JEGYZŐKÖNYV

Adatkezelés XML környezetben

Féléves feladat

Étterem kezelés

Készítette: Biszterszky Mátyás

Neptunkód: L27NCJ

Dátum: 2023.12.11

Tartalomjegyzék:

Bevezetés:

1.Feladat

1a bevezetés

A beadandó feladatom során egy étterem kezelő rendszer szerkezetét modelleztem le XML struktúrában. Az éttermek nyilván tartották a rendeléseket, az ételeket, valamint az ügyféladatokat és egyéb szükséges információkat, hogy hatékonyabban tudják ellátni a vendégeiket. A modern világban egyre több helyen fordítanak figyelmet az informatikai megoldások beillesztésére a vendéglátóiparba, hiszen ennek köszönhetően növelhető a szolgáltatás színvonala, javítható az operatív hatékonyság, és könnyebben kezelhető az adminisztráció.

Az éttermeknek számos kihívással kell megküzdeniük a mindennapi tevékenység során. Egyrészt folyamatosan frissíteniük kell étlapjukat, raktározásukat és vállalatirányítási rendszerüket, másrészt naponta több száz rendelést kell kezelniük, melyek között lehetőleg ne forduljon elő hiba, elkerülve az elégedetlen vendégek negatív visszajelzéseit.

A dokumentumban egy olyan rendszert alkottam meg, amely az XML általánosan elfogadott jelölőnyelvének segítségével képes ábrázolni egy ilyen komplex étterem kezelő rendszer működését. Az XML használatával a rendszer strukturáltan képes kezelni az adatokat, könnyebbé téve a különböző programokkal történő integrációt, az adatok elemzését és megosztását.

A dokumentumban szereplő adatstruktúra részben a következőket tartalmazza: rendelések, ételek, készletek és az éttermekhez tartozó munkatársak és tulajdonosok adatait. Minden egyes entitáshoz tartozó egyedi azonosítók (például OrderID, FoodID, RestaurantID) és a velük kapcsolatos konkrét információk (például árak, értékelések, ételek leírása) segítik az étterem személyzetét a gyors és hatékony információkezelésben.

A rendelési folyamat során a rendszert úgy terveztem meg, hogy azt a konyhapersonaltól kezdve a pincéren át a kiszállítóig mindenki könnyen követhesse. Az ételek állapota például a „Megrendelve” státusztól egészen a „Kiszállítva” állapotig követhető, így minden pillanatban tudjuk, hol tart a folyamat.

Az ételek részletesen specifikáltak, ami azt jelenti, hogy minden fogásnak megvan a maga egyedi azonosítója, ára és leírása, ami lehetővé teszi az étlap rugalmas kezelését az aktuális kínálatnak megfelelően. A leírásokban a hagyományos magyar ételek mellett helyet kapnak a modern, kreatív fogások is, reflektálva a piaci igényekre és a változatos étkezési szokásokra.

Minden restaurant entitás tartalmaz egy csokorban azokat a kulcsinformációkat, amik az ügyfelek számára fontosak lehetnek. Ezek az információk tartalmazzák a címet, a várost, és az étterem nevét.

Ezen keresztül nem csak az adatzár előkészítése, hanem a vásárlóknak való megjelenítés is egyszerűsödik.

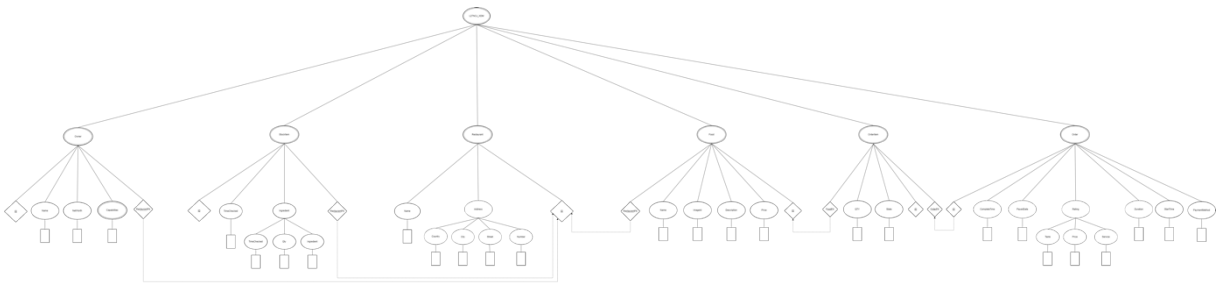
A tulajdonosok adatai pedig betekintést nyújtanak nekünk arra vonatkozóan, hogy milyen háttérrel, tudással és anyagi erőforrással rendelkezik az adott éttermet üzemeltető személy. Az XML dokumentum egyetlen szerkezeti egységben tárja fel ezeket az információkat és ötvözi az üzleti logika megkövetelte követelményeket.

Mindezek alapján az itt látható XML struktúra egy olyan általános és alkalmazható formátum, amely különböző informatikai rendszerekbe való integrálhatóságát biztosítani tudja, így nagyban hozzájárulva az éttermek mindennapi működésének könnyítéséhez, akár egy belső menedzsment rendszer részeként, akár egy külsős szoftver felületén történő megjelenítést célozva.

1b Az adatbázis ER modell tervezése,

A már fent említett adatbázis ER-modelle a következő:

Ennek a konvertálása után a következő XDM modellt kapjuk:



Az adatbázis XDM (XML Document Model) modellre történő konvertálása során az adatokat XML dokumentumoké alakítjuk át. Ez a konvertálás az alábbi szabályok szerint történik:

egyed ⇒ elem
elemi tulajdonság ⇒ szöveg elem
kulcs tulajdonság ⇒ elemjellemző + kulcs megkötés
összetett tulajdonság ⇒ elemeket tartalmazó gyerekelem
többértékű tulajdonság ⇒ gyerekelem, ismétlődéssel
kapcsoló tulajdonság ⇒ elemjellemző + idegen kulcs megkötés
1:N kapcsolat ⇒ elemjellemző + kulcs + idegen kulcs megkötés
N:M kapcsolat⇒ külön kapcsoló elem és idegen kulcsok mindkét oldalra

1c Az XDM modell alapján XML dokumentum készítése,

Az xml dokumentumban példányosítom az egyedek, strukturáltan, megfelelően jelölve a kulcsokat, ahol szükséges pedig az idegenkulcsokat. Ezt a folyamatot megismétlem a többi elemre.

```
<?xml version="1.0" encoding="UTF-8"?>
<L27NCJRestaurants xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XMLSchemaL27ncj.xsd">
  <!-- OrderItem Instances -->
  <OrderItem OrderItemID="1001" OrderID="50012" FoodID="2001">
    <Quantity>2</Quantity>
    <State>Elkészült</State>
  </OrderItem>
  <OrderItem OrderItemID="1002" OrderID="5002" FoodID="2002">
    <Quantity>1</Quantity>
    <State>Folyamatban</State>
  </OrderItem>
  <OrderItem OrderItemID="1003" OrderID="5003" FoodID="2003">
    <Quantity>3</Quantity>
    <State>Kiszállítva</State>
  </OrderItem>
  <OrderItem OrderItemID="1004" OrderID="5004" FoodID="2004">
    <Quantity>4</Quantity>
    <State>Megrendelve</State>
  </OrderItem>
  <!-- Order Instances -->
  <Order OrderID="5001" RestaurantID="3001">
    <TotalPrice>7500.00</TotalPrice>
    <TasteRating>4.5</TasteRating>
    <ServiceRating>4.0</ServiceRating>
    <PaymentDate>2023-04-12T18:30:00</PaymentDate>
    <CompletionTime>2023-04-12T19:00:00</CompletionTime>
    <StartTime>2023-04-12T17:45:00</StartTime>
    <PaymentMethod>Készpénz</PaymentMethod>
  </Order>
  <Order OrderID="5002" RestaurantID="3002">
    <TotalPrice>3200.00</TotalPrice>
    <StartTime>2023-04-13T12:00:00</StartTime>
    <PaymentMethod>Kártya</PaymentMethod>
  </Order>
```

```
  <Order OrderID="5003" RestaurantID="3003">
    <TotalPrice>4500.00</TotalPrice>
    <StartTime>2023-04-14T20:15:00</StartTime>
    <PaymentMethod>Online</PaymentMethod>
  </Order>
  <Order OrderID="5004" RestaurantID="3004">
    <TotalPrice>9800.00</TotalPrice>
    <StartTime>2023-04-15T13:30:00</StartTime>
    <PaymentMethod>Utánvét</PaymentMethod>
  </Order>
<!-- Restaurant Instances --
  <Restaurant RestaurantID="3001">
    <Address>
      <Street>Kossuth Lajos utca</Street>
      <City>Budapest</City>
      <Door>2</Door>
    </Address>
    <Name>Étterem A</Name>
  </Restaurant>
  <Restaurant RestaurantID="3002">
    <Address>
      <Street>Széchenyi tér</Street>
      <City>Debrecen</City>
      <Door>8</Door>
    </Address>
    <Name>Étterem B</Name>
  </Restaurant>
  <Restaurant RestaurantID="3003">
    <Address>
      <Street>Attila út</Street>
      <City>Szeged</City>
      <Door>32</Door>
    </Address>
    <Name>Étterem C</Name>
  </Restaurant>
  <Restaurant OrderItemID="3004" RestaurantID="3004">
    <Address>
      <Street>Vasvári Pál utca</Street>
      <City>Pécs</City>
      <Door>3</Door>
    </Address>
    <Name>Étterem D</Name>
  </Restaurant>
  <!-- Food Instances -->
  <Food FoodID="2001">
    <FoodID>2001</FoodID>
    <Name>Hortobágyi palacsinta</Name>
```

```
    <Price>1200.00</Price>
    <Description>Tradicionalis magyar etel.</Description>
  </Food>
  <Food FoodID="2002">
    <FoodID>2002</FoodID>
    <Name>Gulyásleves</Name>
    <Price>1500.00</Price>
    <Description>Hagyományos magyar leves marhahúsból.</Description>
  </Food>
  <Food FoodID="2003">
    <FoodID>2003</FoodID>
    <Name>Pörkölt</Name>
    <Price>1800.00</Price>
    <Description>Magyar pörkölt sertéshúsból, paprikával és hagymával.</Description>
  </Food>
  <Food FoodID="2004">
    <FoodID>2004</FoodID>
    <Name>Somlói galuska</Name>
    <Price>900.00</Price>
    <Description>Édes magyar desszert.</Description>
  </Food>
  <!-- StockItem Instances -->
  <StockItem StockItemID="6001" RestaurantID="5001">
    <LastChecked>2023-04-11T10:00:00</LastChecked>
    <Quantity>50</Quantity>
  </StockItem>
  <StockItem StockItemID="6002" RestaurantID="5002">
    <LastChecked>2023-04-12T15:30:00</LastChecked>
    <Quantity>30</Quantity>
  </StockItem>
  <StockItem StockItemID="6003" RestaurantID="5001">
    <LastChecked>2023-04-13T09:20:00</LastChecked>
    <Quantity>20</Quantity>
  </StockItem>
  <StockItem StockItemID="6004" RestaurantID="5003">
    <LastChecked>2023-04-14T11:45:00</LastChecked>
    <Quantity>40</Quantity>
  </StockItem>
  <!-- Employee Instances -->
  <Owner OwnerID="7001" RestaurantID="5001">
    <Name>Nagy István</Name>
    <NetWorth>5000</NetWorth>
    <CapabilityName>Vendéglátó ipari diploma</CapabilityName>
  </Owner>
  <Owner OwnerID="7002" RestaurantID="5002">
    <Name>Kovács Éva</Name>
    <NetWorth>8000</NetWorth>
```

```
    <CapabilityName>Befektető</CapabilityName>
  </Owner>
  <Owner OwnerID="7003" RestaurantID="5003">
    <Name>Szabó Gábor</Name>
    <NetWorth>20000</NetWorth>
    <CapabilityName>Tőzsde</CapabilityName>
    <CapabilityName>Kereskedelem</CapabilityName>
  </Owner>
  <Owner OwnerID="7004" RestaurantID="5004">
    <Name>Tóth Anna</Name>
    <NetWorth>3000</NetWorth>
    <CapabilityName>Szakács</CapabilityName>
  </Owner>
</L27NCJRestaurants>
```

1d Az XML dokumentum alapján XMLSchema készítése - saját típusok, ref, key, keyref, speciális elemek,

Ezután létrehozom a xml-ben megadott típusokat illetve kulcsokat meghatározó sémát, kigyűjtöm az egyszerű típusokat, elementeket illetve ezek megszorításait beállítom. Ezt követően meghatározom a saját, komplex típusaimat, ezekre is alkalmazom a megszorításaimat, itt már felhasználva az egyszerű típusokat. Ezt követő lépésként a gyökérelemtől indulva felépítem az XML struktúrát, beállítom az elsődleges kulcsokat, valamint az elsődleges kulcsokra az idegenkulcsokat, illetve az 1:1 kapcsolat megvalósításához használom a Unique kulcsszót is.

```
<?xml version="1.0" encoding="UTF-8"?>

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

  <!-- Simple Types -->

  <xs:simpleType name="nonNegativeInteger">

    <xs:restriction base="xs:integer">

      <xs:minInclusive value="0" />

    </xs:restriction>

  </xs:simpleType>

  <xs:simpleType name="dateTimeFormat">

    <xs:restriction base="xs:dateTime" />

  </xs:simpleType>

  <xs:simpleType name="currencyFormat">

    <xs:restriction base="xs:decimal">

      <xs:fractionDigits value="2" />

    </xs:restriction>
```

```
</xs:simpleType>
<xs:simpleType name="ratingType">
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="5" />
    <xs:fractionDigits value="1" />
  </xs:restriction>
</xs:simpleType>
<!-- Primitive Types -->
<xs:element name="OrderItemID" type="xs:positiveInteger" />
<xs:element name="Quantity" type="xs:positiveInteger" />
<xs:element name="State" type="xs:string" />
<xs:element name="OrderID" type="xs:positiveInteger" />
<xs:element name="FoodID" type="xs:positiveInteger" />
<xs:element name="TotalPrice" type="currencyFormat" />
<xs:element name="TasteRating" type="ratingType" />
<xs:element name="ServiceRating" type="ratingType" />
<xs:element name="PaymentDate" type="dateTimeFormat" />
<xs:element name="CompletionTime" type="dateTimeFormat" />
<xs:element name="StartTime" type="dateTimeFormat" />
<xs:element name="PaymentMethod" type="xs:string" />
<xs:element name="RestaurantID" type="xs:positiveInteger" />
<xs:element name="Name" type="xs:string" />
<xs:element name="Street" type="xs:string" />
<xs:element name="City" type="xs:string" />
<xs:element name="Door" type="xs:positiveInteger" />
<xs:element name="Country" type="xs:string" />
<xs:element name="Price" type="currencyFormat" />
<xs:element name="ImageID" type="xs:positiveInteger" />
<xs:element name="Description" type="xs:string" />
<xs:element name="IngredientID" type="xs:positiveInteger" />
<xs:element name="NetWorth" type="currencyFormat" />
```

```
<xs:element name="MeasurementUnit" type="xs:string" />
<xs:element name="CapabilityName" type="xs:string" />
<xs:element name="EmployeeID" type="xs:positiveInteger" />
<xs:element name="StockItemID" type="xs:positiveInteger" />
<xs:element name="LastChecked" type="dateTimeFormat" />
<xs:element name="IdCardNumber" type="xs:string" />
<!-- Complex Types with References -->
<xs:complexType name="OrderItemType">
  <xs:sequence>
    <xs:element ref="Quantity" />
    <xs:element ref="State" />
  </xs:sequence>
  <xs:attribute name="OrderItemID" type="xs:positiveInteger" />
  <xs:attribute name="OrderID" type="xs:positiveInteger" />
  <xs:attribute name="FoodID" type="xs:positiveInteger" />
</xs:complexType>
<xs:complexType name="OrderType">
  <xs:sequence>
    <xs:element ref="TotalPrice" />
    <xs:element ref="TasteRating" minOccurs="0" />
    <xs:element ref="ServiceRating" minOccurs="0" />
    <xs:element ref="PaymentDate" minOccurs="0" />
    <xs:element ref="CompletionTime" minOccurs="0" />
    <xs:element ref="StartTime" />
    <xs:element ref="PaymentMethod" />
  </xs:sequence>
  <xs:attribute name="OrderID" type="xs:positiveInteger" />
  <xs:attribute name="RestaurantID" type="xs:positiveInteger" />
</xs:complexType>
<xs:complexType name="RestaurantType">
  <xs:sequence>
    <xs:element name="Address">
```

```
            <xs:complexType>
              <xs:sequence>
                <xs:element ref="Street" />
                <xs:element ref="City" />
                <xs:element ref="Door" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element ref="Name" />
        </xs:sequence>
        <xs:attribute name="RestaurantID" type="xs:positiveInteger" />
        <xs:attribute name="OrderItemID" type="xs:positiveInteger" />
      </xs:complexType>
      <xs:complexType name="FoodType">
        <xs:sequence>
          <xs:element ref="FoodID" />
          <xs:element ref="Name" />
          <xs:element ref="Price" />
          <xs:element ref="ImageID" minOccurs="0" />
          <xs:element ref="Description" />
        </xs:sequence>
        <xs:attribute name="FoodID" type="xs:positiveInteger" />
        <xs:attribute name="RestaurantID" type="xs:positiveInteger" />
      </xs:complexType>
      <xs:complexType name="OwnerType">
        <xs:sequence>
          <xs:element ref="Name" />
          <xs:element ref="NetWorth" />
          <xs:element ref="CapabilityName" minOccurs="0" maxOccurs="unbounded"
/>
        </xs:sequence>
        <xs:attribute name="OwnerID" type="xs:positiveInteger" />
```

```
    <xs:attribute name="RestaurantID" type="xs:positiveInteger" />
  </xs:complexType>
  <xs:complexType name="StockItemType">
    <xs:sequence>
      <xs:element ref="LastChecked" />
      <xs:element ref="Quantity" />
    </xs:sequence>
    <xs:attribute name="StockItemID" type="xs:positiveInteger" />
    <xs:attribute name="RestaurantID" type="xs:positiveInteger" />
  </xs:complexType>
  <!-- Root Element -->
  <xs:element name="L27NCJRestaurants">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="OrderItem" type="OrderItemType" minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="Order" type="OrderType" minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="Restaurant" type="RestaurantType" minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="Food" type="FoodType" minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="StockItem" type="StockItemType" minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="Owner" type="OwnerType" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
    <!-- Primary Keys -->
    <xs:key name="OrderItemKey">
      <xs:selector xpath="OrderItem" />
      <xs:field xpath="@OrderItemID" />
    </xs:key>
    <xs:key name="OrderKey">
```

```
    <xs:selector xpath="Order" />
    <xs:field xpath="@OrderID" />
  </xs:key>
  <xs:key name="RestaurantKey">
    <xs:selector xpath="Restaurant" />
    <xs:field xpath="@RestaurantID" />
  </xs:key>
  <xs:key name="FoodKey">
    <xs:selector xpath="Food" />
    <xs:field xpath="@FoodID" />
  </xs:key>
  <xs:key name="IngredientKey">
    <xs:selector xpath="Ingredient" />
    <xs:field xpath="@IngredientID" />
  </xs:key>
  <xs:key name="StockItemKey">
    <xs:selector xpath="StockItem" />
    <xs:field xpath="@StockItemID" />
  </xs:key>
  <xs:key name="EmployeeKey">
    <xs:selector xpath="Employee" />
    <xs:field xpath="@EmployeeID" />
  </xs:key>
  <!-- keyref, unique -->
  <xs:keyref name="OrderItem_OrderID" refer="OrderKey">
    <xs:selector xpath="OrderItem" />
    <xs:field xpath="@Order" />
  </xs:keyref>
  <xs:keyref name="Order_FoodID" refer="FoodKey">
    <xs:selector xpath="OrderItem" />
    <xs:field xpath="@Order" />
  </xs:keyref>
```

```
        <xs:keyref name="Order_RestaurantID" refer="RestaurantKey">
          <xs:selector xpath="Order" />
          <xs:field xpath="@Restaurant" />
        </xs:keyref>
        <xs:keyref name="Restaurant_FoodID" refer="RestaurantKey">
          <xs:selector xpath="Food" />
          <xs:field xpath="@Restaurant" />
        </xs:keyref>
        <xs:keyref name="StockItem_RestaurantID" refer="RestaurantKey">
          <xs:selector xpath="StockItem" />
          <xs:field xpath="@Restaurant" />
        </xs:keyref>
        <xs:keyref name="Owner_RestaurantID" refer="RestaurantKey">
          <xs:selector xpath="Owner" />
          <xs:field xpath="@Restaurant" />
        </xs:keyref>
        <!-- Future unique -->
        <xs:unique name="Owner_RestaurantID_one_to_one">
          <xs:selector xpath="Owner" />
          <xs:field xpath="@Restaurant" />
        </xs:unique>
      </xs:element>
    </xs:schema>
```

2.feladat

2a adatolvasás,

A szokásos 3 könyvtárból történtő import (IO,xml,w3c) után beolvasom a fájlt egy try catch (ez az I/O művelet miatt szükséges) példányosítom a DocumentBuilderFactory a normalizálását követően megnyitom az outpit filet, majd meghívom a printWriter nevű függvényt ami egyszerre írja konzolra és fájlba a bemeneti xml dokumentum tartalmát. A dokumentum főbb elemeit nodeListekben tárolom el, ezeken for ciklussal megyek végig, vizsgálom a gyerekelemeket, ezeknek a tartalmát (contexét).

```
package hu.domparse.l27ncj;
```

```
import java.io.File;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.List;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class DomReadL27ncj {

	public static void main(String[] args) {
		Document root = Read(pathToXml);
		PrintNode(root.getDocumentElement(), 0, System.out);
	}

	public static String pathToXml = "XMLL27ncj.xml";
	public int indenting = 0;

  public static Document Read(String fromPath) {

    String filePath = pathToXml;
    System.out.println("Reading: " + pathToXml);

    File xmlFile = new File(filePath);
```

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

        // now XML is loaded as Document in memory, lets convert it to Object List
        System.out.println();
        return doc;

    } catch (SAXException | ParserConfigurationException ex1) {
        ex1.printStackTrace();

    }catch ( IOException e1 ) {
          System.out.println("Error: " + e1.getMessage());
          System.out.println("Error: " + e1.getLocalizedMessage());
    }
    return null;
}

/**
 * Prints the text for a node, recursively prints the inner node, and so on
 * @param node
 * @param indenting
 */
public static void PrintNode(Node node, int indenting, PrintStream stream) {


        String attrContentStr = extractAttributesStr(node);

        printStartTag(node.getNodeName(), attrContentStr, indenting, stream);
```

```
    NodeList children = node.getChildNodes();


int length = children.getLength();

if (length > 1) {
   for (int i = 0; i < length; i++) {

    Node child = children.item(i);
    var subChildren = child.getChildNodes();

    var nodeName = child.getNodeName();
    if (nodeName == "#text") {
            //stream.print(child.getTextContent());
            //continue;
    }
    if (child.getNodeName() == "#comment") {
            printComment(child.getTextContent(), stream);
            continue;
    }

    if (subChildren.getLength() > 1) {
            PrintNode(child, indenting + 1, stream);

    }else if (child.getTextContent().length() > 0) {
            // Print start and end node
            if (nodeName != "#text") {
                    stream.print("<" + nodeName + ">");
            }

            stream.print(child.getTextContent());

            if (nodeName != "#text") {
```

```
                        stream.print("</" + nodeName + ">");
                    }
        } else {
                    PrintNode(child, 0, stream);
        }
       }
        }else {

                    printLine(node.getTextContent(), "", "", indenting + 2, stream);
        }
        printEndTag(node.getNodeName(), indenting, System.out);
}

/**
 * Prints a node into a text stream
 * @param node
 * @param indenting
 */
public static void PrintNodeOld(Node node, int indenting, PrintStream stream) {


        String attrContentStr = extractAttributesStr(node);

        printStartTag(node.getNodeName(), attrContentStr, indenting, stream);

        NodeList children = node.getChildNodes();


   int length = children.getLength();

        if (length > 1) {
       for (int i = 0; i < length; i++) {
```

```
        Node child = children.item(i);
        var subChildren = child.getChildNodes();
        if (child.getNodeName() == "#text") {
                stream.print(child.getFirstChild().getTextContent());
                continue;
        }
        if (child.getNodeName() == "#comment") {
                printComment(child.getTextContent(), stream);
                continue;
        }

        if (subChildren.getLength() > 1) {
                PrintNode(child, indenting + 1, stream);
        }
       }
        }else {
                //printLine(node.getTextContent(), "", "", indenting + 2, stream);
        }
        printEndTag(node.getNodeName(), indenting, stream);


}

        private static String extractAttributesStr(Node node) {
                String attrContentStr = "";
                NamedNodeMap namedNodeMap = node.getAttributes();

        if (namedNodeMap != null) {
                for (int i = 0; i < namedNodeMap.getLength(); i++) {
                        if (i > 0) {
                                attrContentStr += " ";
                        }
                        String name = namedNodeMap.item(i).getNodeName();
```

```
                String value = namedNodeMap.item(i).getNodeValue();
                attrContentStr += name + "=\"" + value + "\"";
            }
            if (namedNodeMap.getLength() > 0) {
                attrContentStr = " " + attrContentStr + "";
            }
        }
            return attrContentStr;
        }

    /**
     * Gets a tag internal text value
     * @param tag
     * @param element
     * @return A string representing the tag value
     */
    private static String getTagValue(String tag, Element element) {
        NodeList nodeList = element.getElementsByTagName(tag).item(0).getChildNodes();
        Node node = (Node) nodeList.item(0);
        return node.getNodeValue();
    }

    /**
     * PRints a whole line used for the other type of printing out structured data
     * @param element
     * @param suffix used to set start/stop
     * @param attributes Attributes, already formated in correct text format
     * @param indent the indentiation to print
     */
    private static void printLine(String element, String suffix, String attributes, int indent, PrintStream
stream) {
            printIndent(indent, stream);
            stream.print(element + attributes + suffix + "\n");
```

```
        }

    /**
     * Prints a start tag
     * @param element
     * @param attributesStr
     * @param indent
     */
    private static void printStartTag(String element, String attributesStr, int indent, PrintStream stream)
{
        printIndent(indent, stream);
        stream.print("<" + element + attributesStr + ">\n");
    }

    /**
     * Prints an end tag
     * @param element
     * @param indent
     */
    private static void printEndTag(String element, int indent, PrintStream stream) {
        printIndent(indent, stream);
        stream.print("</" + element + ">\n");
    }

    /**
     * Prints out a comment
     * @param text
     */
    private static void printComment(String text, PrintStream stream) {
        stream.print("<!--" + text + "-->\n");
    }

        /**
```

```
     * Prints the indentation
     * @param indent
     */
    private static void printIndent(int indent, PrintStream stream) {
        for (int i = 0; i < indent; i++) {
            stream.print("  ");
        }
    }

}
```

2b adatmódosítás,

A szokásos 3 könyvtárból történtő import (IO,xml,w3c) után beolvasom a fájlt egy try catch (ez az I/O művelet miatt szükséges) példányosítom a DocumentBuilderFactory a normalizálását követően, a dokumentum főbb elemeit nodeListekben tárolom el. Az elemek módosítását úgy végzem el hogy hogy lekérem egy element tartalmát a `getElementsByTagName` dom függvénnyel, ezután a SetContext metódussal módosítom a tartalmát, ugyan ezt, a feladat kiírásnak megfelelően, elvégzem minden esetben

```
package hu.domparse.l27ncj;


import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;


public class DomModifyL27ncj {


    public static void main(String[] args) {



        Document doc = DomReadL27ncj.Read("XMLL27ncj.xml");


        // Add new nodes
  Element newEmployee = doc.createElement("Employee");
  newEmployee.setAttribute("EmployeeID", "7005");
```

```
        Element newName = doc.createElement("Name");

        newName.setTextContent("Fekete Péter");
        Element newIdCardNumber = doc.createElement("IdCardNumber");
        newIdCardNumber.setTextContent("ID13579246");
        newEmployee.appendChild(newName);
        newEmployee.appendChild(newIdCardNumber);
        doc.getDocumentElement().appendChild(newEmployee);

        // Remove nodes
        NodeList employees = doc.getElementsByTagName("Employee");
        if (employees.getLength() > 0) {
            Node firstEmployee = employees.item(0);
            doc.getDocumentElement().removeChild(firstEmployee);
        }

        // Modify nodes
        NodeList foods = doc.getElementsByTagName("Food");
        for (int i = 0; i < foods.getLength(); i++) {
            Node food = foods.item(i);
            if (food.getNodeType() == Node.ELEMENT_NODE) {
                Element foodElement = (Element) food;
                if
("Gulyásleves".equals(foodElement.getElementsByTagName("Name").item(0).getTextContent())) {
                    foodElement.getElementsByTagName("Price").item(0).setTextContent("1600.00");
                }
            }
        }

        DomReadL27ncj.PrintNode(doc.getDocumentElement(), 0, System.out);



         }
```

}

2c adatlekérdezés,

A szokásos 3 könyvtárból történtő import (IO,xml,w3c) után beolvasom a fájlt egy try catch (ez az I/O művelet miatt szükséges) példányosítom a DocumentBuilderFactory a normalizálását követően, a dokumentum főbb elemeit nodeListekben tárolom el. A lekérdezés során egy nodelistbe lekérem az adott fő elem gyerekeit (child/descandant) majd egy for ciklussal végig iterálok a listán, majd ahol a kért adat egyezését megtalálom ,egy stringbe letárolom a kért attribútumot/tartalmat (contexet), kereszt táblás (kereszt elemes) lekérdezés esetén dupla for ciklust használok.

```
package hu.domparse.l27ncj;

import org.w3c.dom.DOMException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import java.lang.reflect.Array;
import java.math.BigDecimal;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

public class DomQueryL27ncj {

    public static void main String[] args) {
        Document doc = DomReadL27ncj.Read("XMLL27ncj.xml");

        // Lekérdezés: Legolcsóbb étel
        Element cheapestFood = findCheapestFood(doc);

        // Legérdezés, étterem a legjobb ízzel
        Element highestRatedRestaurant = findHighestRatedOrder(doc);


        Element findUnpaidOrders = findUnpaidOrders(doc);

        // Lekérdezés, Legutóbbi befizetés
        Element latestPaymentDate = findLatestPaymentDate(doc);

        // Lekérdezés: Leghosszabb nevű alkalmazott
        Element employeeWithLongestName = findOwnerWithLongestName(doc);
    }

    private static Element findCheapestFood(Document doc) {
        NodeList foods = doc.getElementsByTagName("Food");
        Element cheapestFood = null;
        BigDecimal lowestPrice = new BigDecimal(Integer.MAX_VALUE);
        for (int i = 0; i < foods.getLength(); i++) {
            Node food = foods.item(i);
            if (food.getNodeType() == Node.ELEMENT_NODE) {
                Element foodElement = (Element) food;
                BigDecimal price = new BigDecimal(foodElement.getEle-
mentsByTagName("Price").item(0).getTextContent());
                if (price.compareTo(lowestPrice) < 0) {
                    lowestPrice = price;
                    cheapestFood = foodElement;
                }
            }
        }
        if (cheapestFood != null) {
```

```
            System.out.println(formatElement((Element) cheapestFood));
        }
        return cheapestFood;
    }

    private static Element findHighestRatedOrder(Document doc) {
        NodeList orders = doc.getElementsByTagName("Order");
        Element highestRatedOrder = null;
        BigDecimal highestRating = BigDecimal.ZERO;
        for (int i = 0; i < orders.getLength(); i++) {
            Node order = orders.item(i);
            if (order.getNodeType() == Node.ELEMENT_NODE) {
                Element orderElement = (Element) order;
                NodeList tasteRatings = orderElement.getElementsByTagName("Tast-
eRating");
                if (tasteRatings.getLength() > 0) {
                    BigDecimal rating = new BigDecimal(tasteRat-
ings.item(0).getTextContent());
                    if (rating.compareTo(highestRating) > 0) {
                        highestRating = rating;
                        highestRatedOrder = orderElement;
                    }
                }
            }
        }
        if (highestRatedOrder != null) {
            System.out.println(formatElement((Element) highestRatedOrder));
        }
        return highestRatedOrder;
    }

    private static Element findUnpaidOrders(Document doc) {
        NodeList orders = doc.getElementsByTagName("Order");
        Element mostExpensiveIngredient = null;

        ArrayList<Element> orderElements = new ArrayList<Element>();


        BigDecimal highestPrice = BigDecimal.ZERO;
        for (int i = 0; i < orders.getLength(); i++) {
            Node order = orders.item(i);
            Element orderElement = (Element) order;
            boolean hasPayment = orderElement.getElementsByTagName "Payment-
Date".getLength() > 0;
            if (!hasPayment) {
             orderElements.add orderElement;
            }
        }
        for (int i = 0; i < orderElements.size(); i++) {
            System.out.println(formatElement((Element) orderElements.get(i)));
        }
        return mostExpensiveIngredient;
    }

    private static Element findLatestPaymentDate(Document doc) {
        NodeList orders = doc.getElementsByTagName("Order");
        Element latestOrder = null;
        Date latestDate = null;
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");
        for (int i = 0; i < orders.getLength(); i++) {


             if (latestDate == null) {
                    latestOrder = (Element) orders.item(i);
```

```
                NodeList paymentDates = latestOrder.getElementsByTagName("Payment-
Date");
                    try {
                                    latestDate = sdf.parse(payment-
Dates.item(0).getTextContent());
                                } catch (DOMException e) {
                                    e.printStackTrace();
                                } catch (ParseException e) {
                                    e.printStackTrace();
                                }
                    break;
                }
            Node order = orders.item(i);
            if (order.getNodeType() == Node.ELEMENT_NODE) {
                Element orderElement = (Element) order;
                NodeList paymentDates = orderElement.getElementsByTagName("Payment-
Date");
                if (paymentDates.getLength() > 0) {
                    try {
                        Date paymentDate = sdf.parse(payment-
Dates.item(0).getTextContent());
                        if (latestDate == null || paymentDate.after(latestDate)) {
                            latestDate = paymentDate;
                            latestOrder = orderElement;
                        }
                    } catch (ParseException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
        if (latestOrder != null && latestDate != null) {
            System.out.println(formatElement((Element) latestOrder));
        }
        return latestOrder;
    }

    private static Element findOwnerWithLongestName(Document doc) {
        NodeList employees = doc.getElementsByTagName("Owner");
        Element employeeWithLongestName = null;
        int maxLength = 0;
        for (int i = 0; i < employees.getLength(); i++) {
            Node employee = employees.item(i);
            if (employee.getNodeType() == Node.ELEMENT_NODE) {
                Element employeeElement = (Element) employee;
                String name = employeeElement.getEle-
mentsByTagName("Name").item(0).getTextContent();
                if (name.length() > maxLength) {
                    maxLength = name.length();
                    employeeWithLongestName = employeeElement;
                }
            }
        }
        if (employeeWithLongestName != null) {
            System.out.println(formatElement((Element) employeeWithLongestName));
        }
        return employeeWithLongestName;
    }

    private static String formatElement(Element element) {
        StringBuilder result = new StringBuilder();
        result.append("<").append(element.getTagName());

        // Add attributes if any
        if (element.hasAttributes()) {
            for (int i = 0; i < element.getAttributes().getLength(); i++) {
```

```
                result.append(" ")
                        .append(element.getAttributes().item(i).getNodeName())
                        .append("=\"")
                        .append(element.getAttributes().item(i).getNodeValue())
                        .append("\"");
            }
        }

        result.append(">");

        // Add child elements and text content
        NodeList childNodes = element.getChildNodes();
        for (int i = 0; i < childNodes.getLength(); i++) {
            Node child = childNodes.item(i);
            if (child.getNodeType() == Node.ELEMENT_NODE) {
                result.append(formatElement((Element) child));
            } else if (child.getNodeType() == Node.TEXT_NODE) {
                result.append(child.getTextContent());
            }
        }

        result.append("</").append(element.getTagName()).append(">");
        return result.toString();
    }
}
```

2d adatírás,

A szokásos 3 könyvtárból történtő import (IO,xml,w3c) után beolvasom a fájlt egy try catch (ez az I/O művelet miatt szükséges) példányosítom a DocumentBuilderFactory a normalizálását követően, a dokumentum főbb elemeit nodeListekben tárolom el. A dokumentum faszerkezetének felépítését úgy végzem el hogy létrehozom a gyökér elemet, majd ehhez adom hozzá a később létrehozott főelemeket. A kiíratás konzolra és fájlba az 2a,read feladathoz hasonló módon történik.

```
package hu.domparse.l27ncj;


import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;


import java.io.FileWriter;

import java.io.PrintWriter;
```

```
import java.io.StringWriter;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class DomWriteL27ncj {
   public static void main(String[] args) {
      // Create a new Document
      Document doc = createNewDocument();

      // Create root element
      Element rootElement = doc.createElement("L27NCJRestaurants");
      rootElement.setAttribute("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance");
      rootElement.setAttribute("xsi:noNamespaceSchemaLocation", "XMLSchemaL27ncj.xsd");
      doc.appendChild(rootElement);

      // Add OrderItem elements
      addOrderItem(doc, rootElement, "1001", "5001", "2001", "2", "Elkészült");
      addOrderItem(doc, rootElement, "1002", "5002", "2002", "1", "Folyamatban");
      addOrderItem(doc, rootElement, "1003", "5003", "2003", "3", "Kiszállítva");
      addOrderItem(doc, rootElement, "1004", "5004", "2004", "4", "Megrendelve");

      // Add Order elements
      addOrder(doc, rootElement, "5001", "3001", "7500.00", "4.5", "4.0",
            "2023-04-12T18:30:00", "2023-04-12T19:00:00", "2023-04-12T17:45:00", "Készpénz");
      addOrder(doc, rootElement, "5002", "3002", "3200.00",  "4.5", "4.0",
            "2023-04-13T12:00:00", "2023-04-12T19:00:00", "2023-04-12T17:45:00", "Kártya");
      addOrder(doc, rootElement, "5003", "3003", "4500.00",  "4.5", "4.0",
            "2023-04-14T20:15:00", "2023-04-12T19:00:00", "2023-04-12T17:45:00", "Online");
      addOrder(doc, rootElement, "5004", "3004", "9800.00",  "4.5", "4.0",
            "2023-04-15T13:30:00", "2023-04-12T19:00:00", "2023-04-12T17:45:00",  "Utánvét");

      // Add Restaurant elements
```

```
        addRestaurant(doc, rootElement, "3001", "Kossuth Lajos utca", "Budapest", "2", "Étterem A");
        addRestaurant(doc, rootElement, "3002", "Széchenyi tér", "Debrecen", "8", "Étterem B");

        // Output the XML content
        System.out.println(XMLUtils.convertDocumentToString(doc));



        // Write the xml stirng into a file
        String xmlString = XMLUtils.convertDocumentToString(doc);

        try {
            String filename = "XMLL27ncj2.xml";
            PrintWriter writer = new PrintWriter(new FileWriter(filename), true);
            writer.println(xmlString);
            writer.close();
            System.out.println("File write is completed. Note that generated files are not added
automatically to eclipse!");

        }
        catch (Exception e) {
            System.out.println("An error has happend during file write!");
            e.printStackTrace();
        }



    }

    private static Document createNewDocument() {
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            return builder.newDocument();
        } catch (ParserConfigurationException e) {
```

```
            e.printStackTrace();
            return null;
        }
    }

    private static void addOrderItem(Document doc, Element root, String orderItemID, String orderID,
String foodID, String quantity, String state) {
        Element orderItem = doc.createElement("OrderItem");
        orderItem.setAttribute("OrderItemID", orderItemID);
        orderItem.setAttribute("OrderID", orderID);
        orderItem.setAttribute("FoodID", foodID);

        Element quantityElement = doc.createElement("Quantity");
        quantityElement.setTextContent(quantity);
        orderItem.appendChild(quantityElement);

        Element stateElement = doc.createElement("State");
        stateElement.setTextContent(state);
        orderItem.appendChild(stateElement);

        root.appendChild(orderItem);
    }

    private static void addOrder(Document doc, Element root, String orderID, String restaurantID,
String totalPrice,
                                 String tasteRating, String serviceRating, String paymentDate, String
completionTime,
                                 String startTime, String paymentMethod) {
        Element order = doc.createElement("Order");
        order.setAttribute("OrderID", orderID);
        order.setAttribute("RestaurantID", restaurantID);

        addChildElement(doc, order, "TotalPrice", totalPrice);
        addChildElement(doc, order, "TasteRating", tasteRating);
```

```
        addChildElement(doc, order, "ServiceRating", serviceRating);
        addChildElement(doc, order, "PaymentDate", paymentDate);
        addChildElement(doc, order, "CompletionTime", completionTime);
        addChildElement(doc, order, "StartTime", startTime);
        addChildElement(doc, order, "PaymentMethod", paymentMethod);

        root.appendChild(order);
    }

    private static void addRestaurant(Document doc, Element root, String restaurantID, String street,
String city, String door, String name) {
        Element restaurant = doc.createElement("Restaurant");
        restaurant.setAttribute("RestaurantID", restaurantID);

        Element address = doc.createElement("Address");
        addChildElement(doc, address, "Street", street);
        addChildElement(doc, address, "City", city);
        addChildElement(doc, address, "Door", door);

        restaurant.appendChild(address);

        addChildElement(doc, restaurant, "Name", name);

        root.appendChild(restaurant);
    }

    private static void addChildElement(Document doc, Element parent, String tagName, String
textContent) {
        Element child = doc.createElement(tagName);
        if (textContent != null) {
            child.setTextContent(textContent);
        }
        parent.appendChild(child);
```

```
    }
}

class XMLUtils {
    public static String convertDocumentToString(Document doc) {
        try {
            TransformerFactory tf = TransformerFactory.newInstance();
            Transformer transformer = tf.newTransformer();
            transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
            transformer.setOutputProperty(OutputKeys.METHOD, "xml");
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");
            transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");

            StringWriter writer = new StringWriter();
            transformer.transform(new DOMSource(doc), new StreamResult(writer));
            return writer.getBuffer().toString();
        } catch (TransformerException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```