JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

Állatorvosi Rendelő

Készítette: **Tóth Mátyás**

Neptunkód: **OQH5NH**

Dátum: **2025. november**

Miskolc, 2025

Tartalomjegyzék

Bevezetés

A feladat leírása:

A beadandó témája volt egy olyan nyilvántartó rendszer, ami egy állatorvosi rendelő napi működésének támogatására szolgál. A rendszer célja, hogy strukturáltan és hatékonyan kezelje az állatorvosi rendelő minden fontos információját, amely a betegek ellátásához, a gazdák kezeléséhez és a rendelő üzleti folyamatainak nyomon követéséhez szükséges.

A rendszer nyilvántartja a rendelőhöz tartozó állatok alapvető adatait, mint például a nevüket, fajukat, születési dátumukat és az elvégzett oltásaikat. Emellett rögzíti a gazdák elérhetőségeit és lakcímüket, hogy a rendelő bármikor kapcsolatba léphessen velük. A rendszer tartalmazza az orvosok szakmai információit is, beleértve a szakterületüket és beosztásukat, ami segít a megfelelő szakember kiválasztásában az egyes esetekhez.

A rendszer egyik legfontosabb funkciója a kezelések nyilvántartása, amely tartalmazza a kezelés dátumát, a diagnózist és a kezelés árát. Minden kezeléshez hozzárendelhető több gyógyszer is, a megfelelő adagolási és időtartam információkkal együtt. Emellett minden állathoz tartozik egy egészségügyi kiskönyv, amely tartalmazza az állat egészségügyi történetének fontos megjegyzéseit és a kiskönyv létrehozásának dátumát. Ez a struktúra lehetővé teszi, hogy az állatorvosi rendelő hatékonyan nyomon kövesse a betegek teljes egészségügyi történetét, és segít a minőségi állategészségügyi ellátás biztosításában.

Entitások, tulajdonságaik, kapcsolataik:

- **Gazda**
    - gazda_id: a gazda egyedi azonosítója,
    - név: a gazda neve,
    - cím: összetett tulajdonság, városból, utcából és házszámból áll, itt lakik a gazda,
    - telefonszám: a gazda telefonos elérhetősége,
    - email: a gazda e-mail címe.
- **Állat**
    - allat_id: az állat egyedi azonosítója,
    - név: az állat neve,

    - faj: az állat faja,
    - születési_dátum: az állat születési dátuma,
    - oltás: ebből több is lehet egy állatnak, az állat élete során eddig megkapott oltások.
- **Orvos**
    - orvos_id: az orvos egyedi azonosítója,
    - név: az orvos neve,
    - szakterület: az orvos szakterülete, amiben jártas.
    - beosztás: az orvos beosztása.
- **Gyógyszer**
    - gyogyszer_id: a gyógyszer egyedi azonosítója,
    - név: a gyógyszer neve,
    - hatóanyag: a gyógyszerben található hatóanyag neve,
    - adagolás: a gyógyszerhez előírt adagolás.
- **Kezelés**
    - kezelés_id: a kezelés egyedi azonosítója,
    - dátum: a kezelés időpontja,
    - diagnózis: a kezelés során felállított diagnózis,
    - ár: a kezelés költsége.
- **Egészségügyi Könyv**
    - konyv_id: az egészségügyi könyv egyedi azonosítója,
    - létrehozás_dátum: az egészségügyi könyv kiállításának időpontja,
    - megjegyzés: az egészségügyi könyvhöz írt megjegyzés az orvosok által.
- **Gazda – Állat -> (1:N)**
- **Állat – Kezelés -> (1:N)**
- **Orvos – Kezelés -> (1:N)**
- **Állat – Egészségügyi Könyv -> (1:1)**
- **Kezelés – Gyógyszer -> (N:M)**
    - adag: az adott kezelésen adott gyógyszerből előírt adag az állat számára,
    - időtartam: az adott kezelésen adott gyógyszerhez előírt beszedési időtartam.

1. Feladat

1.1 Az adatbázis ER modell tervezése

Az adatbázis tervezése során először azonosítottam a rendszerben előforduló egyedeket és azok legfontosabb tulajdonságait. Mivel egy állatorvosi rendelő nyilvántartó rendszeréről van szó, logikus volt külön entitásként kezelni az Állatot, a Gazdát, az Orvost, a Kezelést, a Gyógyszert, valamint az Egészségügyi Könyvet. Miután megvoltak az egyedek és a hozzájuk tartozó tulajdonságok, a következő lépés az volt, hogy meghatározzam a közöttük lévő kapcsolatokat, és ezek alapján kialakítsam az ER modellt.

A Gazda és az Állat között egy 1:N kapcsolat jött létre, ami azt jelenti, hogy egy gazdának több állata is lehet, viszont egy állatnak egyszerre csak egy gazdája van. Ezt azért is fontos kiemelni, mert a rendszerben minden állathoz kötelező tartoznia egy gazdának, aki felelős érte és akinek a kapcsolati adatai az adminisztrációhoz szükségesek.

Az Állat és a Kezelés között szintén 1:N kapcsolat áll fenn, mivel egy állat több kezelésen is részt vehet az élete során (pl. oltások, vizsgálatok, beavatkozások), de egy kezelés mindig egy konkrét állathoz tartozik. A Kezelés entitásban tároltam a vizsgálat dátumát, diagnózist és árát, így könnyen visszakövethető, hogy az állat milyen kezeléseken esett át.

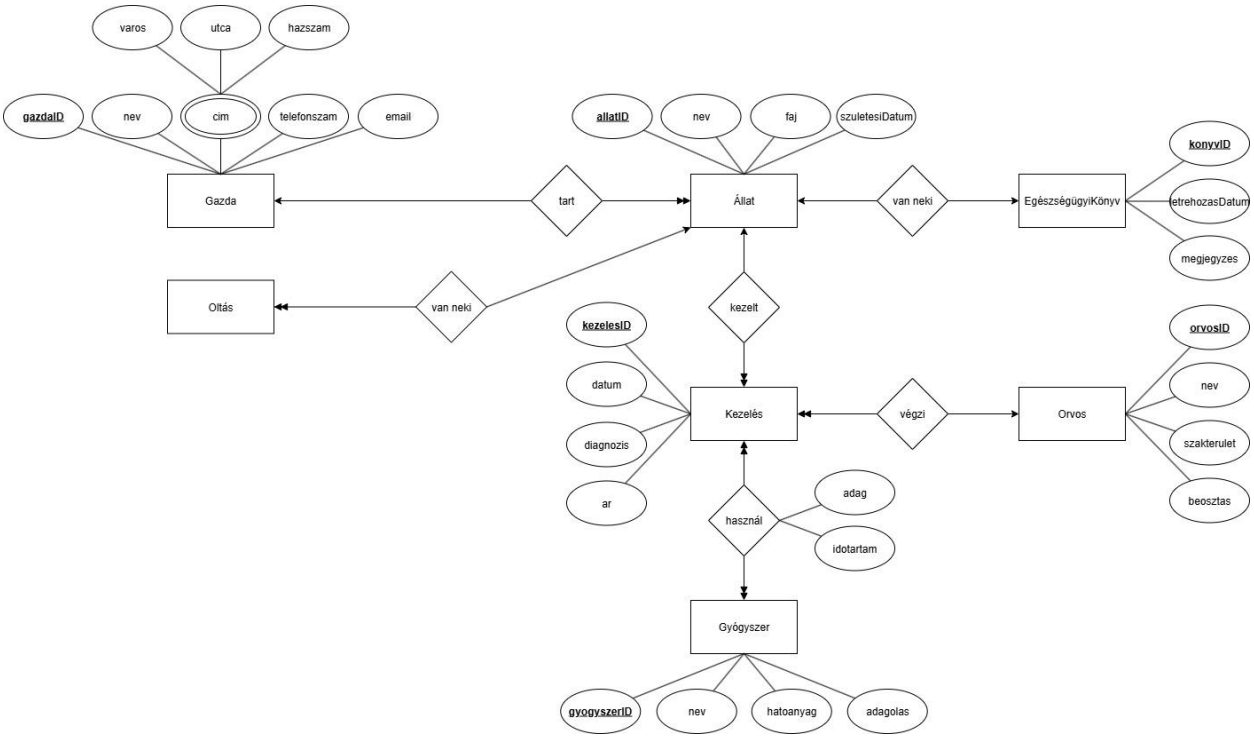
Az Orvos és a Kezelés kapcsolatát szintén 1:N kapcsolatként definiáltam. Itt az „egy” oldal az orvos, mivel egy orvos több kezelést is végezhet, viszont minden kezeléshez egyértelműen hozzárendelhető, hogy azt melyik orvos végezte.

A Kezelés és a Gyógyszer között már N:M kapcsolat alakult ki. Egy kezelés során többféle gyógyszert is alkalmazhatnak, és ugyanaz a gyógyszer más kezeléseknél is előfordulhat. A kapcsolatnak vannak tulajdonságai is (adag, időtartam), ezért ezt a kapcsolatot egy kapcsoló entitás hozza létre az adatbázisban. Itt volt különösen fontos, hogy a kapcsolat ne csak összekösse az entitásokat, hanem adatot is tároljon a gyógyszer felhasználásáról.

Végül az Állat és az EgészségügyiKönyv között egy 1:1 kapcsolatot hoztam létre. Minden állat rendelkezik egy saját egészségügyi könyvvel, és egy egészségügyi könyv csak egy adott állathoz tartozik. Ez tükrözi a valós helyzetet, hiszen az oltások, kezelések dokumentációja az állat saját könyvében található meg.

Az ER modell megrajzolása során az entitásokat téglalapokkal jelöltem, a hozzájuk tartozó tulajdonságokat ellipszisek szemléltették. A kapcsolatok rombuszokkal kerültek ábrázolásra, a kapcsolat típusát (1:N, N:M, 1:1) pedig nyilakkal vagy számos jelöléssel tüntettem fel az entitások között. A többértékű attribútumot (például az „oltás” az Állat entitásban) külön entitásként jelöltem 1:N kapcsolattal, viszont tulajdonságok nélkül, hogy látszódjon, hogy itt több adatot is tárolhat a rendszer. Ettől függetlenül az „Oltás” nem különálló entitás a nyilvántartó rendszerben, viszont az ER modellek ábrázolásánál ez a leghatékonyabb mód a szemléltetésükre.

Összességében a cél az volt, hogy egy olyan jól átgondolt és logikus ER modellt hozzak létre, amely átláthatóan mutatja az entitások közötti összefüggéseket, és később könnyen XDM-modellé alakítható, ezek mentén készült el a következő modell:



1. *ábra: Az ER modell*

1.2 Az adatbázis konvertálása XDM modellre

Az XDM modell létrehozásánál a korábbi ER modellt és az entitások közötti kapcsolatokat vettem figyelembe. Amíg egy ER modell esetén az entitások közötti kapcsolatok kerülnek előtérbe, addig az XDM modell már az XML nyilvántartó-rendszer hierarchiáját hivatott ábrázolni és fontos szerepet kap benne a szintek definiálása, melyek jól mutatják az XML dokumentum fa szerkezetét.

A fa szerkezet miatt itt az entitások tulajdonságait is entitásként ábrázoljuk, így tulajdonságként, vagy attribútumként kizárólag az elsődleges, vagy idegen kulcsok maradnak.

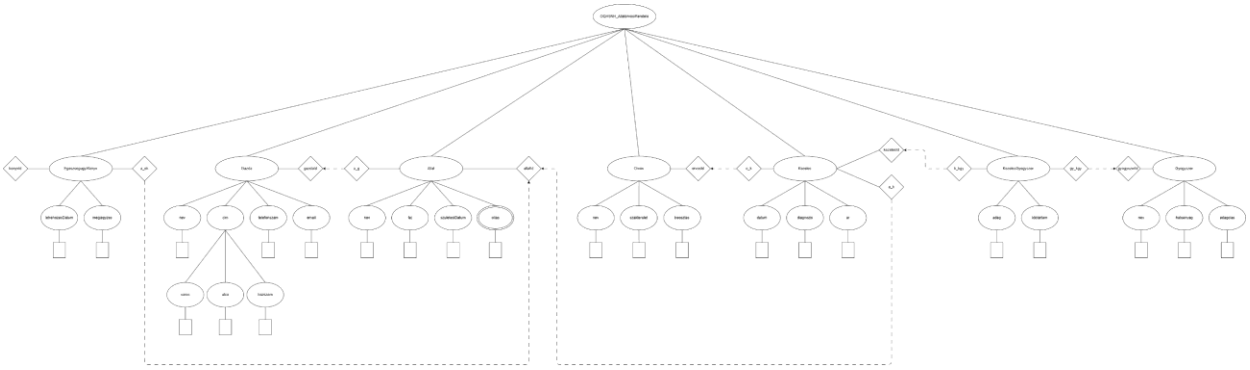
Egy fa szerkezetnek minden esetben van egy egyértelmű abszolút gyökéreleme, így jelen esetben egy fő entitást vettem fel, melyből kiágaznak az 1. szintű entitások.

XDM modell esetén az entitásokat ellipszisként ábrázolom, az attribútumaikat (pl.: elsődleges kulcs, idegen kulcs) rombuszként, a többértékű entitásokat pedig dupla kerettel jelölöm, ezzel jelezve, hogy ebből több egyed is szerepelhet a szülőjén belül. Azon entitások, melyeknek már nincs alárendelve semmilyen más entitás egy téglalappal lesznek összekötve, mely azt szimbolizálja, hogy azon entitás alá egy XML fileban már csak szöveges érték kerül, más gyerekelem nem.

Mindezeket összekötöm vonalakkal, mely most nem a kapcsolatokat, hanem a hierarchiát ábrázolja, így minden szülő – gyerek viszonynál jön létre vonal.

Ezen modell esetén már szükség volt egy kapcsolóentitásra a több-több (N:M) kapcsolat ábrázolására a kezelés és gyógyszer között, így létrejött egy KezelésGyógyszer entitás.

Az idegen kulcsok jelölésére szaggatott vonalat használunk. Ez a vonal az adott entitás idegen kulcsából mutat a hivatkozott entitás elsődleges kulcsára, így jelölve a kapcsolatukat.



2. ábra: Az XDM modell

1.3 Az XDM modell alapján XML dokumentum készítése

Az XDM modell alapján könnyen ki lehet alakítani az XML dokumentumot, hiszen pontosan ugyanez a hierarchia a lényege az XML formátumnak is. Az abszolút gyökérelem az <OQH5NH_AllatorvosiRendelo> nevet kapta, ebbe pedig beágyaztam az 1. szintű

gyerekelemeket, mindegyikből többet. Az elemek attribútumaihoz pedig az elsődleges és idegen kulcsokat is hozzáadtam.

XML dokumentum (részlet):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OQH5NH_AllatorvosiRendelo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                           xsi:noNamespaceSchemaLocation="OQH5NH_XMLSchema.xsd">

    <!-- Gazdák adatai -->
    <!-- Első gazda - Budapest -->
    <gazda gazdaID="G001">
        <nev>Kovács János</nev>
        <cim>
            <varos>Budapest</varos>
            <utca>Fő utca</utca>
            <hazszam>12</hazszam>
        </cim>
        <telefonszam>+36301234567</telefonszam>
        <email>kovacs.janos@email.hu</email>
    </gazda>


    <!-- Állatok adatai -->
    <!-- Első állat - kutya -->
    <allat allatID="A001" g_a="G001">
        <nev>Bodri</nev>
        <faj>Labrador retriever</faj>
        <szuletesiDatum>2020-05-15</szuletesiDatum>
        <oltas>Veszettség</oltas>
        <oltas>Parvo</oltas>
        <oltas>Hepatitis</oltas>
    </allat>

    <!-- Orvosok adatai -->
    <!-- Első orvos - sebész -->
    <orvos orvosID="O001">
        <nev>Dr. Szabó Anna</nev>
        <szakterulet>Sebészet</szakterulet>
        <beosztas>Főorvos</beosztas>
    </orvos>

    <!-- Gyógyszerek adatai -->
    <!-- Első gyógyszer - antibiotikum -->
    <gyogyszer gyogyszerID="GY001">
```

```
        <nev>Amoxicillin</nev>
        <hatoanyag>Amoxicillinum</hatoanyag>
        <adagolas>10 mg/kg naponta kétszer</adagolas>
    </gyogyszer>

    <!-- Kezelések adatai -->
    <!-- Első kezelés - Bodri sebészeti beavatkozása -->
    <kezeles kezelesID="K001" a_k="A001" o_k="O001">
        <datum>2023-11-15</datum>
        <diagnozis>Lábsérülés, sebészeti beavatkozás szükséges</diagnozis>
        <ar>45000</ar>
    </kezeles>

    <!-- Kezelés-Gyógyszer kapcsolatok (N:M kapcsolat) -->
    <!-- Első kezelés gyógyszerei -->
    <kezelesGyogyszer k_kg="K001" gy_kg="GY001">
        <adag>500 mg</adag>
        <idotartam>7 nap</idotartam>
    </kezelesGyogyszer>

    <kezelesGyogyszer k_kg="K001" gy_kg="GY002">
        <adag>5 ml</adag>
        <idotartam>5 nap</idotartam>
    </kezelesGyogyszer>

    <!-- Egészségügyi kiskönyvek (1:1 kapcsolat az állattal) -->
    <!-- Bodri egészségügyi kiskönyve -->
    <egeszsegugyiKonyv konyvID="EK001" a_ek="A001">
        <letrehozasDatum>2020-05-20</letrehozasDatum>
        <megjegyzes>Oltási könyv komplett, minden oltás időben</megjegyzes>
    </egeszsegugyiKonyv>

</OQH5NH_AllatorvosiRendelo>
```

A forráskód részletben szerepel minden entitásból egy-egy példány (állatok, gazdák, orvosok, kezelések, gyógyszerek, egészségügyi könyvek és kezelés-gyógyszerek), melyek itt XML elemekként jelennek meg, a kapcsolatokat pedig az idegen kulcsok jelzik. (Pl.: Gazda -> Állat 1:N kapcsolat kifejezésére bekerült egy “g_a” attribútum minden Állathoz, mely egy gazda elsődleges kulcsára mutat) Az XML teljesen tükrözi az XDM modellt és szemlélteti az egyedek közötti kapcsolatokat.

1.4 Az XML dokumentum alapján XMLSchema készítése

A következő feladatom az volt, hogy a rendelkezésre álló XML dokumentum alapján elkészítsem az XML séma (XMLSchema, XSD) definíciót, amely mind a szerkezetet, mind az adatrestrikciókat leírja és lehetővé teszi az XML fájl automatikus ellenőrzését (validálását).

Először végigolvastam az XML tartalmát, hogy lássam az entitásokat (gazda, állat, orvos, gyógyszer, kezelés, kezelésGyógyszer, egészségügyiKönyv) és az attribútumaikat, valamint a kapcsolati megoldásokat. Fontos volt észrevenni a következő mintákat is: dátumok ISO-formátumban, telefonszámok +36-tal, azonosítók nagybetűs prefix-szel és számokkal, valamint hogy egyes entitásokhoz kapcsolódó metaadatok (pl. Adag, időtartam) a kapcsolóelemben találhatók.

A séma kialakításánál a következő elveket követtem:

- külön simpleType-okat hoztam létre gyakran visszatérő egyszerű típusokra (azonosító, név, város, telefonszám, email, dátum, ár), hogy később újrahasznosíthatóak legyenek és a szabályok központilag legyenek definiálva;
- az összetett struktúrákat complexType-ként modelleztem (pl. sajatCimTipus, sajatGazdaTipus, sajatAllatTipus stb...);
- a többértékű mezőket (például az oltások) maxOccurs=”unbounded”-dal engedélyeztem;
- az N:M kapcsolatot külön elemként hagytam (kezelesGyogyszer), amely tartalmazza mindkét fél azonosítóját attribútumban és a kapcsolathoz kapcsolódó tulajdonságokat (adag, időtartam) elemekben. (Ez megőrzi a relációs adatmodell információgazdagságát XML-ben is);
- az elsődleges kulcsokat és hivatkozásokat XSD xs:key és xs:keyref mechanizmusával valósítottam meg, hogy a dokumentumon belüli referenciák ellenőrizhetők legyenek (pl.: a_k mindig létező allatID-re mutasson);
- az Állat-EgészségügyiKönyv 1:1 kapcsolatát további xs:unique megszorítással biztosítottam, így garantálható, hogy egy adott állathoz legfeljebb egy könyv rendelhető.

A komplex típusok a bemenet szerkezetét tükrözik:

- sajatCimTipus: varos, utca, hazszam sorrenddel (egyszerű, szekvencia alapú struktúra)
- sajatGazdaTipus: név, cím, telefonszám, email + gazdaID attribútum kötelezően

- sajatAllatTipus: oltások minOccurs=”1” maxOccurs=”unbounded”-dal (legalább egy oltás kötelező volt a mintában)
- sajatKezelesGyogyszerTipus: kapcsolóelem, amely két attribútummal (k_kg, gy_kg) hivatkozik a kezelésre és a gyógyszerre, valamint a kapcsolat tulajdonságait (adag, idotartam) elemekként tárolja

A root elem tartalmazza az összes entitás típusú elemet minOccurs=0, maxOccurs=unbounded beállítással, így a fájl sorban tartalmazhat többször különböző entitásokat, ahogy azt a példa is mutatja.

Végeredményként egy működő, validáló XSD születtetett, amely a mintadokumentumot lefedi és lehetővé teszi további XML-dokumentumok szerkezeti és tartalmi ellenőrzését a rendszer számára. Ha később bővítjük az adattartalmat vagy többféle adatforrást integrálunk, könnyen rá lehet húzni további szabályokat vagy finomítani a meglévőket.

Az XML Schema felépítése:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified" attributeFormDefault="unqualified">

    <!-- Saját egyszerű típusok -->

    <!-- ID típusok -->
    <xs:simpleType name="sajatIDTipus">
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z]+[0-9]+"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Név típus -->
    <xs:simpleType name="sajatNevTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="100"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Város típus -->
    <xs:simpleType name="sajatVarosTipus">
        <xs:restriction base="xs:string">
```

```
            <xs:minLength value="1"/>
            <xs:maxLength value="50"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Utca típus -->
    <xs:simpleType name="sajatUtcaTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="100"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Házszám típus -->
    <xs:simpleType name="sajatHazszamTipus">
        <xs:restriction base="xs:positiveInteger">
            <xs:maxInclusive value="9999"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Telefonszám típus -->
    <xs:simpleType name="sajatTelefonszamTipus">
        <xs:restriction base="xs:string">
            <xs:pattern value="\+36[0-9]{9}"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Email típus -->
    <xs:simpleType name="sajatEmailTipus">
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Faj típus -->
    <xs:simpleType name="sajatFajTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="100"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Dátum típus -->
    <xs:simpleType name="sajatDatumTipus">
        <xs:restriction base="xs:date"/>
```

```
    </xs:simpleType>

    <!-- Oltás típus -->
    <xs:simpleType name="sajatOltasTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="100"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Szakterület típus -->
    <xs:simpleType name="sajatSzakteruletTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="100"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Beosztás típus -->
    <xs:simpleType name="sajatBeosztas_Tipus">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Főorvos"/>
            <xs:enumeration value="Szakorvos"/>
            <xs:enumeration value="Gyakornok"/>
            <xs:enumeration value="Asszisztens"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Diagnózis típus -->
    <xs:simpleType name="sajatDiagnozisTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="500"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Ár típus (Forint) -->
    <xs:simpleType name="sajatArTipus">
        <xs:restriction base="xs:positiveInteger">
            <xs:maxInclusive value="9999999"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Hatóanyag típus -->
    <xs:simpleType name="sajatHatoanyagTipus">
```

```
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="200"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Adagolás típus -->
    <xs:simpleType name="sajatAdagolasTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="200"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Adag típus -->
    <xs:simpleType name="sajatAdagTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="100"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Időtartam típus -->
    <xs:simpleType name="sajatIdotartamTipus">
        <xs:restriction base="xs:string">
            <xs:pattern value="[0-9]+\s+nap"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Megjegyzés típus -->
    <xs:simpleType name="sajatMegjegyzesTipus">
        <xs:restriction base="xs:string">
            <xs:minLength value="0"/>
            <xs:maxLength value="500"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Összetett típusok -->

    <!-- Cím összetett típus -->
    <xs:complexType name="sajatCimTipus">
        <xs:sequence>
            <xs:element name="varos" type="sajatVarosTipus"/>
            <xs:element name="utca" type="sajatUtcaTipus"/>
            <xs:element name="hazszam" type="sajatHazszamTipus"/>
```

```
        </xs:sequence>
    </xs:complexType>

    <!-- Gazda típus -->
    <xs:complexType name="sajatGazdaTipus">
        <xs:sequence>
            <xs:element name="nev" type="sajatNevTipus"/>
            <xs:element name="cim" type="sajatCimTipus"/>
            <xs:element name="telefonszam" type="sajatTelefonszamTipus"/>
            <xs:element name="email" type="sajatEmailTipus"/>
        </xs:sequence>
        <xs:attribute name="gazdaID" type="sajatIDTipus" use="required"/>
    </xs:complexType>

    <!-- Állat típus -->
    <xs:complexType name="sajatAllatTipus">
        <xs:sequence>
            <xs:element name="nev" type="sajatNevTipus"/>
            <xs:element name="faj" type="sajatFajTipus"/>
            <xs:element name="szuletesiDatum" type="sajatDatumTipus"/>
            <xs:element name="oltas" type="sajatOltasTipus" minOccurs="1"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="allatID" type="sajatIDTipus" use="required"/>
        <xs:attribute name="g_a" type="sajatIDTipus" use="required"/>
    </xs:complexType>

    <!-- Orvos típus -->
    <xs:complexType name="sajatOrvosTipus">
        <xs:sequence>
            <xs:element name="nev" type="sajatNevTipus"/>
            <xs:element name="szakterulet" type="sajatSzakteruletTipus"/>
            <xs:element name="beosztas" type="sajatBeosztas_Tipus"/>
        </xs:sequence>
        <xs:attribute name="orvosID" type="sajatIDTipus" use="required"/>
    </xs:complexType>

    <!-- Gyógyszer típus -->
    <xs:complexType name="sajatGyogyszerTipus">
        <xs:sequence>
            <xs:element name="nev" type="sajatNevTipus"/>
            <xs:element name="hatoanyag" type="sajatHatoanyagTipus"/>
            <xs:element name="adagolas" type="sajatAdagolasTipus"/>
        </xs:sequence>
        <xs:attribute name="gyogyszerID" type="sajatIDTipus" use="required"/>
```

```
    </xs:complexType>

    <!-- Kezelés típus -->
    <xs:complexType name="sajatKezelesTipus">
        <xs:sequence>
            <xs:element name="datum" type="sajatDatumTipus"/>
            <xs:element name="diagnozis" type="sajatDiagnozisTipus"/>
            <xs:element name="ar" type="sajatArTipus"/>
        </xs:sequence>
        <xs:attribute name="kezelesID" type="sajatIDTipus" use="required"/>
        <xs:attribute name="a_k" type="sajatIDTipus" use="required"/>
        <xs:attribute name="o_k" type="sajatIDTipus" use="required"/>
    </xs:complexType>

    <!-- Kezelés-Gyógyszer kapcsolat típus (N:M) -->
    <xs:complexType name="sajatKezelesGyogyszerTipus">
        <xs:sequence>
            <xs:element name="adag" type="sajatAdagTipus"/>
            <xs:element name="idotartam" type="sajatIdotartamTipus"/>
        </xs:sequence>
        <xs:attribute name="k_kg" type="sajatIDTipus" use="required"/>
        <xs:attribute name="gy_kg" type="sajatIDTipus" use="required"/>
    </xs:complexType>

    <!-- Egészségügyi kiskönyv típus -->
    <xs:complexType name="sajatEgeszsegugyiKonyvTipus">
        <xs:sequence>
            <xs:element name="letrehozasDatum" type="sajatDatumTipus"/>
            <xs:element name="megjegyzes" type="sajatMegjegyzesTipus"/>
        </xs:sequence>
        <xs:attribute name="konyvID" type="sajatIDTipus" use="required"/>
        <xs:attribute name="a_ek" type="sajatIDTipus" use="required"/>
    </xs:complexType>

    <!-- Gyökérelem és kulcsok -->
    <xs:element name="OQH5NH_AllatorvosiRendelo">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="gazda" type="sajatGazdaTipus" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="allat" type="sajatAllatTipus" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="orvos" type="sajatOrvosTipus" minOccurs="0"
maxOccurs="unbounded"/>
```

```
                <xs:element name="gyogyszer" type="sajatGyogyszerTipus"
minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="kezeles" type="sajatKezelesTipus" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="kezelesGyogyszer"
type="sajatKezelesGyogyszerTipus" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="egeszsegugyiKonyv"
type="sajatEgeszsegugyiKonyvTipus" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>

        <!-- Kulcsok definiálása -->
        <xs:key name="gazdaIDKey">
            <xs:selector xpath="gazda"/>
            <xs:field xpath="@gazdaID"/>
        </xs:key>

        <xs:key name="allatIDKey">
            <xs:selector xpath="allat"/>
            <xs:field xpath="@allatID"/>
        </xs:key>

        <xs:key name="orvosIDKey">
            <xs:selector xpath="orvos"/>
            <xs:field xpath="@orvosID"/>
        </xs:key>

        <xs:key name="gyogyszerIDKey">
            <xs:selector xpath="gyogyszer"/>
            <xs:field xpath="@gyogyszerID"/>
        </xs:key>

        <xs:key name="kezelesIDKey">
            <xs:selector xpath="kezeles"/>
            <xs:field xpath="@kezelesID"/>
        </xs:key>

        <xs:key name="konyvIDKey">
            <xs:selector xpath="egeszsegugyiKonyv"/>
            <xs:field xpath="@konyvID"/>
        </xs:key>

        <!-- Keyref-ek - kapcsolatok definiálása -->

        <!-- Állat - Gazda kapcsolat (Gazda - Állat 1:N) -->
```

```
        <xs:keyref name="allatGazdaRef" refer="gazdaIDKey">
            <xs:selector xpath="allat"/>
            <xs:field xpath="@g_a"/>
        </xs:keyref>

        <!-- Kezelés - Állat kapcsolat (Állat - Kezelés 1:N) -->
        <xs:keyref name="kezelesAllatRef" refer="allatIDKey">
            <xs:selector xpath="kezeles"/>
            <xs:field xpath="@a_k"/>
        </xs:keyref>

        <!-- Kezelés - Orvos kapcsolat (Orvos - Kezelés 1:N) -->
        <xs:keyref name="kezelesOrvosRef" refer="orvosIDKey">
            <xs:selector xpath="kezeles"/>
            <xs:field xpath="@o_k"/>
        </xs:keyref>

        <!-- Kezelés-Gyógyszer - Kezelés kapcsolat (N:M kapcsolat első oldala) --
>
        <xs:keyref name="kezelesGyogyszerKezelesRef" refer="kezelesIDKey">
            <xs:selector xpath="kezelesGyogyszer"/>
            <xs:field xpath="@k_kg"/>
        </xs:keyref>

        <!-- Kezelés-Gyógyszer - Gyógyszer kapcsolat (N:M kapcsolat második
oldala) -->
        <xs:keyref name="kezelesGyogyszerGyogyszerRef" refer="gyogyszerIDKey">
            <xs:selector xpath="kezelesGyogyszer"/>
            <xs:field xpath="@gy_kg"/>
        </xs:keyref>

        <!-- Egészségügyi kiskönyv - Állat kapcsolat (1:1 kapcsolat) -->
        <xs:keyref name="egeszsegugyiKonyvAllatRef" refer="allatIDKey">
            <xs:selector xpath="egeszsegugyiKonyv"/>
            <xs:field xpath="@a_ek"/>
        </xs:keyref>

        <!-- További megszorítás: Egészségügyi kiskönyv egyedi állatra (1:1
kapcsolat biztosítása) -->
        <xs:unique name="uniqueAllatForKonyv">
            <xs:selector xpath="egeszsegugyiKonyv"/>
            <xs:field xpath="@a_ek"/>
        </xs:unique>
    </xs:element>
```

```
</xs:schema>
```

2. Feladat

A második feladat során az volt a célom, hogy az XML állományban tárolt adatokat DOM alapú feldolgozással kezeljem. Ehhez létrehoztam egy külön projektet, amelyben több Java osztály segítségével valósítottam meg az adatolvasást, lekérdezést és adatmódosítást. A feladat része volt, hogy az XML dokumentumot beolvassam, majd a tartalmából különféle lekérdezéseket készítsek, illetve módosításokat hajtsak végre, és ezeket kiírjam a konzolra. A megoldás során nagy hangsúlyt kapott a DOM struktúra kezelése, valamint a kód megfelelő kommentelése és dokumentálása.

2.1 Adatolvasás

Ebben a részfeladatban az volt a cél, hogy a DOM (Document Object Model) felhasználásával beolvassam az XML fájlt, majd a dokumentumban szereplő elemeket feldolgozzam és strukturált formában kiírjam a konzolra. A DOM feldolgozás lényege, hogy az egész XML állományt egy fa struktúrában a memóriába tölti be, így minden elemhez, attribútumhoz és gyerek elemhez egyszerűen hozzá lehet férni. A programom ehhez a javax.xml.parsers és az org.w3c.dom csomag osztályait használja.

Első lépésben megadom a betöltendő XML fájl nevét, majd létrehozok egy DocumentBuilderFactory példányt, amellyel beállítom, hogy a DOM parser figyelmen kívül hagyja a kommenteket és a felesleges whitespace-eket:

```
File xmlFile = new File("src/OQH5NH_XML.xml");

// DOM dokumentum létrehozása fileból
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setIgnoringComments(true);
factory.setIgnoringElementContentWhitespace(true);

DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(xmlFile);

// Szöveges gyerekelemek összevonása, normalizálás
doc.getDocumentElement().normalize();
```

a normalize() hívás arra szolgál, hogy az egymás mellett lévő szöveges node-okat (például sortörés miatt) a parser összevonja, így könnyebb dolgozni velük.

Miután az XML dokumentum betöltődött, lekérem a gyökérelem gyerekelemeit, majd végigiterálok rajtuk:

```
            // Node-ok elmentése egy változóba
            NodeList nodes = doc.getDocumentElement().getChildNodes();

            // Iterálás a node-okon
            for (int i = 0; i < nodes.getLength(); i++) {
                Node n = nodes.item(i);

                if (n.getNodeType() == Node.ELEMENT_NODE) {
                    Element elem = (Element) n;
                    String tag = elem.getTagName();
```

A kódban egy if-else szerkezet segítségével vizsgálom, hogy éppen melyik entitást dolgozza fel a program (gazda, állat, orvos, stb.). Minden elemnél kiírásra kerülnek az attribútumai és a hozzá tartozó tulajdonságok.

Például a gazdák adatai így kerülnek feldolgozásra:

```
// Gazdák kiírása
if (tag.equals("gazda")) {
   System.out.println("Gazda (ID: " + elem.getAttribute("gazdaID") + ")");
   System.out.println("Név: " +
elem.getElementsByTagName("nev").item(0).getTextContent());

   Element cim = (Element) elem.getElementsByTagName("cim").item(0);
   System.out.println("Cím: " +
cim.getElementsByTagName("varos").item(0).getTextContent() + ", " +
cim.getElementsByTagName("utca").item(0).getTextContent() + ", " +
cim.getElementsByTagName("hazszam").item(0).getTextContent()
                        );

   System.out.println("Telefonszám: " +
elem.getElementsByTagName("telefonszam").item(0).getTextContent());
   System.out.println("Email: " +
elem.getElementsByTagName("email").item(0).getTextContent());
   System.out.println("-------------------------------------------------");
```

```
}
```

A program végén a betöltött DOM struktúrát újra kiírom egy másik XML fájlba (oqh5nh_xml_domread.xml). Ehhez Transformer objektumot használok, és beállítom a kimeneti dokumentum szépen formázott identálását is:

```
// A node lista és tulajdonságaik, gyerekeik kiírása egy új XML fileba
Transformer transformer = TransformerFactory.newInstance().newTransformer();

transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");
transformer.transform(
        new DOMSource(doc),
        new StreamResult(new File("src/oqh5nh_xml_domread.xml"))
);
```

A részfeladat során egy teljes XML dokumentumot beolvastam a DOM használatával, majd a struktúrában szereplő entitásokat feldolgoztam és formázott módon kiírtam a konzolra. A DOM előnye, hogy a teljes dokumentum memóriában rendelkezésre áll, ezért bármely elemhez könnyen hozzáférhetünk és módosíthatjuk is. A kód az OQH5NHDOMRead.java fileban található.

2.2 Adat-lekérdezés

Ebben a feladatban DOM alapú XML lekérdezéseket valósítottam meg. A feladat célja az volt, hogy legalább négy különböző lekérdezést hozzak létre XPath használata nélkül, kizárólag DOM függvények (getElementsByTagName, attribútum olvasás stb.) alkalmazásával.

A program elején beolvasom az XML-t, majd a normalize() metódussal egységesítem a struktúrát:

```
// XML dokumentum beolvasása
File xmlFile = new File("src/OQH5NH_XML.xml");

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();

// DOM dokumentum létrehozása
Document doc = builder.parse(xmlFile);
doc.getDocumentElement().normalize();
```

A lekérdezéseknél először mindig lekérem az adott elem NodeList-jeit (gazda, allat, kezeles, stb.), majd ciklusokkal járom be őket.

Az első lekérdezés célja, hogy minden gazda nevét és lakóhelyének városát kiírja. Itt egyszerűen végigmegyek a gazda elemek listáján, majd kiolvasom a „nev” és a „varos” gyerekelemeket:

```
NodeList gazdak = doc.getElementsByTagName("gazda");

for (int i = 0; i < gazdak.getLength(); i++) {
   Element gazda = (Element) gazdak.item(i);

   String nev = gazda.getElementsByTagName("nev").item(0).getTextContent();
   String varos = ((Element) gazda.getElementsByTagName("cim").item(0))
   .getElementsByTagName("varos").item(0).getTextContent();

   System.out.println(" - " + nev + " (" + varos + ")");
}
```

Eredmény például:

- Kovács Béla (Budapest

A második lekérdezésnél már kapcsolatokat is kezeltem: az allat elem g_a attribútuma tartalmazza, melyik gazdához tartozik. A program előbb bejárja az állatokat, majd ID alapján visszakeresi a gazdát:

```
NodeList allatok = doc.getElementsByTagName("allat");

for (int i = 0; i < allatok.getLength(); i++) {
   Element allat = (Element) allatok.item(i);

   String allatNev = allat.getElementsByTagName("nev").item(0).getTextContent();
   String faj = allat.getElementsByTagName("faj").item(0).getTextContent();

   // Gazda felderítése ID alapján
   String gazdaID = allat.getAttribute("g_a");
   String gazdaNev = "";

   for (int j = 0; j < gazdak.getLength(); j++) {
       Element gazda = (Element) gazdak.item(j);
```

```
        if (gazda.getAttribute("gazdaID").equals(gazdaID)) {
            gazdaNev = gazda.getElementsByTagName("nev").item(0).getTextContent();
        }
    }

    System.out.println(" - " + allatNev + " (" + faj + ") → Gazdája: " +
gazdaNev);
}
```

Konzol output például:

- Mázli (kutya) -> Gazdája: Kovács Béla

A harmadik lekérdezésnek a célja, hogy minden kezeléshez kiírja:

- mely állatot kezelték (a_k attribútum),
- ki végezte a kezelést (o_k attribútum),
- milyen összegért (ar).

A DOM-mal több lista közti keresést kellett megoldani:

```
NodeList kezelesek = doc.getElementsByTagName("kezeles");
NodeList orvosok = doc.getElementsByTagName("orvos");

for (int i = 0; i < kezelesek.getLength(); i++) {
    Element kezeles = (Element) kezelesek.item(i);

    String datum = kezeles.getElementsByTagName("datum").item(0).getTextContent();
    int ar =
Integer.parseInt(kezeles.getElementsByTagName("ar").item(0).getTextContent());

    // Állat keresése az a_k attributum alapján
    String allatID = kezeles.getAttribute("a_k");
    String allatNev = "";

    for (int j = 0; j < allatok.getLength(); j++) {
        Element allat = (Element) allatok.item(j);
        if (allat.getAttribute("allatID").equals(allatID)) {
         allatNev = allat.getElementsByTagName("nev").item(0).getTextContent();
     }
}
```

```
   // Orvos keresése o_k attribútum alapján
   String orvosID = kezeles.getAttribute("o_k");
   String orvosNev = "";

   for (int k = 0; k < orvosok.getLength(); k++) {
        Element orvos = (Element) orvosok.item(k);
        if (orvos.getAttribute("orvosID").equals(orvosID)) {
           orvosNev = orvos.getElementsByTagName("nev").item(0).getTextContent();
        }
   }

System.out.println(" - " + datum + ": " + allatNev + " kezelése (" + orvosNev +
") → " + ar + " Ft");
}
```

A negyedik lekérdezés a legösszetettebb, itt három entitás között kell kapcsolatot felderíteni:

- állat – kezelés – kezelésGyógyszer – gyógyszer

A kód lényege:

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

    Element allat = (Element) allatok.item(i);

    String allatID = allat.getAttribute("allatID");
    String allatNev = allat.getElementsByTagName("nev").item(0).getTextContent();

    System.out.println(" • " + allatNev + " gyógyszerei:");

    for (int j = 0; j < kezelesek.getLength(); j++) {
        Element kezeles = (Element) kezelesek.item(j);

        if (!kezeles.getAttribute("a_k").equals(allatID)) continue;

        String kezelesID = kezeles.getAttribute("kezelesID");

        for (int k = 0; k < kezGyList.getLength(); k++) {
            Element kapcsolat = (Element) kezGyList.item(k);

            if (kapcsolat.getAttribute("k_kg").equals(kezelesID)) {
```

És végül kiírja az adott állat kapott gyógyszereit.

Output minta:

- Mázli gyógyszerei:

  - Betadine

  - Cestal Plus

Az adat-lekérdezés során sikeresen hajtottam végre több, különböző összetettségű műveletet az XML dokumentum DOM struktúrájának bejárásával. XPath használata nélkül, kizárólag a DOM fa elemein történő manuális kereséssel értem el, hogy gazdák, állatok, kezelések és gyógyszerek között kapcsolatokat tudtam lekövetni és megjeleníteni. A lekérdezések eredményei jól szemléltették az XML felépítését, és azt, hogyan kapcsolódnak egymáshoz az adatok (például attribútumok alapján). A DOM kezelése során hasznos gyakorlati tapasztalatot szereztem az XML adatstruktúra bejárásában és az adatok kinyerésében.

2.3 Adatmódosítás

Az utolsó részfeladatom az XML dokumentum adatinak módosítását kellett végrehajtanom DOM segítségével. Az OQH5NHDOMModify.java osztályban megvalósítottam több különböző módosítást az XML állomány egyes elemein:

- szöveges értékeket változtattam meg,
- új tartalmat írtam be,
- és minden módosítást kiírtam konzolra.

A programban először beolvastam a dokumentumot, majd a megfelelő NodeList objektumon keresztül elértem a módosítandó elemeket. A változtatásokat setTextContent() metódussal írtam át, végül pedig kiírtam a konzolra, hogy milyen módosítás történt.

```
// XML dokumentum beolvasása
File xmlFile = new File("src/OQH5NH_XML.xml");

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.parse(xmlFile);
doc.getDocumentElement().normalize();
```

Itt történik a dokumentum betöltése Dom struktúraként. A normalize() gondoskodik arról, hogy a felesleges whitespace node-ok eltűnjenek, így könnyebben lehet bejárni az elemeket.

Első módosítás

Gazda telefonszámának megváltoztatása

```
NodeList gazdak = doc.getElementsByTagName("gazda");
Element elsoGazda = (Element) gazdak.item(0);
String regiTelefon =
elsoGazda.getElementsByTagName("telefonszam").item(0).getTextContent();
elsoGazda.getElementsByTagName("telefonszam").item(0)
        .setTextContent("+36309998877");
```

Az első gazda megkeresése után a telefonszámot lecseréltem. A régi és az új érték is kiírásra kerül a konzolra.

Második módosítás

Egy állat fajtájának átírása

```
NodeList allatok = doc.getElementsByTagName("allat");
Element masodikAllat = (Element) allatok.item(1);
String regiFaj =
masodikAllat.getElementsByTagName("faj").item(0).getTextContent();
masodikAllat.getElementsByTagName("faj").item(0).setTextContent("Sziámi macska");
```

Itt a második „allat” elem fajtáját módosítom. Ez is kiírásra kerül a konzolra, hogy látható legyen a változás.

Harmadik módosítás

Orvos beosztásának módosítása

```
NodeList orvosok = doc.getElementsByTagName("orvos");
Element harmadikOrvos = (Element) orvosok.item(2);
String regiBeosztas =
harmadikOrvos.getElementsByTagName("beosztas").item(0).getTextContent();
harmadikOrvos.getElementsByTagName("beosztas").item(0).setTextContent("Főorvos");
```

A harmadik orvos beosztása „Főorvos”-ra változik.

Negyedik módosítás

Gyógyszer adagolás átírása

```
NodeList gyogyszerek = doc.getElementsByTagName("gyogyszer");
Element elsoGyogyszer = (Element) gyogyszerek.item(0);
String regiAdagolas =
elsoGyogyszer.getElementsByTagName("adagolas").item(0).getTextContent();
elsoGyogyszer.getElementsByTagName("adagolas").item(0).setTextContent("20 mg/kg
naponta kétszer");
```

Az első gyógyszerhez tartozó adagolás információját átírtam egy új értékre.

Ötödik módosítás

Egészségügyi könyv megjegyzés módosítása

```
NodeList konyvek = doc.getElementsByTagName("egeszsegugyiKonyv");
Element elsoKonyv = (Element) konyvek.item(0);
String regiMegjegyzes =
elsoKonyv.getElementsByTagName("megjegyzes").item(0).getTextContent();
elsoKonyv.getElementsByTagName("megjegyzes").item(0)
        .setTextContent("Oltási könyv komplett, új ellenőrzés szükséges");
```

Ezzel a módosítással az XML dokumentumomban lévő megjegyzés mezőt frissítettem.

A feladat során egyértelműen látszott, hogy a DOM kezelés lehetővé teszi az XML dokumentum részletes bejárását és az adatok tetszőleges manipulációját.

A második feladat során a teljes XML adatkezelési folyamatot hajtottam végre: az XML dokumentum beolvasását, adatainak lekérdezését, valamint a struktúra módosítását DOM segítségével. A különböző részekben megvalósítottam az adatstruktúra értelmezését és kezelését Java nyelven, bemutatva, hogyan lehet a dokumentum elemeit bejárni, kiolvasni és módosítani.

A DOM megközelítés jól szemléltette, hogy az XML állomány hierarchikus szerkezetként könnyen kezelhető memóriában, és minden elem, attribútum, illetve szöveges tartalom közvetlenül elérhető.