
- **Kedvezmények:** A felhasználók számára biztosított kedvezményeket tárolja százalékos értékben és érvényességi dátummal.

2. Kapcsolatok (Relációk)

A modellben az entitások között különféle kapcsolatok jöttek létre:

- **1:N kapcsolat** a Felhasznalok és Raktarak között: Egy felhasználó lehet egy vagy több raktár tulajdonosa.
- **1:N kapcsolat** a Felhasznalok és Kedvezmenyek között: Egy felhasználó több kedvezményt is kaphat.
- **M:N kapcsolat** a Felhasznalok és Raktarak között a Berles_Vasarlas kapcsolótábla révén: Egy felhasználó több raktárt is bérelhet vagy megvásárolhat, és a raktárak bérléseit vagy vásárlásait több felhasználó is használhatja.

3. Tulajdonságok (Attribútumok)

Az entitásokhoz tartozó attribútumok a legfontosabb jellemzőket írják le:

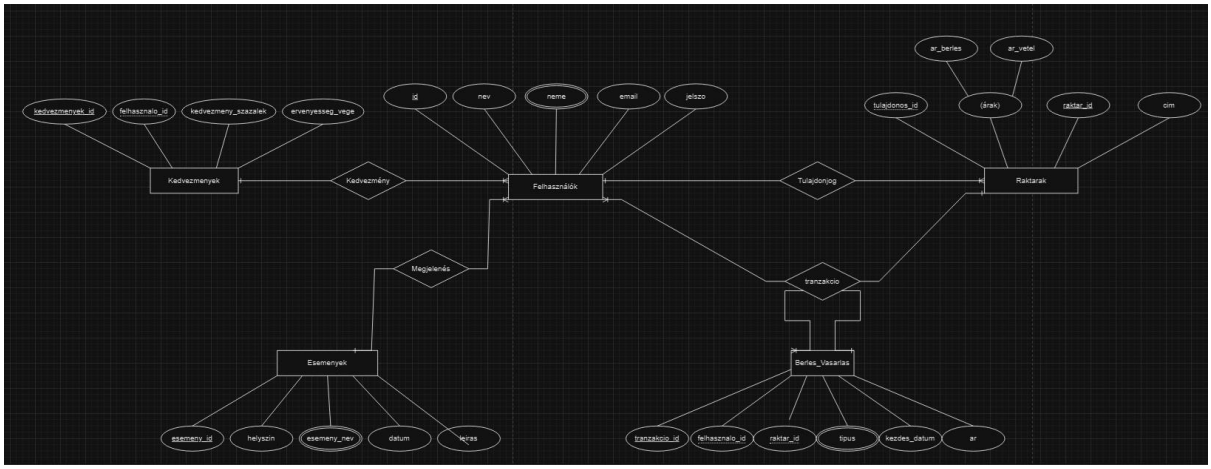
- **Normál attribútumok:** Pl. név, cím, esemény neve.
- **Összetett attribútumok:** Pl. az ár bérlés és vétel szerint külön oszlopban szerepel a Raktarak táblában.
- **Kulcs attribútumok:** Minden entitás egyedi azonosítóval rendelkezik, például id vagy tranzakcio_id.

4. Kapcsolatok ábrázolása

A kapcsolatok nyilakkal vannak jelölve, amelyek jelzik a kapcsolat típusát:

- 1:1, például Kedvezmenyek és Felhasznalok között.
- 1:N, például a raktárak és tulajdonosok között.
- M:N, például a felhasználók és raktárak tranzakcióinál.

A vizuális ábrázolás az ER diagramon különféle formák (téglalapok, ellipszisek, rombuszok) segítségével jelenik meg. A kapcsolatok és attribútumok világosan mutatják az entitások közötti összefüggéseket.



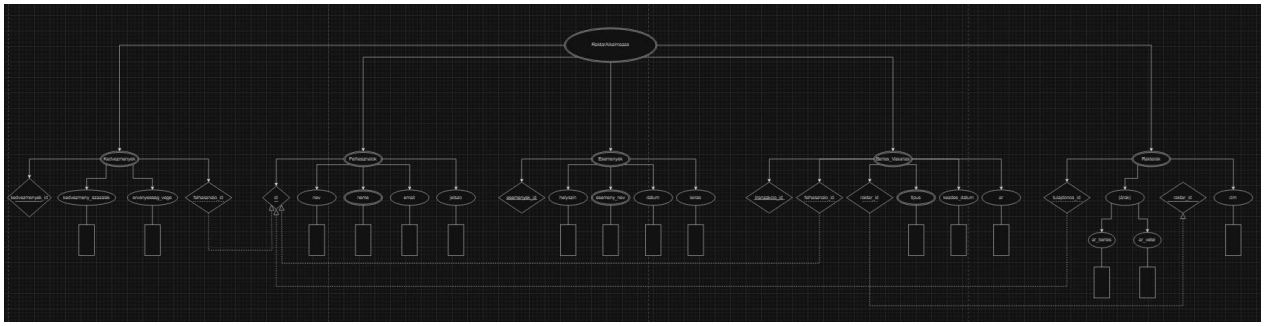
1-es ábra : RaktárAlkalmazás ER Modell

1.2 XDM Modell Megvalósítása :

Az ER modellből az XDM modellre történő átalakítás célja, hogy az adatbázis szerkezetét XML-re optimalizált formában jelenítse meg. Az XDM modell a hierarchikus struktúrát hangsúlyozza, amely jól illeszkedik az XML dokumentumok faalapú felépítéséhez. Az átalakítás során:

1. **Entitások konvertálása elemekké:**
    a. Az ER modell entitásait az XML dokumentum gyökér és alárendelt elemeiként alakítottuk át. Például a Felhasznalok és Raktarak külön XML elemekként jelennek meg.
2. **Kapcsolatok hierarchikus ábrázolása:**
    a. Az 1:N kapcsolatok (pl. felhasználók és raktárak között) XML-en belül szülő-gyermek struktúrával jelennek meg.
    b. Az M:N kapcsolatok (pl. bérlés és vásárlás tranzakciók) külön elemekként kerültek kialakításra, hivatkozva az érintett entitásokra.
3. **Attribútumok szerepeltetése:**
    a. Az egyes entitások attribútumai XML attribútumként vagy elemek alá tartozó alárendelt elemekként jelennek meg, attól függően, hogy egyszerű vagy összetett adatot tárolnak.
4. **Kulcsok és hivatkozások:**
    a. Az elsődleges kulcsok az elemek azonosítói, amelyek attribútumként vannak ábrázolva. Az idegen kulcsok az XML elemek közötti hivatkozások létrehozására szolgálnak.

Az XDM modell a szabványos szimbólumok és hierarchiák használatával készült, biztosítva az egyértelműséget. Az elemeket összekötő vonalak logikusan követik a kapcsolatokat, elkerülve a kereszteződéseket.



2-es ábra : RaktárAlkalmazás XDM Modell

1.3 **Az XDM modell alapján XML dokumentum készítése :**

Az adatbázis XDM modelljének alapjaiból kiindulva elkészítettük az XML dokumentumot, amely az adatok hierarchikus, fa-struktúrában történő ábrázolását teszi lehetővé. Az alábbi lépések mentén történt a megvalósítás:

1. **Gyökérelem meghatározása:**
    a. A teljes XML dokumentum a RaktarAlkalmazas gyökérelem alá került, amely tartalmazza az adatbázis összes tábláját és azok tartalmát.
2. **Táblák átalakítása elemekké:**
    a. Az adatbázis táblák az XML dokumentumban külön elemekként (Felhasznalok, Raktarak, Esemenyek, stb.) jelennek meg.
3. **Sorok ábrázolása elemekként:**
    a. A táblák sorait egy-egy XML elem reprezentálja, például a Felhasznalo vagy a Raktar.
4. **Attribútumok és adatok:**
    a. Az egyes táblák oszlopai az XML elemek alá tartozó alárendelt elemekként lettek feltüntetve, például a Felhasznalo tartalmazza az id, nev, email és egyéb attribútumokat.
5. **Többszörös előfordulás biztosítása:**
    a. Azokon a helyeken, ahol több rekord is előfordul (pl. Felhasznalok, Raktarak, stb.), legalább három példány szerepel az XML dokumentumban.
6. **Kapcsolatok ábrázolása:**
    a. Az idegen kulcsok az XML-ben hivatkozásként jelennek meg, például a tulajdonos_id a Raktarak elemekben.
7. **Megjegyzések használata:**
    a. Az átláthatóság érdekében minden főbb szekciót megjegyzéssel láttunk el, jelezve az adattáblák forrását és szerepét.

Az XML dokumentum kódja :

A fenti lépések alapján az XML kód a teljes adatbázist reprezentálja, a kapcsolatok és attribútumok egyértelműen követhetők. Az XML dokumentumban az egyes táblák elemei hierarchikusan és strukturáltan jelennek meg, biztosítva a könnyű feldolgozhatóságot.

Kód :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<RaktarAlkalmazas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XMLSchemaIKXS9J.xsd">



    <!-- Felhasználók táblája -->

    <Felhasznalok>

        <Felhasznalo>

            <id>1</id>

            <nev>Kovács István</nev>

            <email>istvan.kovacs@example.com</email>

            <neme>Férfi</neme>

            <jelszo>jelszo123</jelszo>

        </Felhasznalo>

        <Felhasznalo>

            <id>2</id>

            <nev>Nagy Erika</nev>

            <email>erika.nagy@example.com</email>

            <neme>Nő</neme>

            <jelszo>biztonsagosJelszo</jelszo>

        </Felhasznalo>

        <Felhasznalo>

            <id>3</id>

            <nev>Szabó Péter</nev>

            <email>peter.szabo@example.com</email>

            <neme>Férfi</neme>

            <jelszo>12345</jelszo>

        </Felhasznalo>

    </Felhasznalok>

    <!-- Raktárak táblája -->

    <Raktarak>
```

```
    <Raktar>
        <raktar_id>1</raktar_id>
        <ar_berles>10000</ar_berles>
        <ar_vetel>150000</ar_vetel>
        <cim>Budapest, Váci út 10.</cim>
        <tulajdonos_id>1</tulajdonos_id>
    </Raktar>
    <Raktar>
        <raktar_id>2</raktar_id>
        <ar_berles>8000</ar_berles>
        <ar_vetel>120000</ar_vetel>
        <cim>Debrecen, Fő út 12.</cim>
        <tulajdonos_id>2</tulajdonos_id>
    </Raktar>
    <Raktar>
        <raktar_id>3</raktar_id>
        <ar_berles>15000</ar_berles>
        <ar_vetel>200000</ar_vetel>
        <cim>Pécs, Tavasz utca 5.</cim>
        <tulajdonos_id>3</tulajdonos_id>
    </Raktar>
</Raktarak>
<!-- Események táblája -->
  <Esemenyek>
    <Esemeny>
        <esemeny_id>1</esemeny_id>
        <esemeny_nev>Licit</esemeny_nev>
        <datum>2024-12-05</datum>
        <helyszin>Budapest, Váci út 10.</helyszin>
        <leiras>Raktár árverés.</leiras>
    </Esemeny>
```

```
    <Esemeny>
        <esemeny_id>2</esemeny_id>
        <esemeny_nev>Kiárusítás</esemeny_nev>
        <datum>2024-12-10</datum>
        <helyszin>Debrecen, Fő út 12.</helyszin>
        <leiras>Kedvezményes áron elérhető termékek.</leiras>
    </Esemeny>
    <Esemeny>
        <esemeny_id>3</esemeny_id>
        <esemeny_nev>Bolhapiac</esemeny_nev>
        <datum>2024-12-15</datum>
        <helyszin>Pécs, Tavasz utca 5.</helyszin>
        <leiras>Használt tárgyak árusítása.</leiras>
    </Esemeny>
</Esemenyek>
      <!-- Bérlések táblája -->
  <Berlesek>
    <Berles>
        <tranzakcio_id>1</tranzakcio_id>
        <felhasznalo_id>2</felhasznalo_id>
        <raktar_id>1</raktar_id>
        <tipus>Bérlés</tipus>
        <kezdes_datum>2024-12-01</kezdes_datum>
        <ar>10000</ar>
    </Berles>
  </Berlesek>
 <!-- Kedvezmények táblája -->
  <Kedvezmenyek>
    <Kedvezmeny>
        <kedvezmeny_id>1</kedvezmeny_id>
        <felhasznalo_id>3</felhasznalo_id>
```

```
            <kedvezmeny_szazalek>15</kedvezmeny_szazalek>

            <ervenyesseg_vege>2024-12-31</ervenyesseg_vege>

        </Kedvezmeny>

      </Kedvezmenyek>

    </RaktarAlkalmazas>
```

1.4 Az XML dokumentum alapján XMLSchema készítése:

Az XML dokumentumunkhoz XSD (XML Schema Definition) létrehozásának célja, hogy meghatározza az adatok szerkezetét, az adatelemek típusát, valamint az adatok közötti kapcsolatok érvényességét. Ez segít biztosítani az XML dokumentum adatainak érvényességét, például az egyedi azonosítók (PK - Primary Key) és a kapcsolatok (FK - Foreign Key) helyes meghatározásával. Az XSD tartalmaz speciális típusokat, referenciaelemeket és komplex típusokat is, hogy az adatbázishoz kapcsolódó üzleti logikát tükrözze.

Az XSD implementálása során:

1. Minden elemhez és attribútumhoz megfelelő adattípusokat rendeltünk.
2. Komplex típusokat hoztunk létre, amelyek logikailag összetartozó elemek csoportját definiálják.
3. Egyedi azonosítókat (key) és idegen kulcsokat (keyref) definiáltunk, hogy biztosítsuk az adatok közötti kapcsolatok konzisztenciáját.
4. Speciális elemeket (pl. enumerációkat) hoztunk létre bizonyos mezők értékeinek korlátozására.

A kód teljes körű validációt nyújt az XML dokumentum számára, beleértve az egyedi azonosítókat és az idegen kulcsok hivatkozásait is.

Kód :

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">


    <!-- RaktarAlkalmazas gyökérelem -->

    <xs:element name="RaktarAlkalmazas">

        <xs:complexType>

            <xs:sequence>

                <xs:element name="Felhasznalok" type="FelhasznalokType"/>

                <xs:element name="Raktarak" type="RaktarakType"/>

                <xs:element name="Esemenyek" type="EsemenyekType"/>

                <xs:element name="Berlesek" type="BerlesekType"/>
```

```
            <xs:element name="Kedvezmenyek" type="KedvezmenyekType"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<!-- Felhasznalok -->
<xs:complexType name="FelhasznalokType">
    <xs:sequence>
        <xs:element name="Felhasznalo" type="FelhasznaloType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="FelhasznaloType">
    <xs:sequence>
        <xs:element name="id" type="xs:int"/>
        <xs:element name="nev" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
        <xs:element name="neme" type="NemeType"/>
        <xs:element name="jelszo" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<!-- Neme enum -->
<xs:simpleType name="NemeType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Férfi"/>
        <xs:enumeration value="Nő"/>
        <xs:enumeration value="Egyéb"/>
    </xs:restriction>
</xs:simpleType>

<!-- Raktarak -->
<xs:complexType name="RaktarakType">
    <xs:sequence>
        <xs:element name="Raktar" type="RaktarType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="RaktarType">
    <xs:sequence>
        <xs:element name="raktar_id" type="xs:int"/>
        <xs:element name="ar_berles" type="xs:decimal"/>
        <xs:element name="ar_vetel" type="xs:decimal"/>
        <xs:element name="cim" type="xs:string"/>
        <xs:element name="tulajdonos_id" type="xs:int"/>
    </xs:sequence>
</xs:complexType>

<!-- Esemenyek -->
<xs:complexType name="EsemenyekType">
    <xs:sequence>
        <xs:element name="Esemeny" type="EsemenyType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="EsemenyType">
    <xs:sequence>
        <xs:element name="esemeny_id" type="xs:int"/>
        <xs:element name="esemeny_nev" type="EsemenyNevType"/>
        <xs:element name="datum" type="xs:date"/>
        <xs:element name="helyszin" type="xs:string"/>
        <xs:element name="leiras" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<!-- Esemeny_nev enum -->
<xs:simpleType name="EsemenyNevType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Licit"/>
        <xs:enumeration value="Kiárusítás"/>
        <xs:enumeration value="Bolhapiac"/>
    </xs:restriction>
</xs:simpleType>

<!-- Berlesek -->
<xs:complexType name="BerlesekType">
```

```
    <xs:sequence>
        <xs:element name="Berles" type="BerlesType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="BerlesType">
    <xs:sequence>
        <xs:element name="tranzakcio_id" type="xs:int"/>
        <xs:element name="felhasznalo_id" type="xs:int"/>
        <xs:element name="raktar_id" type="xs:int"/>
        <xs:element name="tipus" type="TipusType"/>
        <xs:element name="kezdes_datum" type="xs:date"/>
        <xs:element name="ar" type="xs:decimal"/>
    </xs:sequence>
</xs:complexType>

<!-- Tipus enum -->
<xs:simpleType name="TipusType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Bérlés"/>
        <xs:enumeration value="Vétel"/>
    </xs:restriction>
</xs:simpleType>

<!-- Kedvezmenyek -->
<xs:complexType name="KedvezmenyekType">
    <xs:sequence>
        <xs:element name="Kedvezmeny" type="KedvezmenyType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="KedvezmenyType">
    <xs:sequence>
        <xs:element name="kedvezmeny_id" type="xs:int"/>
        <xs:element name="felhasznalo_id" type="xs:int"/>
        <xs:element name="kedvezmeny_szazalek" type="xs:decimal"/>
        <xs:element name="ervenyesseg_vege" type="xs:date"/>
    </xs:sequence>
```

```
    </xs:complexType>


</xs:schema>
```

2. Feladat : DOM program készítése

A feladat egy DOM program (DOM szabvány elemeit felhasználva) készítése az XML dokumentum (XML_IKXS9J.xml) adatai adminisztrálása alapján

Project name: DOMParseIKXS9J

Package: hu.domparse.IKXS9J

Class names:

DomReadIKXS9J

DOMWriteIKXS9J

DomQueryIKXS9J

DomModifyIKXS9J

2.1 DomRead megvalósítása / adatolvasás :

A projekt célja egy **DOM-alapú XML-feldolgozó** létrehozása, amely képes egy XML fájl elemeinek beolvasására, feldolgozására, és az adatok kinyerésére Java nyelven. A program különböző XML elemeket dolgoz fel, például felhasználókat, raktárakat, eseményeket, bérléseket és kedvezményeket. Az XML-feldolgozás során a program a org.w3c.dom és a javax.xml.parsers csomagokat használja.

Főbb lépések:

1. **XML-fájl betöltése**: A program a DocumentBuilderFactory és DocumentBuilder osztályokat használja az XML fájl beolvasására és DOM objektummá alakítására.
2. **Gyökérelem normalizálása**: Ez biztosítja, hogy a DOM szerkezet helyes legyen, és könnyebben lehessen vele dolgozni.
3. **Adatok feldolgozása**: A program különböző elemtípusok (Felhasznalo, Raktar, stb.) szerint iterál az XML dokumentumon, és kinyeri az adott elem attribútumait vagy gyermekelemeit.
4. **Hibakezelés**: Az XML-feldolgozás során esetleges hibákat a try-catch blokk kezeli, ezzel biztosítva a program robusztusságát.
5.

Kiemelt kódrészlet és magyarázat

Példa a felhasználók feldolgozására:

```
NodeList felhasznalok = document.getElementsByTagName("Felhasznalo");
```

```
for (int i = 0; i < felhasznalok.getLength(); i++) {

    Node node = felhasznalok.item(i);

    if (node.getNodeType() == Node.ELEMENT_NODE) {

        Element felhasznalo = (Element) node;

System.out.println("ID:"+felhasznalo.getElementsByTagName("id").item(0).getTextContent());

System.out.println("Név:"+felhasznalo.getElementsByTagName("nev").item(0).getTextContent());

System.out.println("Email:"+felhasznalo.getElementsByTagName("email").item(0).getTextContent()
);

System.out.println("Neme:"+felhasznalo.getElementsByTagName("neme").item(0).getTextContent());

System.out.println("Jelszó:"+felhasznalo.getElementsByTagName("jelszo").item(0).getTextContent
());

        System.out.println();

    }

}
```

A teljes kód az alábbi linken található: Github link

2.2 DomWrite megvalósítása / adatírás :

A DomWriteIKXS9J program célja az XML dokumentum fa struktúrájának beolvasása, kiírása a konzolra, és a dokumentum egy másik fájlba történő mentése. Ez a program jól demonstrálja a DOM (Document Object Model) használatát az XML dokumentumok feldolgozására és manipulálására.

A program fő lépései:

1. **XML dokumentum beolvasása:**
    a. A DocumentBuilderFactory és a DocumentBuilder osztályokat használja az XML fájl beolvasására és DOM objektummá alakítására.
    b. Az eredeti fájl neve: XML_IKXS9J.xml.

2. **Fa struktúra kiírása a konzolra:**
    - A printNode metódus rekurzív módon bejárja az XML dokumentumot, és kiírja a csomópontok nevét, valamint az attribútumokat és szöveges tartalmakat.
    - Ez segít vizuálisan megérteni az XML dokumentum hierarchiáját.

3. **Az XML dokumentum mentése új fájlba:**
    - A TransformerFactory és a Transformer osztályokat használja az XML dokumentum mentésére.
    - A mentett fájl neve: XML_IKXS9J1.xml.
    - Az OutputKeys.INDENT beállítással formázott (behúzott) struktúrát hoz létre.

Példa a kódból :

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(new File("XML_IKXS9J1.xml"));
transformer.transform(source, result);
```

4. **Rekurzív metódus a fa struktúra feldolgozására (printNode):**
   - Bejárja az összes gyermekelem-csomópontot és attribútumot, és konzolra írja azok nevét és értékeit.
   - Használ tabulátorokat a hierarchia vizualizálására.

Példa a Kódból :

```
private static void printNode(Node node, int indent) {
    String indentString = new String(new char[indent]).replace('\0', '\t');
    System.out.println(indentString + "Node Name: " + node.getNodeName());
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        NodeList childNodes = node.getChildNodes();
        for (int i = 0; i < childNodes.getLength(); i++) {
            Node childNode = childNodes.item(i);
            printNode(childNode, indent + 1);
        }
    }
    NamedNodeMap attributes = node.getAttributes();
    if (attributes != null) {
        for (int i = 0; i < attributes.getLength(); i++) {
            Node attr = attributes.item(i);
            System.out.println(indentString + "\tAttribute: " + attr.getNodeName() + " = " +
attr.getNodeValue());
        }
    }
}
```

A teljes kód linkje : Github Link

2.3 DomQuery megvalósítása / adatlekérdezés :

A DomQueryIKXS9J osztály célja az XML-fájlok adatainak lekérdezése és kiolvasása DOM API segítségével. A tervezés során a fő szempont az volt, hogy különböző entitások, mint

például *Felhasználók*, *Raktárak*, *Események*, *Bérlések* és *Kedvezmények* adatait könnyedén elérhessük és kiírhassuk a konzolra.

A DOM alapú megközelítés lehetővé teszi az XML-struktúra elemzését, az adatok hierarchikus bejárását és azok feldolgozását.

Fontos kódrészlet és magyarázat :

Az alábbi kódrészlet egy felhasználó adatainak lekérdezését mutatja be az XML-fájl első *Felhasznalo* eleméből:

```
NodeList felhasznalok = document.getElementsByTagName("Felhasznalo");

if (felhasznalok.getLength() > 0) {

    Element felhasznalo = (Element) felhasznalok.item(0);

    String id = felhasznalo.getElementsByTagName("id").item(0).getTextContent();

    String nev = felhasznalo.getElementsByTagName("nev").item(0).getTextContent();

    String email = felhasznalo.getElementsByTagName("email").item(0).getTextContent();

    String neme = felhasznalo.getElementsByTagName("neme").item(0).getTextContent();

    System.out.println("Felhasználó: " + id + ", " + nev + ", " + email + ", " + neme);

}
```

Link a teljes kódhoz : Github Link

2.4 DomModify megvalósítása / adatmódosítás :

A DomModifyIKXS9J osztály az XML dokumentum adatmódosítására szolgál, a DOM (Document Object Model) segítségével. Ez az osztály az XML fájlban található elemek értékeit módosítja, majd a frissített tartalmat visszamenti az eredeti fájlba. Az alábbi példák szemléltetik az egyes elemek módosítását:

Fontos kód kiemelése és magyarázata

Felhasználó módosítása:

```
NodeList felhasznalok = document.getElementsByTagName("Felhasznalo");

for (int i = 0; i < felhasznalok.getLength(); i++) {

    Node node = felhasznalok.item(i);

    if (node.getNodeType() == Node.ELEMENT_NODE) {

        Element felhasznalo = (Element) node;

        String id = felhasznalo.getElementsByTagName("id").item(0).getTextContent();
```

```
      if (id.equals("1")) {
felhasznalo.getElementsByTagName("nev").item(0).setTextContent("Új Név");
felhasznalo.getElementsByTagName("email").item(0).setTextContent("ujemail@example.com");
           System.out.println("Felhasználó módosítva.");
        }
    }
}
```

Az XML-ből lekérdezzük az összes Felhasznalo elemet.
Az id elem értékét ellenőrizzük, és ha az megegyezik az előre meghatározott értékkel (pl. "1"), a felhasználó nevét és e-mail címét módosítjuk.

Dokumentum mentése:

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(inputFile); // Eredeti fájl felülírása
transformer.transform(source, result);
System.out.println("XML dokumentum sikeresen módosítva és mentve.");
```

A Transformer segítségével a memóriában módosított DOM struktúrát visszaírjuk az eredeti XML fájlba.

Tervezési megfontolások

1. A módosításokat ID-k alapján hajtjuk végre, amely egyedi azonosító minden elemhez.
2. A kód különböző elemek (Felhasznalo, Raktar, Esemeny) módosítását támogatja, így jól strukturált és könnyen bővíthető más elemek kezelésére.

Link a teljes kódhoz : Github Link