Jegyzőkönyv

Adatkezelés XML-ben

Féléves feladat

Autókölcsönző nyilvántartás

Készítette: **Énekes Szabolcs Dániel**

Neptunkód: **A86ANU**

Dátum: **2025. novmember 28.**

Miskolc, 2025

Tartalom

Bevezetés

A beadandó feladat célja egy XML alapú adatkezelő rendszer megtervezése és megvalósítása, amely az XML technológiák gyakorlati alkalmazását mutatja be. A projekt során olyan adatmodellt kellett készíteni, amely több egységet, kapcsolatot, tulajdonságot és szabályt tartalmaz, és alkalmas valódi információk strukturált tárolására. A feladat keretében először egy ER modellt kellett elkészíteni, majd azt XDM modellé konvertálni, amely már az XML dokumentum fa-struktúráját tükrözi.

A projekt további része az XML dokumentum tényleges előállítása volt konkrét adatokkal, valamint az adatmodellhez tartozó XSD séma megírása, amely biztosítja az XML állomány szerkezetének és adattípusainak érvényességét. Ezek után DOM alapú Java programokat kellett létrehozni, amelyek képesek az XML dokumentum beolvasására, lekérdezésére és módosítására.

A feladat leírása

A feladat egy olyan XML alapú adatnyilvántartó rendszer elkészítése volt, amely egy valósághű, több egységet és kapcsolatot tartalmazó adatmodellt jelenít meg. A rendszer témáját szabadon lehetett megválasztani. Jelen dokumentumban egy autókölcsönző rendszer modellje került kidolgozásra. A feladat előírásai szerint az ER modellnek legalább öt különböző egyedet kellett tartalmaznia, többféle kapcsolattal (1:1, 1:N és N:M). Az egyedeknek többféle adattípust kellett tartalmazniuk: normál, kulcs, összetett, illetve többértékű attribútumokat.

Az ER modell elkészítése után a következő lépés annak XDM modellé alakítása volt. Az XDM modell lényegében az XML fájl szerkezetének logikai terve. Ebben meg kellett határozni a gyökérelemet, az alárendelt elemek hierarchiáját, valamint a kapcsolatok reprezentálását ID és IDREF attribútumokkal. A feladat kikötése szerint a kapcsolati vonalak nem keresztezhették egymást, ami a diagram vizuális megszerkesztésében külön figyelmet igényelt.

A harmadik lépés az XML dokumentum létrehozása volt konkrét, kitöltött adatokkal. A dokumentumnak a teljes XDM modell struktúrájára épült, és legalább két példányt kellett tartalmaznia minden többször előforduló elemcsoportból. Az XML-t validálhatóvá kellett tenni, ezért XSD sémát is készíteni kellett hozzá, amely meghatározza az elemek nevét, adattípusát, kötelezőségét, attribútumokat, valamint az egyszerű és komplex típusokat.

A következő feladat egy háromrészes Java program elkészítése volt DOM technológia alkalmazásával. A „Read” program feladata az XML dokumentum beolvasása és feldolgozása, valamint a fontosabb adatok kiírása volt. A „Query” programban legalább négy különböző

lekérdezést kellett megvalósítani, amelyek az XML dokumentumból származó adatokat jelenítettek meg feltételek alapján. A „Modify” programban pedig legalább négy különböző módosítást kellett elvégezni az XML dokumentumon, például új elem hozzáadása, egy meglévő elem adatainak módosítása vagy egy elem törlése. A módosítások eredményét szintén a konzolra kellett kiírni, valamint a módosított XML-t új fájlba kellett menteni.

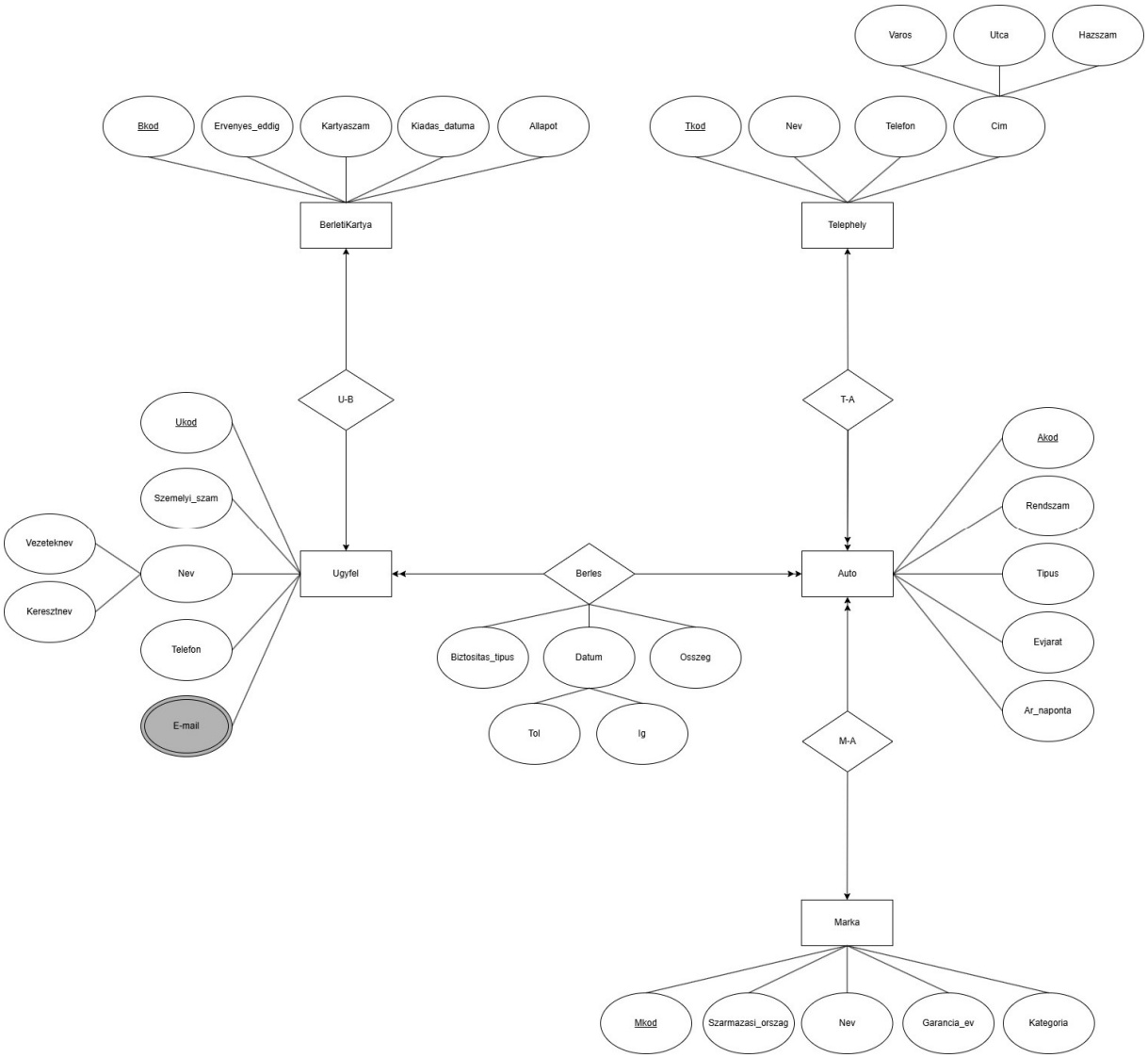
Összességében a feladat teljesítéséhez az adatmodellezés, az XML struktúra és XSD sématervezés, valamint a Java DOM feldolgozás együttes ismeretére volt szükség. A dokumentum további részei részletesen bemutatják a modell elkészítésének lépéseit, a létrehozott XML és XSD állományokat, valamint a Java programok működését.

1. Autókölcsönző XML alapú adatnyilvántartó rendszere

Ez a fejezet az adatbázis logikai felépítését bemutató ER modell megtervezésével foglalkozik. A feladat célja egy olyan adatszerkezet kialakítása volt, amely alkalmas egy autókölcsönző vállalat működésének pontos, szabványos és konzisztens leírására. Az ER modell az adatbázis koncepcionális szintű megjelenítését szolgálja, és meghatározza a főbb egyedeket, azok tulajdonságait, valamint az egyedek közötti kapcsolatokat. A modell alapját képező entitások és kapcsolatok később kiindulópontként szolgáltak az XDM modell, az XML dokumentum és az XSD séma létrehozásához.

1.1 Az adatbázis ER modell tervezése

Az ER modell megtervezése során első lépésként meghatároztam, hogy az autókölcsönző rendszer működéséhez mely egyedek és mely kapcsolatok szükségesek. A feladat előírása szerint a modellnek legalább öt egyedet kellett tartalmaznia, továbbá többféle kapcsolatot (1:1, 1:N, N:M), különféle adattípusokat és tulajdonságcsoportokat, például normál, kulcs, összetett és többértékű attribútumok.



1. ábra: Az ER modell

A rendszerben az alábbi főbb egyedeket azonosítottam:

- Ügyfél (Ugyfel) – az autókölcsönző szolgáltatást igénybe vevő személy. Tartalmaz személyes adatokat, összetett név attribútumot, valamint többértékű e-mail címlistát.
- BérletKártya (BerletKartya) – opcionális azonosító kártya, amely kedvezményeket biztosíthat. Egy ügyfélnek egy bérletkártyája lehet (1:1 kapcsolat).
- Autó (Auto) – a kölcsönözhető járművek. Egy autóhoz tartozik ára, típusa, évjárata és rendszáma.
- Márka (Marka) – az autók gyártója. Egy márkához több autó is tartozhat (1:N kapcsolat).
- Telephely (Telephely) – ahol az autók fizikailag megtalálhatók. Egy telephelyen több autó lehet (1:N kapcsolat)

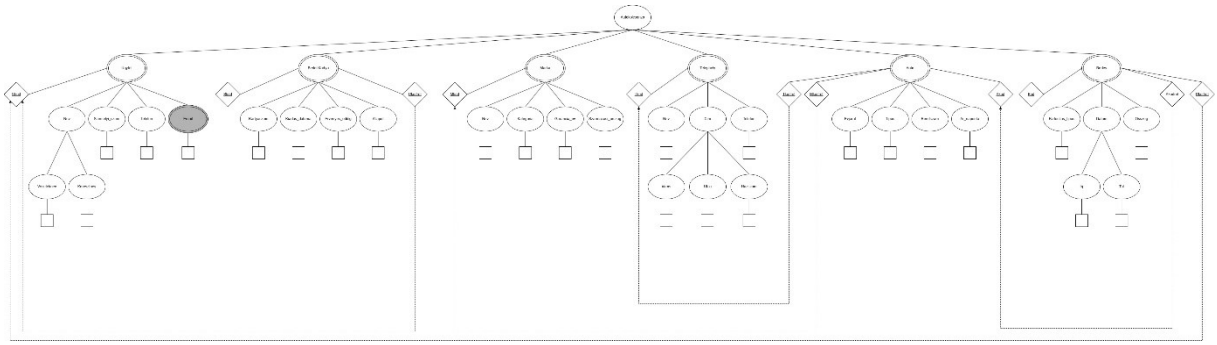
- Bérlés (Berles) – az Ügyfél és Autó közötti N:M kapcsolat megvalósítása. A bérlés rendelkezik saját attribútumokkal (dátum, időtartomány, összeg, biztosítás típusa).

A modell kialakításánál fontos szempont volt, hogy minden kapcsolat megfelelően tükrözze a valós működést. A Bérlés kapcsolat azért különösen lényeges, mert ez biztosítja az N:M kapcsolatot az Ügyfél és az Autó között, és egyben tárolja a bérléshez kapcsolódó fontos adatokat is. Ez a kapcsolat tehát saját, önálló attribútumokkal rendelkező összekötő entitásként jelenik meg.

A tervezés során összetett attribútumként került meghatározásra az ügyfél neve (Vezetéknév + Keresztnév), valamint a telephely címe (Város, Utca, Házszám). Többértékű attribútum a többször megadható e-mail címek listája. Minden egyed esetében meghatározásra került a kulcsattribútum (pl. ukod, akod mkod), amely biztosítja az egyedi azonosítást.

1.2 Az adatbázis konvertálása XDM modellre

Az ER modell elkészítése után a következő lépés az adatszerkezet XDM modellé alakítása volt. Az XDM feladata az, hogy az ER diagram logikai felépítését egy XML-ben használható, hierarchikus struktúrává alakítsa.



2. ábra: Az XDM modell

A modell kialakításánál ügyeltem arra, hogy a fa-struktúra egyértelműen tükrözze az ER modell kapcsolatait, valamint hogy az összetett attribútumok (pl. név, cím) külön al-elemként jelenjenek meg. A bérlés kapcsolat önálló adattartalommal rendelkező elemként került ábrázolásra, mivel több attribútumot is tartalmaz. A diagram szerkesztése során szabványos jelölést alkalmaztam, és figyeltem arra, hogy a kapcsolati (PK-FK) vonalak keresztezzék egymást, megfelelve a feladat formai előírásának.

1.3 Az XDM modell alapján XML dokumentum készítése

Az XDM modell elkészítése után a következő lépés az XML dokumentum létrehozása volt a modell szerkezetének megfelelően. A gyökérelemet (autokolcsonzo) alárendelt gyűjtőelemekkel egészítettem ki, amelyek az XDM modell hierarchiáját követik. A dokumentumban minden többször előforduló csoportból legalább két példányt hoztam létre,

hogy az XML szerkezete teljes értékű adatokat tartalmazzon, és alkalmas legyen a DOM alapú lekérdezések és módosítások kipróbálására is. A kód itt érhető el.

```
<autokolcsonzo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="A86ANU_XMLSchema.xsd">
```

- Az autokolcsonzo a teljes dokumentum gyökéreleme, minden más ebben van.
- Az xmlns:xsi névtér deklarációval engedem, hogy az xsi: előtaggal XSD-hez kapcsolódó attribútumokat használjak.
- Az xsi:noNamespaceSchemaLocation=”A86ANU_XMLSchema.xsd” megmondja, hogy melyik XSD séma ellenőrizze ezt az XML-t.

```
<ugyfelek>


    <!-- Egy ügyfél adatai -->
    <ugyfel ukod="U001">
        <nev>
            <vezeteknev>Kiss</vezeteknev>
            <keresztnev>János</keresztnev>
        </nev>
        <szemelyi_szam>123456AB</szemelyi_szam>
        <telefon>06201234567</telefon>

        <!-- Többértékű adat: több email cím is lehet -->
        <emailek>
            <email>janos.kiss@gmailcom</email>
            <email>j.kiss@munkahely.hu</email>
        </emailek>
    </ugyfel>
```

…

</ugyfelek>

- Az <ugyfelek> egy gyűjtőelem, benne több <ugyfel> ismédlődhet.
- Az ukod=”U001” egy azonosító attribútum, amire később más elemek hivatkoznak.
- A nev egy összetett alstruktúra, ami jóltükrözi az ER-diagramos felbontást.
- Az emailek alatt több <email> elem lehet, ez tipikus többértékű adat XML-ben.

```
<!--

        ============================

        BÉRLETKÁRTYÁK

        Kapcsolat: 1 ügyfél → N kártya

        ukodref az Ügyfélre mutat

        ============================

    -->

    <berletkartyak>


        <berletkartya bkod="B001" ukodref="U001">

            <kartyaszam>11112222</kartyaszam>

            <kiadas_datuma>2024-01-01</kiadas_datuma>

            <ervenyes_eddig>2026-01-01</ervenyes_eddig>

            <allapot>ervenyes</allapot>

        </berletkartya>

…

</berletkartyak>
```

- A bkod a bérletkártya saját azonosítója.
- ukodref=”U001” hivatkozik az ugyfel elem ukod attribútumára.

```
<!--
    ============================
    TELEPHELYEK
    Ezekre hivatkozik az <auto> @tkodref
    ============================
-->
<telephelyek>

    <telephely tkod="T001">
        <nev>Belvárosi telephely</nev>
        <telefon>061234567</telefon>
        <cim>
            <varos>Budapest</varos>
            <utca>Fő utca</utca>
            <hazszam>10</hazszam>
        </cim>
    </telephely>
...
</telephelyek>
```

- A cim elem egy önálló, összetett adat a telephelyen belül.
- A varos, utca, hazszam mezőket nem követlenül a telephely alatt tárolom, hanem csoportosítom őket egy cím elem alá.

```
<!--
```

```
        ==========================

        AUTÓK

        Kapcsolatok:

        - @mkodref → márka

        - @tkodref → telephely

        ==========================

    -->

    <autok>


        <auto akod="A001" mkodref="M001" tkodref="T001">

            <rendszam>ABC-123</rendszam>

            <tipus>szemelyauto</tipus>

            <evjarat>2020</evjarat>

            <ar_naponta>15000</ar_naponta>

        </auto>

…

</autok>
```

- Az akod az autók egyedi azonosítója.
- Az mkodref a márkára, a tkodref a telephelyre mutat.

1.4 Az XML dokumentum alapján XMLSchema készítése

Az XML dokumentum elkészítése után következő lépésként létrehoztam az XSD sémát, amely az XML szerkezetének, adattípusainak és kapcsolódási szabályainak formális leírását adja. A séma tartalmaz saját egyszerű típusokat, valamint összetett típusokat, amelyek az XML elemek felépítését írják le. Az ER és XDM modellekben meghatározott elsődleges kulcsokat az XML-ben xs:ID, a külső kulcsokat pedig xs:IDREF segítségével valósítottam meg, így a

séma biztosítja a hivatkozások konzisztenciáját. A bonyolultabb elemekhez külön komplex típusokat definiáltam. A kód itt érhető el.

```
<!--
    ============================
    EGYSZERŰ TÍPUSOK (pattern)
    ============================
-->

<!-- Kód típus: általános azonosítókhoz -->
<xs:simpleType name="kodtype">
    <xs:restriction base="xs:string">
        <xs:pattern value="[A-Z0-9]{3,10}" />
    </xs:restriction>
</xs:simpleType>

<!-- Rendszám formátum: ABC-123 -->
<xs:simpleType name="rendszamtype">
    <xs:restriction base="xs:string">
        <xs:pattern value="[A-Z]{3}-[0-9]{3}" />
    </xs:restriction>
</xs:simpleType>
```

- Mindkettő xs:simpleType, azaz egyszerű típus.
- A kodtype egy általános azonosító, ami csak nagybetű és szám, és 3-10 karakter hosszú.
- A rendszamtype kifejezetten rendszámra van kihegyezve, 3 nagybetű, kötőjel 3 szám.

```
<!--
    ============================
    KOMPLEX TÍPUSOK
    Ezek definiálják az elemek szerkezetét
    ============================
-->

<!-- Ügyfél neve összetett adat -->
<xs:complexType name="nevtype">
    <xs:sequence>
        <xs:element name="vezeteknev" type="xs:string" />
        <xs:element name="keresztnev" type="xs:string" />
    </xs:sequence>
</xs:complexType>

<!-- Cím összetett adat -->
<xs:complexType name="cimtype">
    <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>
```

- A nevtype és a cimtype újra felhasználható összetett típusok: nevtype = vezetéknév + keresztnév, cimtype = város + utca + házszám

```
<!-- Ügyfél típusa -->
<xs:complexType name="ugyfeltype">
    <xs:sequence>
        <xs:element name="nev" type="nevtype" />
        <xs:element name="szemelyi_szam" type="xs:string" />
        <xs:element name="telefon" type="xs:string" />
        <xs:element name="emailek">
            <xs:complexType>
                <xs:sequence>
                                    <xs:element  name="email"  type="xs:string"
maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="ukod" type="xs:ID" use="required" />
</xs:complexType>
```

- Az emailek alatt egy névtelen complexType, benne email maxOccurs=”unbounded”, azaz tetszőleges számú email lehet.

```
<!-- Bérletkártya -->
<xs:complexType name="berletkartyatype">
```

```
    <xs:sequence>
        <xs:element name="kartyaszam" type="xs:string" />
        <xs:element name="kiadas_datuma" type="xs:date" />
        <xs:element name="ervenyes_eddig" type="xs:date" />
        <xs:element name="allapot" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="bkod" type="xs:ID" use="required" />
    <xs:attribute name="ukodref" type="xs:IDREF" use="required" />
</xs:complexType>
```

- A bérletkártya önálló entitás, saját adatokkal.
- ukodref – IDREF, azaz, hivatkozás egy xs:ID értékre (itt az ügyfél ukod-jára).

```
<!-- Autó -->
<xs:complexType name="autotype">
    <xs:sequence>
        <xs:element name="rendszam" type="rendszamtype" />
        <xs:element name="tipus" type="xs:string" />
        <xs:element name="evjarat" type="xs:int" />
        <xs:element name="ar_naponta" type="xs:int" />
    </xs:sequence>
    <xs:attribute name="akod" type="xs:ID" use="required" />
    <xs:attribute name="mkodref" type="xs:IDREF" use="required" />
    <xs:attribute name="tkodref" type="xs:IDREF" use="required" />
</xs:complexType>
```

- Az akod az egyedi ID, az mkodref és a tkodref hivatkoznak a márkára és a telephelyre.

```
<!--
        ============================
        GYŰJTŐ TÍPUSOK (LISTÁK)
        ============================
    -->

    <xs:complexType name="ugyfelektype">
        <xs:sequence>
            <xs:element name="ugyfel" type="ugyfeltype" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="berletkartyaktype">
        <xs:sequence>
            <xs:element        name="berletkartya"        type="berletkartyatype"
maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="markaktype">
        <xs:sequence>
            <xs:element name="marka" type="markatype" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
```

```
    <xs:complexType name="telephelyektype">
        <xs:sequence>
            <xs:element name="telephely" type="telephelytype" maxOccurs="unbounded"
/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="autoktype">
        <xs:sequence>
            <xs:element name="auto" type="autotype" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="berlesektype">
        <xs:sequence>
            <xs:element name="berles" type="berlestype" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>

    <!--
        ============================
        GYÖKÉRELEM DEFINÍCIÓJA
```

```
    ==========================

-->


<xs:element name="autokolcsonzo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ugyfelek" type="ugyfelektype" />
            <xs:element name="berletkartyak" type="berletkartyaktype" />
            <xs:element name="markak" type="markaktype" />
            <xs:element name="telephelyek" type="telephelyektype" />
            <xs:element name="autok" type="autoktype" />
            <xs:element name="berlesek" type="berlesektype" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- A *type végű struktúrák mind gyűjtő/lista típusok, bennük a megfelelő elem unbounded.
- A gyökérelem, az autokolcsonzo tartalmazza az összes fő „táblát”.

2. DOM kezelés Java-ban

2.1 Adatolvasás

Az adatolvasást az A86ANUDOMRead.java valósítja meg, amely a DOM parser segítségével betölti és feldolgozza az A86ANU_XML.xml dokumentumot. A program első lépésként beolvassa a teljes XML állományt egy Document objektumba, majd normalizálja azt a konzisztens feldolgozás érdekében. A feldolgozás során a program végigiterál az XML minden fontos elemén, és kiírja a konzolra az összes releváns adatot jól olvasható blokkokba rendezve. A kód itt érhető el.

```
-               // ------------------------------------------------
-               // XML FÁJL BETÖLTÉSE DOM-MODELBE
-               // ------------------------------------------------
-               // A File objektum csak hivatkozik az XML fájlra.
-               File xmlFile = new File("A86ANU_XML.xml");
-
-               // DocumentBuilderFactory létrehozása - ez készíti elő a DOM
   parsert.
-               DocumentBuilderFactory                dbFactory                =
   DocumentBuilderFactory.newInstance();
-               dbFactory.setNamespaceAware(true);   // névterek felismerésének
   engedélyezése
-
-               // A tényleges DOM parser példány
-               DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
-
-               // A dokumentum beolvasása és DOM (fa-struktúra) létrehozása
-               Document doc = dBuilder.parse(xmlFile);
-
-               // Normalizálás: whitespace-ek, szövegtöredékek egyesítése
-               doc.getDocumentElement().normalize();
```

Itt a kód betölti az XML-t egy DOM fa struktúrába, előkészíti a feldolgozásra, és normalizálja a dokumentumot a megbízható bejáráshoz.

```
// ================================================
// ÜGYFELEK BLOKK - <ugyfel> elemek feldolgozása
// ================================================
private static void printUgyfelek(Document doc) {
    System.out.println("===== ÜGYFELEK =====");
```

```
        // Összes <ugyfel> elem lekérése NodeList formában
        NodeList ugyfelList = doc.getElementsByTagName("ugyfel");
        System.out.println("Ügyfelek száma: " + ugyfelList.getLength());
        System.out.println("-----------------------------");

        // Bejárjuk az összes ügyfelet
        for (int i = 0; i < ugyfelList.getLength(); i++) {

            // Az aktuális <ugyfel> elemre hivatkozás
            Element ugyfel = (Element) ugyfelList.item(i);

            // Attribútum lekérése (ukod)
            String ukod = ugyfel.getAttribute("ukod");

            // Név beágyazott elemének feldolgozása
            Element nevElem = (Element) ugyfel.getElementsByTagName("nev").item(0);
            String vezetek =
nevElem.getElementsByTagName("vezeteknev").item(0).getTextContent();
            String kereszt =
nevElem.getElementsByTagName("keresztnev").item(0).getTextContent();

            // Személyi szám, telefon
            String szemelyi =
ugyfel.getElementsByTagName("szemelyi_szam").item(0).getTextContent();
```

```
            String                    telefon                                    =
ugyfel.getElementsByTagName("telefon").item(0).getTextContent();

            // Konzolos kiírás blokk formában
            System.out.println("Ügyfél [" + ukod + "]");
            System.out.println("  Név       : " + vezetek + " " + kereszt);
            System.out.println("  Szem. sz. : " + szemelyi);
            System.out.println("  Telefon   : " + telefon);

            // Email lista feldolgozása (több <email> elem is lehet)
            NodeList emailLista = ugyfel.getElementsByTagName("email");
            for (int j = 0; j < emailLista.getLength(); j++) {
                String email = emailLista.item(j).getTextContent();
                System.out.println("  Email     : " + email);
            }

            System.out.println();
        }

        System.out.println();
    }
```

- A metódus kiolvassa a dokumentumból az összes <ugyfel> elemet, majd mindegyiket bejárja.

- Az attribútumokat és a beágyazott adatokat kiolvassa és blokk formájában kiírja a konzolra.

```
    // ================================================
    // XML DOKUMENTUM MENTÉSE
    // ================================================
    private static void saveDocument(Document doc, String fileName) throws Exception
{

        TransformerFactory tf = TransformerFactory.newInstance();
        Transformer t = tf.newTransformer();

        // Kimeneti formázás beállítása
        t.setOutputProperty(OutputKeys.INDENT, "yes"); // behúzás
        t.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");
        t.setOutputProperty(OutputKeys.ENCODING, "UTF-8");

        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(new File(fileName));

        // A DOM → XML fájl generálása
        t.transform(source, result);
    }
```

- A metódus egy Transformer segítségével szépen formázott XML-fájlt készít a memória-beli DOM dokumentumból.

2.2 Adat-lekérdezés

Az adatlekérdezést az A86ANUDOMQuery.java osztály valósítja meg, amely a DOM API-ra építve, xPath kifejezések használata nélkül hajtja végre a szükséges lekérdezéseket az XML dokumentumban. A program első lépésként betölti az A86ANUXML.xml állományt, majd különféle feltételek alapján végigiterál az elemek listáin, és manuálisan vizsgálja meg attribútumokat, alárendelt elemeket és értékeket. A lekérdezések között szerepel például az autók ár szerinti szűrése, egy adott ügyfélhez tartozó bérlések kigyűjtése, egy telephelyen található autók listázása, valamint azoknak az ügyfeleknek a meghatározása, akikhez tartozik bérletkártya. Minden lekérdezés a konzolra írja a találatokat jól áttekinthető formában. A kód itt érhető el.

```
// ------------------------------------------------

// XML DOKUMENTUM BETÖLTÉSE DOM-MODELBE

// ------------------------------------------------

Document doc = loadDocument("A86ANU_XML.xml");


// 1. lekérdezés: drága autók (min. ár alapján)

queryDragaAutok(doc, 16000);

System.out.println("=======================================");


// 2. lekérdezés: egy adott ügyfél összes bérlése

queryBerlesekUgyfelSzerint(doc, "U001");

System.out.println("=======================================");


// 3. lekérdezés: autók egy megadott telephelyen

queryAutokTelephelySzerint(doc, "T001");

System.out.println("=======================================");
```

```
        // 4. lekérdezés: azon ügyfelek, akiknek van bérletkártyájuk
        queryUgyfelekBerlettel(doc);
```

A program először betölti az XML fájlt egy DOM-modellbe, majd egymás után lefuttat négy különböző lekérdezést.

Kilistázza a megadott összegnél drágább autókat

Lekérdezi egy adott ügyfél összes bérlését

Megjeleníti, hogy egy kijelölt telephelyen milyen autók találhatóak

Kiírja azoknak az ügyfeleknek a nevét, akikhez tartozik legalább egy bérletkártya

2.3 Adatmódosítás

Az XML dokumentum módosítását az A86ANUDOMModify.java osztály végzi, amely a DOM API segítségével új elemeket hoz létre, meglévő elemek értékeit módosítja, illetve adott csomópontokat töröl a dokumentumból. A program első lépésként beolvassa az A86ANUXML.xml állományt, majd külön metódusok segítségével hajtja végre az egyes módosítási műveleteket. A feladat követelményeinek megfelelően összesen négy változtatást valósítottam meg: új ügyfél hozzáadása, új autó felvétele, egy autó napi árának megváltoztatása, valamint meglévő bérlés törlése. Minden módosítás után a program konzolra kiírja, hogy milyen elem, milyen adatokkal változott meg. A kód itt érthető el.

```
// ------------------------------------------------
 // 1. MÓDOSÍTÁS - ÚJ ÜGYFÉL HOZZÁADÁSA
 // ------------------------------------------------
 private static void addUjUgyfel(Document doc) {

     System.out.println("1. módosítás: új ügyfél hozzáadása");

     // Gyökérelem: <autokolcsonzo>
     Element gyoker = doc.getDocumentElement();
```

```
        // <ugyfelek> elem keresése
        Element ugyfelek = (Element) gyoker.getElementsByTagName("ugyfelek").item(0);

        // Új <ugyfel> elem létrehozása
        Element uj = doc.createElement("ugyfel");
        uj.setAttribute("ukod", "U005");  // új ügyfél azonosító

        // <nev> összetett elem
        Element nev = doc.createElement("nev");

        Element vez = doc.createElement("vezeteknev");
        vez.setTextContent("Szabó");

        Element ker = doc.createElement("keresztnev");
        ker.setTextContent("Péter");

        nev.appendChild(vez);
        nev.appendChild(ker);

        // Egyéb adatok
        Element szemelyi = doc.createElement("szemelyi_szam");
        szemelyi.setTextContent("555555EF");
```

```
        Element tel = doc.createElement("telefon");
        tel.setTextContent("06301234567");

        Element emailek = doc.createElement("emailek");
        Element email = doc.createElement("email");
        email.setTextContent("peter.szabo@pelda.hu");
        emailek.appendChild(email);

        // Az <ugyfel> felépítése
        uj.appendChild(nev);
        uj.appendChild(szemelyi);
        uj.appendChild(tel);
        uj.appendChild(emailek);

        // Hozzáadás az <ugyfelek> listához
        ugyfelek.appendChild(uj);

        System.out.println("   Új ügyfél felvéve: ukod=U005, név=Szabó Péter");
        System.out.println();
    }
```

A metódus feladata, hogy egy teljesen új <ugyfel> elemet hozzon létre, feltöltse a szükséges adatokkal, majd hozzáadja azt az XML dokumentum ügyfél listájához.

Először megkeresi a gyökérelemből az <ugyfelek> szekciót, majd létrehoz egy új <ugyfel> elemet, beállítja az azonosítóját, felépíti a benne található összetett nev elemet, valamint a személyi számot, a telefonszámot és az email címet.

Miután az új ügyfél elem teljesen összeállt, a metódus hozzáfűzi az <ugyfelek> lista végéhez, így az új ügyfél bekerül a DOM dokumentumba.