JEGYZŐKÖNYV

Adatkezelés XML környezetben

Féléves feladat

Telekom Hibabejelentő

Készítette: **Varga Bence**
Neptunkód: **CKFEC9**

Dátum: 2024.11.18

Tartalomjegyzék (hiper-hivatkozott)

Bevezetés A beadandóban egy XML-alapú hibabejelentő rendszert mutatok be, amely TV, Telefon és Internet hibák rögzítésére és kezelésére szolgál.

A feladat leírása:

A beadandó ötletem a munkából adódott, mivel Telekomos ügyintézőként dolgozom. Ennek kapcsán egy olyan hibabejelentő adatbázist fogok bemutatni XML nyelven, amely a munkám során előforduló problémák kezelését segíti. Ez az adatbázis lehetőséget nyújt arra, hogy TV, Telefon, illetve Internet szolgáltatásokhoz kapcsolódó hibákat hatékonyan lehessen rögzíteni, nyomon követni és kezelni.

A hibabejelentő rendszer felépítése több szempontot figyelembe vesz. Tartalmazza az ügyfél adatait, például nevét, elérhetőségeit, valamint a bejelentett hiba pontos típusát és leírását. Emellett fontos szerepet kap a bejelentés időpontja és státusza is, például hogy az éppen "nyitott", "folyamatban lévő" vagy "lezárt" állapotban van-e. Az XML alapú struktúra lehetővé teszi a hibabejelentések gyors kereshetőségét és integrációját más rendszerekkel.

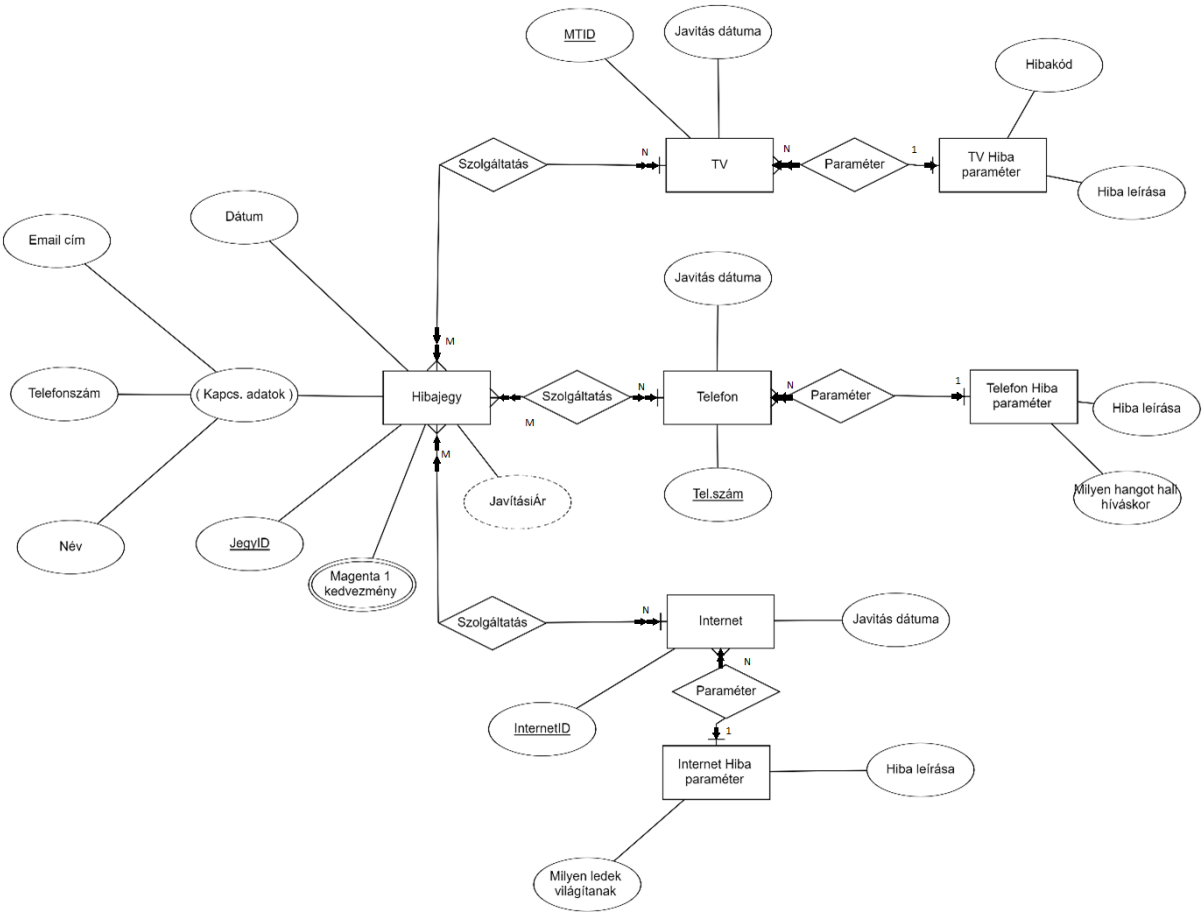
A projekt során részletesen bemutatom az XML fájl felépítését, a szükséges elemeket, attribútumokat, valamint azok hierarchiáját. Tervem szerint a rendszerhez tartozó egyes mezők között keresési és szűrési funkciók is megvalósíthatók, például hiba típus, időintervallum vagy ügyfél alapján. Ezen kívül foglalkozni fogok azzal is, hogy az adatbázis hogyan illeszthető egy meglévő ügyfélszolgálati rendszerbe, és milyen módon segíti a hatékony ügyintézést a mindennapi munka során.

Végezetül, az XML alapú hibabejelentő nemcsak a Telekom ügyfélszolgálati feladataihoz nyújt segítséget, hanem általánosítható más hasonló területekre is, ahol strukturált hibaadatok kezelésére van szükség.

1. Az ELSŐ feladat

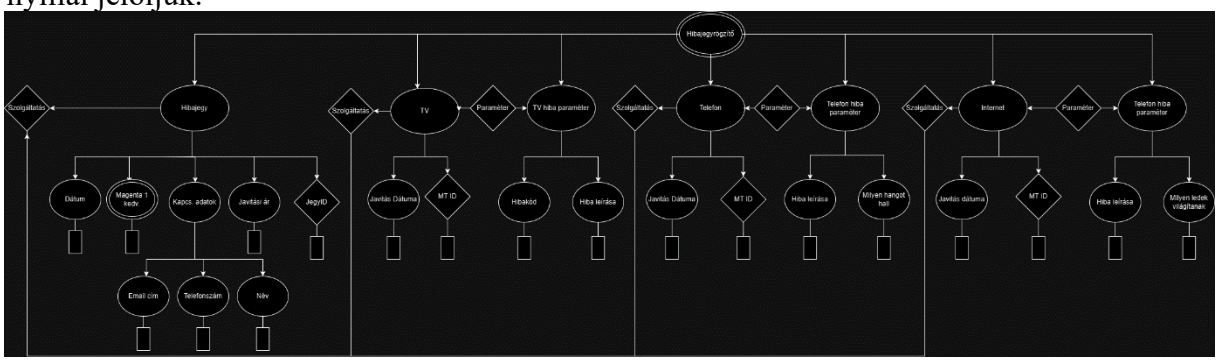
1.1 Az adatbázis ER modell tervezése

- **A Hibajegy egyed tulajdonságai**
    - JegyID: A Hibajegy egyed elsődleges kulcsa.
    - Kapcsolattartói adatok: Összetett tulajdonság. A vevő kapcsolattartói adatai.
    - Magenta 1 kedvezmény: Az ügyfél Magenta 1 kedvezménnyel rendelkezik, többértékű tulajdonság
- **A TV egyed tulajdonságai**
    - MTID: A TV egyed elsődleges kulcsa.
    - Javítás dátuma: Mikor várható a javítás
- **A Hiba paraméter egyed (TV) tulajdonságai**
    - Hibakód: Milyen hibát ír ki a TV
    - Hiba leírása: Milyen hibát észlel az ügyfél
- **Telefon egyed tulajdonságai**
    - Telefonszám: A Telefon egyed elsődleges kulcsa.
    - Javítás dátuma: Mikor várható a javítás
- **A Hiba paraméter egyed (Telefon) tulajdonságai**
    - Milyen hangot hall híváskor: Az ügyfelünk milyen hangot hall, amikor próbál hívni
    - Hiba leírása: Milyen hibát észlel az ügyfél
- **Internet egyed tulajdonságai**
    - InternetID: Az Internet egyed elsődleges kulcsa.
    - Javítás dátuma: Mikor várható a javítás
- **A Hiba paraméter egyed (Internet) tulajdonságai**
    - Milyen LED-ek világítanak a routeren? : Több fajta LED található a routeren, általában ezekből meglehet fejteni a hibát.
    - Hiba leírása: Milyen hibát észlel az ügyfél



1.2 **Az adatbázis konvertálása XDM modellre**

XDM modellnél háromféle jelölés alkalmazunk. Ezek az ellipszis, a rombusz, illetve a téglalap. Az ellipszis jelöli az elemeket, minden egyedből elem lesz, ezen felül a tulajdonságokból is. A rombusz jelöli az attribútumokat, amelyek a kulcs tulajdonságokból keletkeznek. A téglalap jelöli a szöveget, amely majd az XML dokumentumban fog megjelenni. Azoknak az elemeknek, amelyek többször is előfordulhatnak, a jelölése dupla ellipszissel történik. Az idegenkulcsok és a kulcsok közötti kapcsolatot szaggatott vonalas nyíllal jelöljük.



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

Az XDM modell alapján az XML dokumentumot az alábbi logika szerint készítettem el:

Először létrehoztam a gyökérelemet, ami a `<Hibajegy>` volt, mint a teljes hierarchia központi eleme. Ez tartalmazta az egyes hibajegyrekordokat `<HibajegyRecord>` formájában, amelyek az egyes hibabejelentések adatait képviselik. A hibajegyrekordokon belül gyerekelemeket definiáltam, például `<KapcsolattartoiAdatok>`, amelyek az ügyfél kapcsolattartási információit tartalmazzák.

A gyerekelemek közé tartoztak még a `<TV>`, `<Telefon>` és `<Internet>` elemek is. Minden ilyen elemnél attribútumként szerepeltek a kulcsok (például `MTID` a TV esetében) és idegen kulcsok, amelyek a szülőelemre (`JegyID`) hivatkoztak. A többszörös előfordulású elemeknél, például a `<Hiba>` esetében, több példányt hoztam létre, és az egyes hibák tulajdonságait, például a hibakódot és leírást, alárendelt elemekben ábrázoltam.

Ez a hierarchikus struktúra a relációs adatmodell logikai felépítését tükrözte, miközben az XML forma nyújtotta rugalmasságot kihasználva pontosan reprezentálta az adatok közötti kapcsolatokat.

```
<Hibajegy>
    <HibajegyRecord JegyID="1">
        <KapcsolattartoiAdatok>
            <Email>elso@gmail.com</Email>
            <Nev>Varga Bence</Nev>
            <Telefonszam>36705026225</Telefonszam>
        </KapcsolattartoiAdatok>
        <TV>
            <MTID>123456789</MTID>
            <Hiba>
                <Hibakod>E:404</Hibakod>
                <HibaLeiras>Nincs kép</HibaLeiras>
            </Hiba>
        </TV>
    </HibajegyRecord>
</Hibajegy>
```

1.4 **Az XML dokumentum alapján XMLSchema készítése**

Az XML dokumentumhoz szükséges validációt elősegítő sémát az alábbi módon hoztam létre:

Először létrehoztam az egyszerű elemeket, amelyeket később a komplex típusok felépítésében használtam. Ezekhez saját típusokat definiáltam, összesen hatot. Ezek között volt olyan, amelyet regex segítségével szűkítettem le, például az azonosítók formátumára, valamint volt enumeration típus is, amely előre meghatározott értékkészlettel rendelkezik.

Ezek után megalkottam a teljes XML dokumentum felépítését. A komplex típusokban az egyszerű típusokra hivatkoztam, és meghatároztam az elemek minimális és maximális előfordulását, figyelembe véve, hogy egy többszöri előfordulású elemhez az XML-ben legalább három példány tartozik. Minden komplex típushoz megadtam az attribútumokat is, amelyekhez definiáltam az elsődleges kulcsokat, idegen kulcsokat, valamint különleges kapcsolatokat, például `key` és `keyref` használatával.

A sémát úgy alakítottam ki, hogy megfeleljen az XML dokumentum hierarchiájának, biztosítva az adatok integritását és helyességét.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Hibajegy">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="HibajegyRecord"
type="HibajegyRecordType" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
        <xs:key name="HibajegyPrimaryKey">
            <xs:selector xpath="HibajegyRecord"/>
            <xs:field xpath="@JegyID"/>
        </xs:key>
    </xs:element>

    <xs:complexType name="HibajegyRecordType">
        <xs:sequence>
            <xs:element name="KapcsolattartoiAdatok"
type="KapcsolattartoiAdatokType"/>
        </xs:sequence>
        <xs:attribute name="JegyID" type="xs:ID"
use="required"/>
    </xs:complexType>
</xs:schema>
```

2.1 Adatolvasás – DOMReadCKFEC9.java

A **DOMReadCKFEC9.java** program célja, hogy egy XML fájlt beolvasson és annak tartalmát kiírja a konzolra. Ehhez a **DOM (Document Object Model) API-t** használja, amely lehetővé teszi az XML dokumentumok hierarchikus kezelését és feldolgozását.

A program először betölti az XML fájlt, és azt egy **DOM dokumentummá** alakítja, amely a fájl szerkezetét egy objektumként ábrázolja a memóriában. Ezután a `normalize()` metódus segítségével biztosítja, hogy az XML dokumentum helyesen legyen feldolgozva (eltávolítja a felesleges üres szövegeket és egyéb hibákat).

A dokumentum elemeit egyenként végigjárva a program kiírja azok nevét és tartalmát. Mivel az XML struktúrája hierarchikus, a program minden egyes csomópontot (elemet) feldolgoz, és ha az adott elem tartalmaz adatokat, azokat is megjeleníti a konzolon.

A kód kivételkezelést is tartalmaz, így ha a fájl betöltése közben hiba történik (például a fájl nem található, vagy helytelen formátumú), a program nem omlik össze, hanem a hibaüzenetet kiírja a képernyőre.

Ez a megközelítés lehetővé teszi, hogy könnyedén beolvassuk és feldolgozzuk az XML fájlokat, és egyszerűen kiírjuk azok tartalmát konzolos alkalmazásokban.

```
File inputFile = new File("XMLCKFEC9.xml");
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(inputFile);
doc.getDocumentElement().normalize();

System.out.println("Gyökérelem: " +
doc.getDocumentElement().getNodeName());
NodeList nodeList = doc.getElementsByTagName("*");
for (int i = 0; i < nodeList.getLength(); i++) {
    Node node = nodeList.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element elem = (Element) node;
        System.out.println("Elem neve: " + elem.getNodeName()
+
                           ", Tartalom: " +
elem.getTextContent());
    }
}
```

2.2 Adatírás – DOMWriteCKFEC9.java

A **DOMWriteCKFEC9.java** program célja, hogy egy XML fájlt fa struktúrában kiírjon a konzolra, majd a módosított vagy új adatokkal ellátott XML dokumentumot elmentse egy új fájlba. Ehhez a **DOM API**-t használja, amely lehetővé teszi az XML dokumentumok kezelését, módosítását és új fájlba írását.

A program először betölti az XML fájlt, majd a **DOM dokumentumot** átalakítja egy objektumá, amely tartalmazza a teljes dokumentum hierarchikus szerkezetét. A **fa struktúra** kiírásához rekurzív metódust alkalmaz, amely végigiterál az XML dokumentum minden elemén, és azok nevét, illetve tartalmát kiírja a konzolra. A rekurzió biztosítja, hogy minden gyermekelem (bármilyen szintű) is feldolgozásra kerüljön.

A második fontos funkció a fájl mentése. Miután az XML dokumentumot módosítottuk (akár új elemek hozzáadásával, akár meglévők módosításával), a program a **Transformer API** segítségével új fájlt hoz létre. A **Transformer** osztály képes szép formázásban menteni az XML dokumentumot, azaz megőrzi a hierarchiát, és az adatokat jól olvasható formában rendezi.

Ez a program tehát két fő feladatot lát el:

1. Kiírja az XML fájl tartalmát fa struktúrában a konzolra.
2. Az XML dokumentumot módosítva új fájlt generál, amely tartalmazza az összes módosítást.

A program hibakezelést is tartalmaz, így ha bármilyen probléma merülne fel a fájlok betöltésével vagy mentésével kapcsolatban, a megfelelő hibaüzenetek kiírásra kerülnek, és a program nem omlik össze.

Ez a megközelítés ideális eszköz az XML fájlok feldolgozására, módosítására, és új fájlok generálására.

```
public static void printTree(Node node, int depth) {
   for (int i = 0; i < depth; i++) {
      System.out.print("  ");
   }
   if (node.getNodeType() == Node.ELEMENT_NODE) {
      System.out.println("<" + node.getNodeName() + ">");
   }
   NodeList childNodes = node.getChildNodes();
   for (int i = 0; i < childNodes.getLength(); i++) {
      printTree(childNodes.item(i), depth + 1);
   }
   if (node.getNodeType() == Node.ELEMENT_NODE) {
      for (int i = 0; i < depth; i++) {
         System.out.print("  ");
      }
      System.out.println("</" + node.getNodeName() + ">");
   }
}
```

2.3 Adatlekérdezés – DOMQueryCKFEC9.java

A **DOMQueryCKFEC9.java** program célja, hogy egy XML dokumentumból adatokat kérdezzen le a **DOM API** segítségével. Az adatlekérdezés során a program különböző elemeket keres az XML fájlban, és azokat a konzolra írja ki, megfelelve a megadott kritériumoknak.

A program első lépésként betölti az XML fájlt és **DOM dokumentummá** alakítja azt. Ezután az **XPath** segítségével hajt végre lekérdezéseket az XML dokumentumban. Az XPath egy rendkívül erőteljes eszköz, amely lehetővé teszi a dokumentum elemeinek pontos meghatározását a struktúra alapján. A programban szereplő lekérdezések célja, hogy különböző típusú adatokat, például hibajegyeket, kapcsolattartói adatokat, vevőket, vagy hibákat kérdezzenek le, és a találatokat konzolon jelenítsék meg.

A programban különféle lekérdezéseket végezhetünk:

- Kiválaszthatjuk az adott **`Hibajegy`** elemet a `JegyID` alapján.
- Lekérdezhetjük a **`KapcsolattartoiAdatok`** elemeket, és kiírhatjuk azok tartalmát.
- Különböző összetett szűréseket végezhetünk, például kereshetünk minden olyan **`Telefon`** elemet, ahol a telefonos adatokat tartalmazza.

A program végigjárja a dokumentumot, lekérdezi az összes példányt ugyanazzal a taggel, majd feldolgozza azokat. A lekérdezéseket úgy alakítjuk, hogy ne szükségszerűen az XPath kifejezésekkel történjenek, hanem inkább az XML dokumentumban található elemeket és azok kapcsolatait használja fel a program.

A kód végén minden találat kiírásra kerül a konzolra, és a program adatai kiértékelhetőek. Ez a fajta lekérdezés hasznos, ha csak egy adott típusú adatot akarunk keresni egy nagyobb XML dokumentumból, vagy ha összetettebb adatokat szeretnénk szűrni.

A programban a kivételkezelés biztosítja, hogy ha a keresett elem nem található, vagy ha egy hiba történik az XML feldolgozása közben, akkor a megfelelő hibaüzenet jelenik meg, és a program nem omlik össze.

```
XPath xPath = XPathFactory.newInstance().newXPath();
String expression =
"/Hibajegy/HibajegyRecord[@JegyID='1']/KapcsolattartoiAdatok/N
ev";
Node node = (Node) xPath.compile(expression).evaluate(doc,
XPathConstants.NODE);
System.out.println("Kapcsolattartó neve: " +
node.getTextContent());
```

2.4 Adatmódosítás – DOMModifyNeptunkod.java

A **DOMModifyNeptunkod.java** program célja, hogy egy XML fájlban adatokat módosítson és új adatokat adjon hozzá. A program a **DOM API**-t használja az XML dokumentumok manipulálására.

1. **XML fájl betöltése**: A fájlt DOM dokumentummá alakítja.
2. **Módosítások végrehajtása**: Új elemeket hoz létre, meglévő adatokat módosít (pl. telefonszám, fizetés), és szükség esetén törli a felesleges elemeket.
3. **Fájl mentése**: A módosított adatokat új fájlba menti.
4. **Kivételkezelés**: A program kezeli a fájlok betöltésével és mentésével kapcsolatos hibákat.

Ez a program lehetővé teszi az XML fájlok egyszerű módosítását és a változások elmentését.

```
public class DOMModifyNeptunkod {
    public static void main(String[] args) {
        try {
            // XML betöltése
            Document doc =
DocumentBuilderFactory.newInstance()
                    .newDocumentBuilder()
                    .parse(new File("XMLNeptunkod.xml"));
            doc.getDocumentElement().normalize();

            // Új elem hozzáadása
            Element newElement = doc.createElement("ÚjElem");
            newElement.setTextContent("Új tartalom");
            doc.getDocumentElement().appendChild(newElement);

            // Elem módosítása
            Node telefon =
doc.getElementsByTagName("Telefon").item(0);
            if (telefon != null)
telefon.setTextContent("Módosított tartalom");

            // XML mentése
            Transformer transformer =
TransformerFactory.newInstance().newTransformer();
            transformer.transform(new DOMSource(doc), new
StreamResult(new File("XMLNeptunkod_Modified.xml")));

            System.out.println("XML sikeresen módosítva.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```