JEGYZŐKÖNYV

Adatkezelés XML környezetben

Féléves feladat

Vendéglátás

Készítette: István Miklós

Neptunkód: VN7XCW

Dátum: 2023.12.05

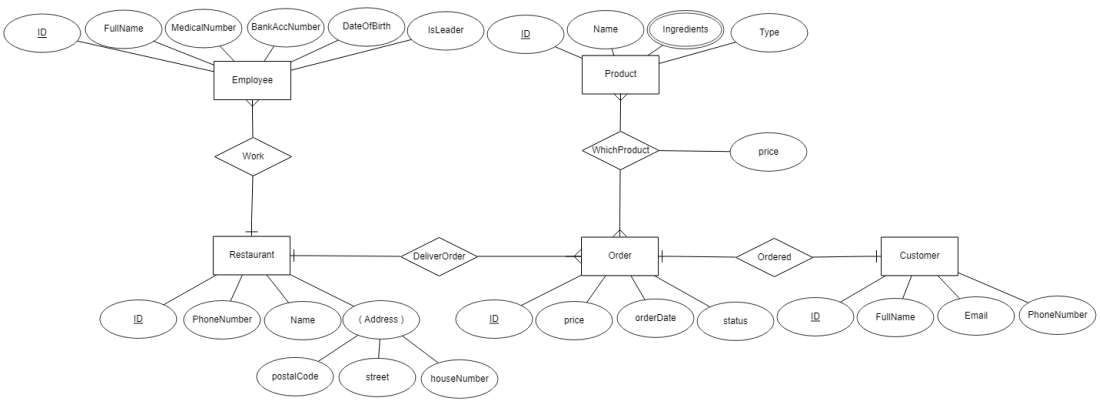
1. Tartalom

1. Feladat leírása

A beadandó témája egy olyan adatbázis, amely több cég által kezelt szalonok különböző adatait tartja számon. Lehetőség van egyes szolgáltatások, illetve vendégek, vagy akár az adott helyszín adatainak lekérdezésére.

- Restaurant egyed tulajdonságai
  - id: a kulcs
  - phoneNumber: az étterem telefonszáma
  - name: az étterem neve
  - address: összetett tulajdonság, az étterem címe
- Employee egyed tulajdonságai
  - id: a dolgozó egyedi kulcsa
  - fullName: a dolgozó teljes neve
  - MedicalNumber: a dolgozó orvosi száma
  - BankAccNumber: a dolgozó számlaszáma
  - DateOfBirth: a dolgozó születési dátuma
  - IsLeader: vezető e
- Order egyed tulajdonságai
  - id: az rendelés egyed kulcsa
  - status: aktív e még a rendelés
  - orderDate: rendelés időpontja
  - price: a rendelés végösszege
- Customer egyed tulajdonságai
  - id: a vásárló egyedi azonosítója
  - fullName: a vásárló teljes neve
  - email: a vásárló email címe
  - phoneNumber: a vásárló telefonszáma
- Product egyed tulajdonságai
  - id: termék kulcsa
  - name: a termék neve
  - ingredients: a termék készítéséhez felhasznált alapanyagok, mely lehet többértékű
  - type: a termék típusa

A feladat ER modellje:



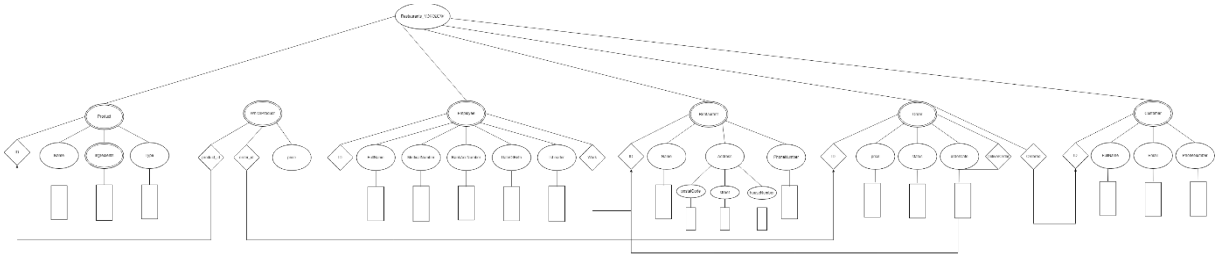
Az egyedek közötti kapcsolatok:

- Employee és Restaurant közötti kapcsolat: Work
    - Egy-több kapcsolat van közöttük, mivel egy alkalmazott csak egy étteremnél dolgozhat.
- Restaurant és Order közötti kapcsolat: DeliverOrder
    - Egy-több kapcsolat van közöttük, mivel egy étteremhez tartozhat több rendelés is, de egy adott rendelés nem tartozhat több étteremhez.
- Order és Product közötti kapcsolat: WhichProduct
    - Több-több kapcsolat van közöttük, mivel egy termék tartozhat több rendeléshez is, illetve egy rendeléshez is tartozhat több termék.
- Order és Customer közötti kapcsolat: Ordered
    - Egy-egy kapcsolat van közöttük, mivel csak egy rendelés lehet aktív egy vevőnek, és egy rendelés csak egy vevőhöz tartozhat.

2. Az ER modell konvertálása XDM modellé

Az XDM modell elkészítése során háromféle jelölést alkalmaztam: az ellipszist, rombuszt és téglalapot. Minden egyed egy ellipszis elemként jelenik meg ebben a modellben, melyek mivel többértékűek így duplával kell jelölni őket. Megjelennek itt még a kulcs tulajdonságok is rombuszként, illetve a szövegeket a téglalapok reprezentálják.

A feladat XDM modellje:



3. XML dokumentum készítése

Következő lépésnek az XML dokumentumot készítettem el az XDM modell segítségével. Minden dupla ellipszissel jelölt elemhez legalább három példányt készítettem el. A kulcs elemek is megjelennek a gyerekelemeknél attribútumokként, illetve az idegenkulcsok is.

Kód:

```
<?xml version="1.0" encoding="UTF-8"?>
<Restaurants_VN7XCW     xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="XMLSchema_VN7XCW.xsd">

    <!-- Restaurant instance -->
    <restaurant id="R001">
        <name>Sample Restaurant</name>
        <phoneNumber>06703334445</phoneNumber>
```

```
    <address>
        <postalCode>22222</postalCode>
        <street>Maple street</street>
        <houseNumber>66</houseNumber>
    </address>
</restaurant>

<restaurant id="R002">
    <name>Sample Restaurant2</name>
    <phoneNumber>06703338888</phoneNumber>
    <address>
        <postalCode>22222</postalCode>
        <street>Maple street</street>
        <houseNumber>22</houseNumber>
    </address>
</restaurant>

<restaurant id="R003">
    <name>Sample Restaurant3</name>
    <phoneNumber>06703339999</phoneNumber>
    <address>
        <postalCode>22222</postalCode>
        <street>Maple street</street>
        <houseNumber>11</houseNumber>
    </address>
</restaurant>

<!--Employees-->
<employee id="E001" work="R001">
    <fullName>John Doe</fullName>
    <medicalNumber>M123</medicalNumber>
    <bankAccountNumber>ABC123456</bankAccountNumber>
    <dateOfBirth>1990-05-15</dateOfBirth>
    <isLeader>No</isLeader>
</employee>

<employee id="E002" work="R002">
    <fullName>John Doe2</fullName>
    <medicalNumber>M222</medicalNumber>
    <bankAccountNumber>ABC222222</bankAccountNumber>
    <dateOfBirth>1997-08-05</dateOfBirth>
    <isLeader>Yes</isLeader>
</employee>

<employee id="E003" work="R003">
    <fullName>John Doe3</fullName>
```

```
    <medicalNumber>M333</medicalNumber>
    <bankAccountNumber>ABC333333</bankAccountNumber>
    <dateOfBirth>2002-01-22</dateOfBirth>
    <isLeader>No</isLeader>
</employee>

<!-- Order instance -->
<order id="O001" deliverOrder="R001" ordered="C001">
    <price>2500</price>
    <orderDate>2023.10.15</orderDate>
    <status>Active</status>
</order>

<order id="O002" deliverOrder="R002" ordered="C002">
    <price>15000</price>
    <orderDate>2023.04.15</orderDate>
    <status>Shipped</status>
</order>

<order id="O003" deliverOrder="R003" ordered="C003">
    <price>4560</price>
    <orderDate>2023.12.15</orderDate>
    <status>Shipped</status>
</order>

<!-- Product instance -->
<product id="P001">
    <name>Product Name</name>
    <type>Type</type>
    <ingredient>ham</ingredient>
    <ingredient>cheese</ingredient>
    <ingredient>corn</ingredient>
</product>

<product id="P002">
    <name>Product Name2</name>
    <type>Type2</type>
    <ingredient>cheese</ingredient>
    <ingredient>onion</ingredient>
</product>

<product id="P003">
    <name>Product Name3</name>
    <type>Type3</type>
    <ingredient>ham</ingredient>
    <ingredient>pineapple</ingredient>
```

```
        <ingredient>corn</ingredient>
    </product>

    <!-- Customer instance -->
    <customer id="C001">
        <fullName>Jane</fullName>
        <email>janesmith@example.com</email>
        <phoneNumber>06704445556</phoneNumber>
    </customer>

    <customer id="C002">
        <fullName>Jane2</fullName>
        <email>janesmith2@example.com</email>
        <phoneNumber>06304543212</phoneNumber>
    </customer>

    <customer id="C003">
        <fullName>Jane3</fullName>
        <email>janesmith3@example.com</email>
        <phoneNumber>06307778886</phoneNumber>
    </customer>

    <!--Which product kapcsolat-->

    <which_product product_id="P001" order_id="O001">
        <price>2600</price>
    </which_product>

    <which_product product_id="P002" order_id="O002">
        <price>3200</price>
    </which_product>

    <which_product product_id="P003" order_id="O003">
        <price>3690</price>
    </which_product>

</Restaurants_VN7XCW>
```

4. XMLSchema készítése

Mindezek után az XMLSchemát készítettem el az XML dokumentum validációjához. Először létrehoztam az egyszerű típusokat, amelyekre később tudok referálni, ezek után pedig az egyedi, saját típusokat is, amelyekhez regexek mellett enumerationt is használtam. Ezután következett a tényleges felépítés, ahol a root elem parent elemeit complexType segítségével határoztam meg, amin belül a gyerekelemeknél tudtam referálni az előzőleg létrehozott

egyszerű típusokra. Mindezek után meghatároztam a kulcsokat, idegenkulcsokat, illetve a unique kikötéseket is.

Az XMLSchema kód

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

    <!--Egyszerű elemek-->

    <xs:element name="name" type="xs:string" />
    <xs:element name="phoneNumber" type="phoneType" />
    <xs:element name="postalCode" type="xs:int" />
    <xs:element name="street" type="xs:string" />
    <xs:element name="houseNumber" type="xs:int" />

    <xs:element name="fullName" type="xs:string" />
    <xs:element name="medicalNumber" type="xs:string" />
    <xs:element name="bankAccountNumber" type="xs:string" />
    <xs:element name="dateOfBirth" type="timeType" />
    <xs:element name="isLeader" type="isLeader" />

    <xs:element name="price" type="xs:int" />
    <xs:element name="orderDate" type="timeType" />
    <xs:element name="status" type="statusType" />

    <xs:element name="type" type="xs:string" />
    <xs:element name="ingredient" type="xs:string" />
    <xs:element name="idotartam" type="xs:int" />

    <xs:element name="email" type="emailType" />

    <!--Saját típusok-->

    <xs:simpleType name="isLeader">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Yes" />
            <xs:enumeration value="No" />
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="statusType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Active" />
            <xs:enumeration value="Shipped" />
        </xs:restriction>
    </xs:simpleType>
```

```
    <xs:simpleType name="phoneType">
        <xs:restriction base="xs:string">
            <xs:pattern value="(06(20|30|31|50|60|70)\d{7})"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="timeType">
        <xs:restriction base="xs:string">
                        <xs:pattern   value="([12]\d{3}.(0[1-9]|1[0-2]).(0[1-
9]|[12]\d|3[01]))" />
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="emailType">
        <xs:restriction base="xs:string">
            <xs:pattern value="[\w\.]+@([\w]+\.)+[\w]{2,4}" />
        </xs:restriction>
    </xs:simpleType>

    <!--Felépítés-->

    <xs:element name="Restaurants_VN7XCW">
        <xs:complexType>
            <xs:sequence>
                              <xs:element   name="restaurant"   minOccurs="1"
maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="name" />
                            <xs:element ref="phoneNumber" />
                                  <xs:element  name="address"  minOccurs="1"
maxOccurs="1">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element ref="postalCode" />
                                        <xs:element ref="street" />
                                        <xs:element ref="houseNumber" />
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" />
                    </xs:complexType>
                </xs:element>
```

```
                                <xs:element   name="employee"   minOccurs="1"
maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="fullName" />
                            <xs:element ref="medicalNumber" />
                            <xs:element ref="bankAccountNumber" />
                            <xs:element ref="dateOfBirth" />
                            <xs:element ref="isLeader" />
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" />
                        <xs:attribute name="work" type="xs:string" />
                    </xs:complexType>
                </xs:element>
               <xs:element name="order" minOccurs="1" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="price" />
                            <xs:element ref="orderDate" />
                            <xs:element ref="status" />
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" />
                         <xs:attribute name="deliverOrder" type="xs:string"
/>
                        <xs:attribute name="ordered" type="xs:string" />
                    </xs:complexType>
                </xs:element>
                                 <xs:element   name="product"   minOccurs="1"
maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="name" />
                            <xs:element ref="type" />
                                 <xs:element ref="ingredient" minOccurs="1"
maxOccurs="unbounded" />
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" />
                    </xs:complexType>
                </xs:element>
                                <xs:element   name="customer"   minOccurs="1"
maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="fullName" />
                            <xs:element ref="email" />
                            <xs:element ref="phoneNumber" />
```

```
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" />
                    </xs:complexType>
                </xs:element>
                            <xs:element  name="which_product"  minOccurs="1"
maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="price" />
                        </xs:sequence>
                        <xs:attribute name="product_id" type="xs:string" />
                        <xs:attribute name="order_id" type="xs:string" />
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>

        <!--Kulcsok-->

        <xs:key name="restaurant_kulcs">
            <xs:selector xpath="restaurant" />
            <xs:field xpath="@id" />
        </xs:key>

        <xs:key name="employee_kulcs">
            <xs:selector xpath="employee" />
            <xs:field xpath="@id" />
        </xs:key>

        <xs:key name="order_kulcs">
            <xs:selector xpath="order" />
            <xs:field xpath="@id" />
        </xs:key>

        <xs:key name="product_kulcs">
            <xs:selector xpath="product" />
            <xs:field xpath="@id" />
        </xs:key>

        <xs:key name="customer_kulcs">
            <xs:selector xpath="customer" />
            <xs:field xpath="@id" />
        </xs:key>

        <!--Idegen kulcsok-->
```

```
        <xs:keyref refer="restaurant_kulcs" name="employee_idegen_kulcs">
            <xs:selector xpath="employee" />
            <xs:field xpath="@work" />
        </xs:keyref>

                                    <xs:keyref       refer="restaurant_kulcs"
name="order_restaurant_idgen_kulcs">
            <xs:selector xpath="order" />
            <xs:field xpath="@deliverOrder" />
        </xs:keyref>

       <xs:keyref refer="customer_kulcs" name="order_customer_idegen_kulcs">
            <xs:selector xpath="order" />
            <xs:field xpath="@ordered" />
        </xs:keyref>

                                       <xs:keyref        refer="product_kulcs"
name="which_product_product_idgen_kulcs">
            <xs:selector xpath="which_product" />
            <xs:field xpath="@product_id" />
        </xs:keyref>

                                         <xs:keyref         refer="order_kulcs"
name="which_product_order_idgen_kulcs">
            <xs:selector xpath="which_product" />
            <xs:field xpath="@order_id" />
        </xs:keyref>

        <!--1:1-->

        <xs:unique name="unique_order">
            <xs:selector xpath="order" />
            <xs:field xpath="@ordered" />
        </xs:unique>

    </xs:element>

</xs:schema>
```

5. DOM adatolvasás

Először is meghatároztam azt, hogy melyik az a fájl amiből a beolvasást végezni kell. Létrehoztam a megfelelő változókat, illetve egy metódust is amelynek segítségével a tartalmat ki lehet írni a konzolra. Ebben megnézi, hogy milyen attribútumai vannak az adott Node-nak, illetve hogy vannak e gyerekelemei, ha igen, akkor kiírja őket.

Kód:

```
package DOMParse;

import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.File;
public class DomReadVN7XCW {
    public static void main(String[] args) {

        try {

            File xmlFile = new File("src/XML_VN7XCW.xml");

            if(!xmlFile.exists())
            {
                System.out.println("The file not found.");
                return;
            }

            DocumentBuilderFactory                dbFactory                =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(xmlFile);
            doc.getDocumentElement().normalize();

            Element rootElement = doc.getDocumentElement();
            System.out.println("Root          element:          "          +
rootElement.getNodeName());

            WriteOutContent(rootElement, "");

        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    private static void WriteOutContent(Node node, String indent) {
        if (node.getNodeType() == Node.ELEMENT_NODE) {
            System.out.println(indent + node.getNodeName());

            if (node.hasAttributes()) {
                NamedNodeMap attrib = node.getAttributes();
                for (int i = 0; i < attrib.getLength(); i++) {
                    Node attribute = attrib.item(i);
                    System.out.println(indent + attribute.getNodeName() + "
= " + attribute.getNodeValue());
                }
            }

            if (node.hasChildNodes()) {
                NodeList childList = node.getChildNodes();
                for (int i = 0; i < childList.getLength(); i++) {
                    Node child = childList.item(i);
                    WriteOutContent(child, indent + "  ");
                }
            }
        } else if (node.getNodeType() == Node.TEXT_NODE) {
            String data = node.getNodeValue().trim();
            if (!data.isEmpty()) {
```

```
                System.out.println(indent + data);
            }
        }
    }

}
```

6. DOM adatmódosítás

Dinamikussá tudtam tenni az adatmódosítást olyan szinten, hogy a user-től kérem be, hogy melyik az az elem, amit szeretne módosítani, pontosan milyen attribútummal, kulccsal rendelkezik, és melyik az az elem, amit benne módosítani szeretne, és mire.

Kód:

```
package DOMParse;

import org.w3c.dom.*;

import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.util.Scanner;

public class DOMModify_VN7XCW {

    public static void main(String[] args) {

        try {
            File xmlFile = new File("XML_VN7XCW.xml");

                                    DocumentBuilderFactory     dbFactory     =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(xmlFile);
            doc.getDocumentElement().normalize();

            Scanner sc = new Scanner(System.in);

             System.out.println("Add the name of the element which you want
to modify: ");
            String elementName = sc.nextLine();

            System.out.println("Add ID of the element: ");
            String elementID = sc.nextLine();
```

```
            System.out.println("Add the name of the attribute: ");
            String propertyName = sc.nextLine();

            System.out.println("Add new value: ");
            String newValue = sc.nextLine();

           if (!modifyElementByID(doc, elementName, elementID, propertyName,
newValue))
            {
                System.out.println("You added wrong input!!!");
            }

            sc.close();

            writeToFile(doc, "Modified_XML_VN7XCW.xml");

            System.out.println("Data successfully modified.");

        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    public static boolean modifyElementByID(Document doc, String elementName,
String elementID,
                                            String propertyName, String newValue)
{
        NodeList nodeList = doc.getElementsByTagName(elementName);

        for (int i = 0; i < nodeList.getLength(); i++) {
            Node node = nodeList.item(i);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element element = (Element) node;
                                                NodeList     childNodes     =
element.getElementsByTagName(propertyName);
                Node childNode = childNodes.item(0);
                childNode.setTextContent(newValue);
            }
            else {
                return false;
            }
        }
        return true;
    }
```

```
    public static void writeToFile(Document doc, String filename) {
        try {
                                TransformerFactory    transformerFactory    =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");

            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(new File(filename));
            transformer.transform(source, result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

7. DOM adatlekérdezés

Ezt is dinamikussá tudtam tenni, úgy, mint az adatmódosítást is. Ennél meg kell adni, hogy melyik az az elem, amit le szeretnénk kérdezni és milyen id-val rendelkezik. Ezt XPath segítségével valósítottam meg. Természetesen végre lehetne hajtani más fajta lekérdezéseket is, más metódusokkal, akkor az xpathQueryt kellene megváltoztatni hozzá. Pl.: meg lehetne oldani azt is, hogy a restaurant elem első 2 elemét írja ki, vagy hogy csak azokat a termékeket írja ki, amelyek többe kerülnek, mint 2000Ft.

Kód:

```
package DOMParse;

import org.w3c.dom.*;

import javax.xml.parsers.*;
import java.io.File;
import java.util.Scanner;
import javax.xml.xpath.*;

public class DomQueryVN7XCW {
    public static void main(String[] args) {
        try {
            File xmlFile = new File("XML_VN7XCW.xml");

            if (!xmlFile.exists()) {
```

```
                System.out.println("The file not found.");
                return;
            }

                                    DocumentBuilderFactory     dbFactory     =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(xmlFile);
            doc.getDocumentElement().normalize();


            //LEKÉRDEZNI KÍVÁNT ELEMENT LEKÉRDEZÉSE

            Scanner sc = new Scanner(System.in);

            System.out.println("Add the name of the element: ");

            String elementName = sc.nextLine();

            System.out.println("Add ID of the element: ");

            String elementID = sc.nextLine();

            sc.close();

            selectElementByID(doc, elementName, elementID);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void selectElementByID(Document doc, String elementName,
String elementID) {
        NodeList nodeList = doc.getElementsByTagName(elementName);
        String xpathQuery = "//" + elementName + "[@id='" + elementID + "']";

        // Evaluate the XPath expression and retrieve the matching node
        XPathFactory xPathfactory = XPathFactory.newInstance();
        XPath xpath = xPathfactory.newXPath();
        XPathExpression expr = null;
        try {
            expr = xpath.compile(xpathQuery);
        } catch (XPathExpressionException e) {
            System.out.println("Element not found.");
            throw new RuntimeException(e);
```

```
        }
        Node result = null;
        try {
            result = (Node) expr.evaluate(doc, XPathConstants.NODE);
        } catch (XPathExpressionException e) {
            System.out.println("Element not found.");
            throw new RuntimeException(e);
        }

        // Process the result
        if (result != null) {
            // Access the matched node and print its data
            Element element = (Element) result;
            NodeList children = element.getChildNodes();
            for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                            System.out.println(child.getNodeName() + ": " +
child.getTextContent().trim());
                }
            }
        } else {
            System.out.println("Element not found.");
        }
    }
}
```

8. DOM adatírás

Ez a kódmegvalósítás hasonlít az adatbeolvasásra, annyi különbséggel, hogy itt nem csak a konzolra íratom ki a tartalmat, hanem egy új fájlba is. Ugyanúgy végig iterálok a beolvasott fájlon, kinyerem a tartalmát, majd a transformer és streamResult segítségével íratom ki ezt egy új fájlba.

Kód:

```
package DOMParse;

import java.io.File;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
```

```
public class DOMWrite_VN7XCW {
    public static void main(String[] args) {
        try {

            File xmlFile = new File("XML_VN7XCW.xml");
                                DocumentBuilderFactory    dbFactory    =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(xmlFile);
            doc.getDocumentElement().normalize();

            Element rootElement = doc.getDocumentElement();
           System.out.println("Root element: " + rootElement.getNodeName());
            writeOutContent(rootElement, "");

            writeToFile(doc, "New_XML_VN7XCW.xml"); // Create new xml file

                 System.out.println("Updated  version  of  XML  saved  into
New_XML_VN7XCW.xml file");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void writeOutContent(Node node, String indent) {
        if (node.getNodeType() == Node.ELEMENT_NODE) {
            System.out.println(indent + node.getNodeName());

            if (node.hasAttributes()) {
                NamedNodeMap attrib = node.getAttributes();
                for (int i = 0; i < attrib.getLength(); i++) {
                    Node attribute = attrib.item(i);
                     System.out.println(indent + attribute.getNodeName() + "
= " + attribute.getNodeValue());
                }
            }

            if (node.hasChildNodes()) {
                NodeList childList = node.getChildNodes();
                for (int i = 0; i < childList.getLength(); i++) {
                    Node child = childList.item(i);
                    writeOutContent(child, indent + "  ");
                }
            }
        } else if (node.getNodeType() == Node.TEXT_NODE) {
```

```
            String datas = node.getNodeValue().trim();
            if (!datas.isEmpty()) {
                System.out.println(indent + datas);
            }
        }
    }

    public static void writeToFile(Document doc, String filename) {
        try {
                                TransformerFactory    transformerFactory    =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");

            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(new File(filename));
            transformer.transform(source, result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```