JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

Járműkölcsönző cég nyilvántartás

Készítette: **Ardon Milán**

Neptunkód: **E00XC3**

Dátum: **2025. december**

Miskolc, 2025

Tartalomjegyzék

Bevezetés

A beadandó feladat elkészítése során járműkölcsönző cégek adatainak nyilvántartásával foglalkoztam, amelynek célja egy áttekinthető, logikusan felépített adatbázis megtervezése és megvalósítása volt. Azért választottam ezt a témát, mert a járműkölcsönzők működése jól szemlélteti a valós életben előforduló összetett adatkapcsolatokat.

A feladat első lépéseként elkészítettem az ER modellt és annak XDM megfelelőjét, majd ezekre építve létrehoztam az XML dokumentumot és a hozzá tartozó sémát. A második részében pedig DOM-alapú módszerekkel valósítottam meg az adatállomány beolvasását, lekérdezését és módosítását.

A feladat leírása

A feladat során egy járműkölcsönző cég működéséhez kapcsolódó adatnyilvántartási rendszer alapjainak megtervezését és kidolgozását végeztem el, amelynek célja egy olyan struktúra kialakítása volt, amely jól modellezi a járműkölcsönzéshez kapcsolódó alapvető adatelemeket és azok összefüggéseit.

A tervezési folyamat első lépése az ER modell elkészítése volt, amelyben egy központi Kölcsönző egyed köré szerveztem a további adatcsoportokat, majd ezt hierarchikus XDM modellre konvertáltam.

Az ER modellt és az XDM modellt a draw.io felületén készítettem el, mivel ez a szerkesztő átláthatóan támogatja a szabványos adatmodellezési jelöléseket, és jól lehet vele komplex kapcsolati hálókat szemléltetni.

A grafikus tervezést követően elkészítettem az XML dokumentumot, amelyet Visual Studio Code-ban állítottam össze, a modellek alapján kialakított elemekkel és megfelelő kommentekkel, valamint minden ismétlődő elemhez két példánnyal.

Az XML szerkezetét ezután XML Schema segítségével formalizáltam, ahol saját típusokat, kulcsokat, hivatkozásokat és összetett elemeket definiáltam az adatok konzisztenciájának biztosításához.

A feladat végső részében az elkészített XML dokumentumot DOM alapú feldolgozással kellett kezelni. Ennek során Eclipse környezetben külön programokat hoztam létre az adatolvasás, az adatlekérdezés és az adatmódosítás megvalósítására az XML struktúrában található adatokhoz.

1. ER, XDM, XML és XMLSchema elkészítése

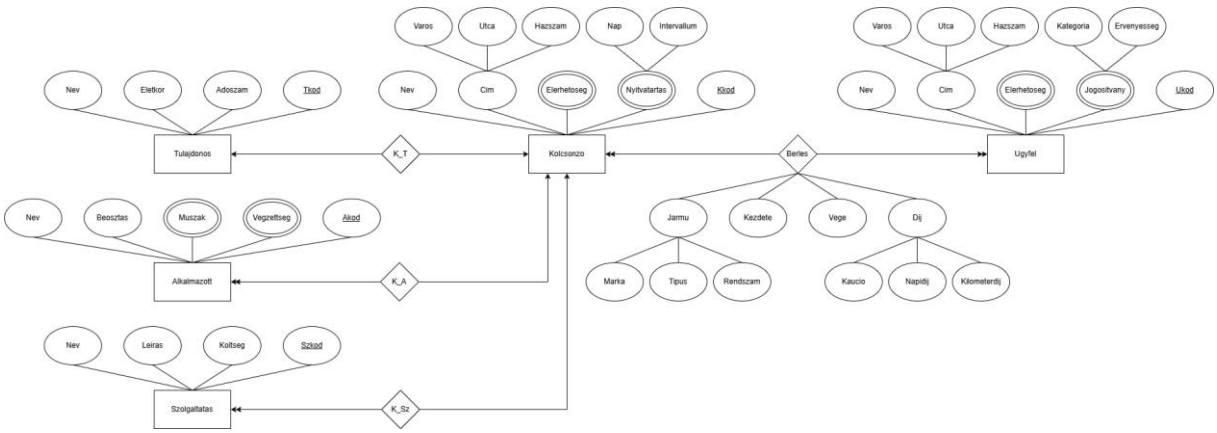
1.1 Az adatbázis ER modell tervezése

A modell központi elemeként a **Kölcsönző** egyedet hoztam létre, mivel ez fogja össze a rendszer többi adatcsoportját, és ez biztosítja a megfelelő struktúrát a kapcsolatok kialakításához. A többi egyedet ehhez az alaphoz kapcsoltam, különféle kapcsolatfajták alkalmazásával, hogy a valós működés sajátosságait minél pontosabban visszaadjam. Az elkészült ER diagram a fejezet végén található (**1. ábra**), amely jól szemlélteti a teljes modell felépítését és az egyedek közötti kapcsolatokat.

A **Kölcsönző – Tulajdonos** kapcsolatot **1:1 típusúra** terveztem, mivel feltételezésem szerint egy kölcsönzőhöz egyetlen tulajdonos tartozik, és fordítva is, tehát ez egy szoros, kizárólagos kapcsolat. A kölcsönző működésének további meghatározó résztvevői az **Alkalmazottak**, akik a mindennapi ügymenetet végzik, ezért itt **1:N kapcsolatot** alakítottam ki: egy kölcsönzőhöz több alkalmazott tartozhat, de minden alkalmazott csak egy adott kölcsönzőnél dolgozik.

A modellben fontos szerepet kapnak a **Szolgáltatások** is, amelyek a kölcsönző által nyújtott extra lehetőségeket reprezentálják (utánfutó bérlés vagy egyéb kiegészítő szolgáltatások). Ezeket szintén **1:N kapcsolatban** kötöttem a Kölcsönzőhöz, hiszen egy kölcsönző többféle szolgáltatást kínálhat, de egy szolgáltatás csak egy adott kölcsönzőhöz tartozik.

A rendszer egyik legösszetettebb része az **Ügyfél** és a járműkölcsönzés folyamata. Az Ügyfél és a Kölcsönző között **N:M típusú kapcsolatot** hoztam létre, amelyet egy külön **Bérlés** kapcsolóentitással oldottam meg. Ez az egyed saját attribútumokat is tartalmaz, mint a bérlés időtartama (mettől meddig tart), illetve a különböző díjak (kaució, napidíj, kilométerdíj).



1. ábra: ER modell

1.2 Az adatbázis konvertálása XDM modellre

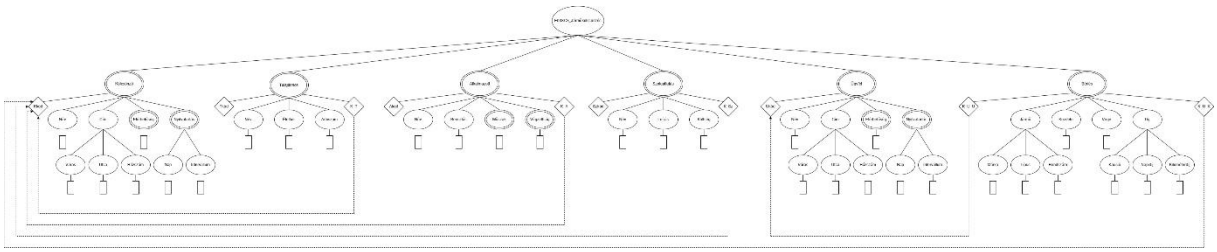
Az ER modell elkészítése után a következő lépés az volt, hogy a rendszer szerkezetét XML-orientált formában is ábrázoljam, vagyis elkészítsem az XDM (XML Data Model) modellt, ami **egyfajta átmenetet képez az ER diagram és a tényleges XML fájl között**, és segít abban, hogy a hierarchikus szerkezet áttekinthető és következetes legyen.

Az XDM modellben már nem jelöltem külön központi egyedet, mivel a teljes adatállomány egy közös **gyökérelem** alá került. Az egyedek így egymás mellett jelennek meg, önálló XML-komponensként, viszont a közöttük lévő kapcsolatok továbbra is lekövetik a korábbi logikai összefüggéseket.

Ennek megfelelően a Kölcsönző, Tulajdonos, Alkalmazott, Szolgáltatás és Ügyfél elemek különálló szinten szerepelnek, a **kapcsolatokat pedig PK–FK jelölésekkel** jelenítettem meg, ügyelve arra, hogy a vonalak ne keresztezzék egymást.

A legösszetettebb kapcsolat az Ügyfél és a Kölcsönző között fennálló N:M kapcsolat, amelyet – az ER modellhez hasonlóan – külön **Bérlés egyeddel** oldottam meg. Ez az elem az XDM-ben is önálló entitásként szerepel, és kétirányú kapcsolattal kötődik mind az Ügyfélhez, mind a Kölcsönzőhöz.

A Bérléshez tartozó attribútumokat (például a bérlés időtartama és a különböző díjak) ugyancsak beépítettem a modellbe, közvetlen az egyed alá ahogy az elkészült XDM modell (**2.ábra**) egyértelműen mutatja.



2. ábra: XDM modell

1.3 Az XDM modell alapján XML dokumentum készítése

A feladat célja az volt, hogy a korábban elkészített XDM modell alapján létrehozzam a járműkölcsönző rendszer XML állományát. Az XDM elvárásainak megfelelően a dokumentum hierarchiája nem az adatbázis-szerkezetet követi, hanem egy **jarmukolcsonzes** gyökérelem alatt sorolja fel az összes egyed példányát. A kapcsolatok az egyedek között nem beágyazással, hanem **hivatkozó attribútumokkal** valósulnak meg.

```
<kolcsonzo kkod="k1">
    <nev>CityCar Rent</nev>
    <elerhetoseg>+36-30-555-1111</elerhetoseg>
    <elerhetoseg>citycar@rent.hu</elerhetoseg>
    <nyitvatartas>
        <nap>Hétfő</nap>
        <intervallum>09:00-18:00</intervallum>
    </nyitvatartas>
    <nyitvatartas>
        <nap>Csütörtök</nap>
        <intervallum>10:00-18:00</intervallum>
    </nyitvatartas>
    <nyitvatartas>
        <nap>Péntek</nap>
        <intervallum>08:00-16:00</intervallum>
    </nyitvatartas>
</kolcsonzo>
```

A **kolcsonzo** elem a kölcsönző adatait tartalmazza, többértékű és összetett elemekkel egyaránt, mint például **elerhetoseg** és **nyitvatartas**. Ez a kódrészlet jól illusztrálja a többértékű tulajdonságokat, hogy a dokumentumban külön XML-elemként szerepelnek.

```
<alkalmazott akod="a1" k_a="k2">
    <nev>Tóth Márk</nev>
    <muszak>Délelőtt</muszak>
    <muszak>Délután</muszak>
    <vegzettseg>Közgazdász</vegzettseg>
    <vegzettseg>Vállalkozásmenedzsment</vegzettseg>
</alkalmazott>
```

Az **alkalmazott** elem **k_a** attribútuma (mint ahogy már korábban is említettem) egyértelműen összekapcsolja a dolgozót a megfelelő kölcsönzővel, miközben az egyedek a gyökérelem alatt függetlenül helyezkednek el.

```
<berles k_u_k="k1" k_u_u="u1">
    <jarmu>
        <marka>Toyota</marka>
        <tipus>Corolla</tipus>
        <rendszam>ABC-123</rendszam>
    </jarmu>
    <kezdete>2025-05-10</kezdete>
    <vege>2025-05-17</vege>
    <dij>
        <kaucio>40000</kaucio>
        <napidij>9000</napidij>
        <kilometerdij>70</kilometerdij>
    </dij>
</berles>
```

Itt látható, hogy az N:M kapcsolatot (**berles**) **külön elemként** kezeljük, ahol a kapcsolatot szintén az attribútumok (**k_u_k, k_u_u**) biztosítják a megfelelő kölcsönzőhöz (**k1**) és ügyfélhez (**u1**) egyaránt.

1.4 Az XML dokumentum alapján XMLSchema készítése

Az XML dokumentum validálásához elkészítettem az XML Schema-t (XSD), amely az egyedekhez **saját**, **komplex típusokat**, **kulcsokat** (key) és **kulcshivatkozásokat** (keyref) definiál. Például a kölcsönző típusa a következőképpen néz ki:

```
<xs:complexType name="kolcsonzoTipus">
    <xs:sequence>
        <xs:element name="nev" type="xs:string"/>
        <xs:element name="cim" type="cimTipus"/>
        <xs:element name="elerhetoseg" type="xs:string"
                max0ccurs="unbounded"/>
        <xs:element name="nyitvatartas"
                type="nyitvatartasTipus" max0ccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="kkod" type="xs:string"
                use="required"/>
</xs:complexType>
```

A **nev** és **cim** elemek egyszerűen a kölcsönző nevét és címét tárolják, az utóbbi szintén egy összetett típussal definiálva (**cimTipus**) ami lehetővé teszi, hogy egy elem több al-elemet tartalmazzon. Az **elerhetoseg** és **nyitvatartas** a **maxOccurs="unbounded"** attribútummal többször is előfordulhatnak, így a Schema engedélyezi, hogy egy kölcsönzőnek több elérhetősége és nyitvatartási időszaka legyen.

A **kkod** attribútum a kölcsönző **egyedi azonosítója**, amely kulcsként szolgál a dokumentum többi elemével való kapcsolatokhoz. A kulcs így néz ki a kölcsönző esetében definiálva:

```
<xs:key name="kolcsonzo_kulcs">
    <xs:selector xpath="kolcsonzo"/>
    <xs:field xpath="@kkod"/>
</xs:key>
```

Ez tesz róla, hogy minden kölcsönző rendelkezzen egyedi **kkod** attribútummal, így **nem lehet két azonos azonosítójú kölcsönző** a dokumentumban.

Az idegen kulcs kapcsolatokat keyref segítségével valósítottam meg, például az alkalmazottak kölcsönzőhöz való kapcsolódását így ellenőriztem:

```
<xs:keyref refer="kolcsonzo_kulcs"
           name="kolcsonzo_alkalmazott_idegen_kulcs">
    <xs:selector xpath="alkalmazott"/>
    <xs:field xpath="@k_a"/>
</xs:keyref>
```

Az **@k_a** attribútum jelzi, hogy az adott alkalmazott melyik kölcsönzőhöz tartozik. A **refer="kolcsonzo_kulcs"** biztosítja, hogy csak létező kölcsönzőhöz lehessen az alkalmazottat hozzárendelni. Így a Schema **ellenőrzi az adatok konzisztenciáját**, és nem enged meg hibás hivatkozásokat.

A tulajdonosok és kölcsönzők 1:1 kapcsolatát az alábbi módon valósítottam meg a **unique** kulcsszó alkalmazásával:

```
<xs:unique name="unique_tulajdonos">
    <xs:selector xpath="tulajdonos"/>
    <xs:field xpath="@k_t"/>
</xs:unique>
```

Ez garantálja, hogy **egy kölcsönzőhöz csak egy tulajdonos** tartozhasson, így az 1:1 kapcsolat az XSD-ben is érvényesül. A key, keyref és unique mind a gyökérelem element-en belül vannak.

Nem csak az egyedekhez, hanem az összetett attribútumokhoz is készítettem saját komplex típusokat. A korábban említett **cimTipus** például így néz ki:

```
<xs:complexType name="cimTipus">
    <xs:sequence>
        <xs:element name="varos" type="xs:string"/>
        <xs:element name="utca" type="xs:string"/>
        <xs:element name="hazszam" type="xs:integer"/>
    </xs:sequence>
</xs:complexType>
```

2. DOM osztályok elkészítése

2.1 Adatolvasás

A feladat célja az XML dokumentum (**E00XC3XML.xml**) feldolgozása DOM parser segítségével, majd a teljes tartalom blokk formában történő kiírása a konzolra, valamint mentése új XML fájlba (E00XC3XML_output.xml). A kódot **E00XC3DOMRead.java** fájlban valósítottam meg.

Első lépésben az XML dokumentumot egy **DocumentBuilder segítségével beolvastam.** A **normalize()** hívás biztosítja, hogy a DOM struktúra egyértelmű és konzisztens legyen (pl. az egymást követő whitespace-ek összevonásra kerülnek).

```
File inputFile = new File("E00XC3XML.xml");

DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(inputFile);

doc.getDocumentElement().normalize();
```

Ez a blokk felelős azért, hogy a nyers XML fájl egy memóriában kezelhető dokumentumfára alakuljon. A parser ilyenkor a teljes XML-t beolvassa, így bármely elem bármikor elérhető a további feldolgozáshoz.

A dokumentum beolvasását követően a következő feladat a releváns csomópontok kinyerése és feldolgozása volt. Ehhez először a kívánt elemcsoportot kértem le a dokumentumból, majd sorban végigiteráltam minden előforduláson. Az alábbi példa jól mutatja, hogyan olvastam be az **alkalmazottak** adatait:

```
NodeList alkalmazottak =
doc.getElementsByTagName("alkalmazott");

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

    Element a = (Element) alkalmazottak.item(i);

    System.out.println("Aktuális elem: " + a.getTagName());
```

```
    System.out.println("Alkalmazott kód: " +
a.getAttribute("akod"));

    System.out.println("Kölcsönző kódja: " +
a.getAttribute("k_a"));

    System.out.println("Alkalmazott neve: " +
a.getElementsByTagName("nev").item(0).getTextContent());

    System.out.println("Beosztás: " +
a.getElementsByTagName("beosztas").item(0).getTextContent());


    System.out.println("Műszakok:");

    NodeList mu = a.getElementsByTagName("muszak");

    for (int j = 0; j < mu.getLength(); j++)

        System.out.println("  " +
mu.item(j).getTextContent().trim());


    System.out.println("Végzettségek:");

    NodeList vg = a.getElementsByTagName("vegzettseg");

    for (int j = 0; j < vg.getLength(); j++)

        System.out.println("  " +
vg.item(j).getTextContent().trim());


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

}
```

Az alkalmazottak listáját a **getElementsByTagName()** függvénnyel kérem le, amely egy **NodeList**-ben adja vissza az összes alkalmazott elemet. A lista bejárása során minden csomópontot **Element típusra castolok**, így azonnali hozzáférést kapok az attribútumaihoz (például **akod, k_a**). A további adatok már gyermekcsomópontokként találhatók meg. A **trim()** használata biztosítja, hogy a kiolvasott értékek ne tartalmazzanak felesleges whitespace-eket.

A konzolra történő kiíratás mellett a program a beolvasott dokumentumot **új XML fájlba** is menti. Ehhez a Java **Transformer** osztályát használom, amely a teljes DOM struktúrát – az összes csomóponttal, attribútummal és hierarchiával – XML formátummá alakít. **A Document objektumból DOMSource készül, majd ezt egy StreamResult célfájlba írom:**

```
TransformerFactory transformerFactory =
TransformerFactory.newInstance();

Transformer transformer = transformerFactory.newTransformer();

DOMSource source = new DOMSource(doc);

StreamResult result = new StreamResult(new
File("EOOXC3XML_output.xml"));

transformer.transform(source, result);
```

A dokumentum kiírása előtt a **removeEmptyTextNodes** segédfüggvénnyel eltávolítottam az üres text node-okat, így a Transformer által generált XML tiszta és olvasható marad. A függvény rekurzívan végigmegy az összes elem gyermekein, és minden olyan text node-ot töröl, amely csak whitespace-t tartalmaz.

```
private static void removeEmptyTextNodes(Element element) {
    NodeList children = element.getChildNodes();
    for (int i = children.getLength() - 1; i >= 0; i--) {
        Node child = children.item(i);
        if (child.getNodeType() == Node.TEXT_NODE) {
            if (child.getTextContent().trim().isEmpty()) {
                element.removeChild(child);
            }
        } else if (child.getNodeType() == Node.ELEMENT_NODE) {
            removeEmptyTextNodes((Element) child);
        }
    }
```

2.2 Adat-lekérdezés

A DOM-alapú feldolgozás második lépése a **különféle lekérdezések végrehajtása** volt az XML dokumentumon. A folyamat hasonlóan kezdődött, mint az adatolvasásnál: a dokumentumot beolvastam, majd normalizáltam, hogy a DOM struktúra konzisztens legyen. Ezután a **getElementsByTagName()** metódussal lekértem a kívánt csomópontokat, és **NodeList**-en iterálva dolgoztam fel az adatokat.

1. lekérdezés: Kölcsönzők nyitva szombaton

```
NodeList kolcsonzok = doc.getElementsByTagName("kolcsonzo");
for (int i = 0; i < kolcsonzok.getLength(); i++) {
    Element kolcsonzoElem = (Element) kolcsonzok.item(i);
    NodeList nyitvatartasok =
kolcsonzoElem.getElementsByTagName("nyitvatartas");
    for (int j = 0; j < nyitvatartasok.getLength(); j++) {
        Element nyitElem = (Element) nyitvatartasok.item(j);
        String nap =
nyitElem.getElementsByTagName("nap").item(0).getTextContent();
     if ("Szombat".equals(nap)) {
          String intervallum =
          nyitElem.getElementsByTagName("intervallum")
          .item(0).getTextContent();
             // A cím és elérhetőségek kiolvasása
        }
     // Konzolra írás
    }
}
```

Ebben a lekérdezésben a DOM segítségével minden kölcsönző nyitvatartási idejét vizsgáltam, és csak azokat jelenítettem meg, akik **szombaton nyitva** vannak. A cím és az elérhetőségek szintén gyermekelemekből kerültek kiolvasásra, így a teljes információ strukturáltan jelenik meg a konzolban.

2. lekérdezés: Legdrágább szolgáltatás kölcsönzőként

```
NodeList szList = doc.getElementsByTagName("szolgaltatas");
for (int i = 0; i < kolcsonzok.getLength(); i++) {
    Element kolcsonzoElem = (Element) kolcsonzok.item(i);
    String kkod = kolcsonzoElem.getAttribute("kkod");
    Element maxSz = null;
    int maxAr = -1;
    for (int j = 0; j < szList.getLength(); j++) {
        Element sz = (Element) szList.item(j);
        if (kkod.equals(sz.getAttribute("k_sz"))) {
            int ar =
                Integer.parseInt(sz.getElementsByTagName("kolts
                eg").item(0).getTextContent());
            if (ar > maxAr) {
                maxAr = ar;
                maxSz = sz;
            }
        }
    }
     if (maxSz != null) {
         // Konzolra írás
     }
}
```

Ebben a lekérdezésben minden kölcsönző összes szolgáltatását végigjártam, és az árakat összehasonlítva azonosítottam a legdrágább szolgáltatást. A kiválasztott elem attribútumait (például szolgáltatás kódját) és a gyermekelemeket (név, leírás, költség) kombinálva teljes, részletes információt jelenítettem meg a konzolon. Így nemcsak az ár, hanem a szolgáltatás pontos adatai is áttekinthetők egy helyen.

3. lekérdezés: Ügyfelek, akiknek van B és A kategóriás jogosítványuk

```
NodeList ugyfelek = doc.getElementsByTagName("ugyfel");
for (int i = 0; i < ugyfelek.getLength(); i++) {
    Element uElem = (Element) ugyfelek.item(i);
    NodeList jogositvanyok =
          uElem.getElementsByTagName("jogositvany");
    boolean hasB = false;
    boolean hasA = false;
    for (int j = 0; j < jogositvanyok.getLength(); j++) {
        Element jog = (Element) jogositvanyok.item(j);
        String kategoria =
          jog.getElementsByTagName("kategoria")
          .item(0).getTextContent();
        if ("B".equals(kategoria)) hasB = true;
        if ("A".equals(kategoria)) hasA = true;
    }
    if (hasB && hasA) {
        // Ügyfél adatainak a lekérése
    }
     // Konzolra írás
}
```

Ez a lekérdezés az ügyfelek jogosítványait vizsgálja. A **getElementsByTagName("ugyfel")** lekéri az összes ügyfelet, majd minden ügyfélhez tartozó jogositvany elemeket iteráljuk. Két logikai változó (**hasB** és **hasA**) jelzi, hogy az ügyfél rendelkezik-e B és A kategóriás jogosítvánnyal. Ha mindkettő megvan, az ügyfél további adatai kiolvashatók a feldolgozáshoz.

4. lekérdezés: Bérlések összesített napidíja

```
NodeList berlesek = doc.getElementsByTagName("berles");

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

    Element bElem = (Element) berlesek.item(i);

    LocalDate kezdete = LocalDate.parse(bElem
.getElementsByTagName("kezdete").item(0).getTextContent());

    LocalDate vege = LocalDate.parse(bElem
.getElementsByTagName("vege").item(0).getTextContent());

    long napokSzama = ChronoUnit.DAYS.between(kezdete, vege) +
1;

    Element dijElem = (Element)
bElem.getElementsByTagName("dij").item(0);

    int napidijPerNap = Integer.parseInt(dijElem
.getElementsByTagName("napidij").item(0).getTextContent());

    int napidijOsszesen = (int) (napidijPerNap * napokSzama);

    String ugyfelKod = bElem.getAttribute("k_u_u");

    String ugyfelNev = "";

    for (int j = 0; j < ugyfelek.getLength(); j++) {

        Element uElem = (Element) ugyfelek.item(j);

        if (ugyfelKod.equals(uElem.getAttribute("ukod"))) {

            ugyfelNev =
uElem.getElementsByTagName("nev").item(0).getTextContent();

            break;

        }} // Konzolra írás
```

A lekérdezés során a **getElementsByTagName("berles")** segítségével összegyűjtjük az összes berles elemet. Minden bérlésnél kiolvassuk a kezdő- és befejező dátumot, kiszámoljuk a napok számát a **ChronoUnit.DAYS.between()**-nel, majd a napi díjat megszorozzuk a napok számával az összesített díjhoz. A **k_u_u** attribútum alapján megtaláljuk a megfelelő **ugyfel** elemet, és kiolvassuk az ügyfél nevét, így minden bérléshez rendelkezésre állnak a szükséges adatok.

2.3 Adatmódosítás

A DOM-alapú adatmódosítás az utolsó része a feladatnak, melynek célja az XML dokumentum tartalmának programozott változtatása. A folyamat szintén a dokumentum beolvasásával és normalizálásával kezdődik, majd az egyes módosítási műveletek következnek.

1. módosítás: Első kölcsönző címének változtatása

A **getElementsByTagName("kolcsonzo")** segítségével lekérem az összes kölcsönzőt, majd az első elemhez (**item(0)**) hozzáférek. A cim gyermekelemekből kiolvasom a **varos**, **utca** és **hazszam** értékeket. A **setTextContent()** metódussal pedig átírom a cím adatait, így az első kölcsönző új címet kap. (Budapest Fő utca 99)

```
NodeList kolcsonzok = doc.getElementsByTagName("kolcsonzo");

Element firstKolcsonzo = (Element) kolcsonzok.item(0);

Element cimElem = (Element)
firstKolcsonzo.getElementsByTagName("cim").item(0);

cimElem.getElementsByTagName("varos").item(0)
.setTextContent("Budapest");

cimElem.getElementsByTagName("utca").item(0)
.setTextContent("Fő utca");

cimElem.getElementsByTagName("hazszam").item(0)
.setTextContent("99");
```

2. módosítás: C kategóriás jogosítványok módosítása D-re

Lekérem az összes **ugyfel** elemet egy **NodeList**-be. Minden ügyfélhez tartozó **jogositvany** elemet végigjárva ellenőrzöm a kategoria értékét. Ha az érték "C", akkor azt átírom "D"-re a **setTextContent()** segítségével.

```
NodeList ugyfelek = doc.getElementsByTagName("ugyfel");

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

    Element ugyfel = (Element) ugyfelek.item(i);

    NodeList jogositvanyok =
    ugyfel.getElementsByTagName("jogositvany");

    for (int j = 0; j < jogositvanyok.getLength(); j++) {
```

```
        Element jogositvany = (Element) jogositvanyok.item(j);

        Element kategoria = (Element)
        jogositvany.getElementsByTagName("kategoria").item(0);

        if (kategoria.getTextContent().equals("C")) {

            System.out.println("Ügyfél: " + ugyfel
.getElementsByTagName("nev").item(0).getTextContent() + " -
Eredeti kategória: " + kategoria.getTextContent());

            kategoria.setTextContent("D");

            System.out.println("Új kategória: D");

        }

}}
```

3. módosítás: Alkalmazottak kölcsönzőjének cseréje

Az összes **alkalmazott** elem attribútumát (**k_a**) vizsgálom. Az érték "**k1**" és "**k2**" között cserélődik. Az **setAttribute()** metódussal módosítom az attribútumot, így az alkalmazottak kölcsönzőhöz való hozzárendelése változik.

```
NodeList alkalmazottak =
doc.getElementsByTagName("alkalmazott");

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

        Element alkalmazott = (Element) alkalmazottak.item(i);

        String eredetiKod = alkalmazott.getAttribute("k_a");

        String ujKod = eredetiKod.equals("k1") ? "k2" :
(eredetiKod.equals("k2") ? "k1" : eredetiKod);

        System.out.println("Alkalmazott: " + alkalmazott
.getElementsByTagName("nev").item(0).getTextContent() + " -
Eredeti kölcsönző: " + eredetiKod + ", Új kölcsönző: " +
ujKod);

        alkalmazott.setAttribute("k_a", ujKod);

}
```

4. módosítás: Új szolgáltatás hozzáadása kommenttel

Létrehoztam egy kommentet **doc.createComment()** segítségével, majd egy új **szolgaltatas** elemet a megfelelő attribútumokkal (**szk, k_sz**) a **setAttribute()** használatával, valamint gyermekelemekkel (**nev, leiras, koltseg**) az **appendChild()**-al. A **parent.insertBefore()** segítségével a komment és az új szolgáltatás a megfelelő pozícióba kerül a DOM-ban.

```
Comment comment = doc.createComment(" 5. Szolgáltatás ");
Element ujSzolgaltatas = doc.createElement("szolgaltatas");
ujSzolgaltatas.setAttribute("szk", "sz5");
ujSzolgaltatas.setAttribute("k_sz", "k1");
Element nev = doc.createElement("nev");
nev.setTextContent("Biciklitartó bérlés");
ujSzolgaltatas.appendChild(nev);
Element leiras = doc.createElement("leiras");
leiras.setTextContent("Kiegészítő biciklitartó biztosítása a járműhöz hosszabb utazásokhoz.");
ujSzolgaltatas.appendChild(leiras);
Element koltseg = doc.createElement("koltseg");
koltseg.setTextContent("3000");
ujSzolgaltatas.appendChild(koltseg);
Node lastSz = doc.getElementsByTagName("szolgaltatas")
.item(doc.getElementsByTagName("szolgaltatas").getLength()-1);
Node parent = lastSz.getParentNode();
parent.insertBefore(comment, lastSz.getNextSibling());
parent.insertBefore(ujSzolgaltatas, comment.getNextSibling());
```

Végezetül pedig a feladat alapján a Transformer segítségével a módosított DOM-fát XML formátumba írjuk ki a konzolra. (itt a **StreamResult** nem egy File hanem **System.out** lesz). A korábban már használt **removeEmptyTextNodes()** függvény pedig eltávolítja a felesleges whitespace szöveges csomópontokat, hogy az XML tiszta és formázott legyen.