JEGYZŐKÖNYV

Adatkezelés XML környezetben

Féléves feladat

Telekommunikációs cég

Készítette: **Érsek Norbert**
Neptunkód: **IIJU0Z**
Dátum: **2023.12.11.**

Table of Contents

A feladat leírása:

A féléves feladatomban egy telekommunikációs cég lebutított adatbázisát modellezem le. Többek között a feladat különböző internet csomagokat tartalmaz, illetve egy pár felhasználó előfizetését nyilvántartja.

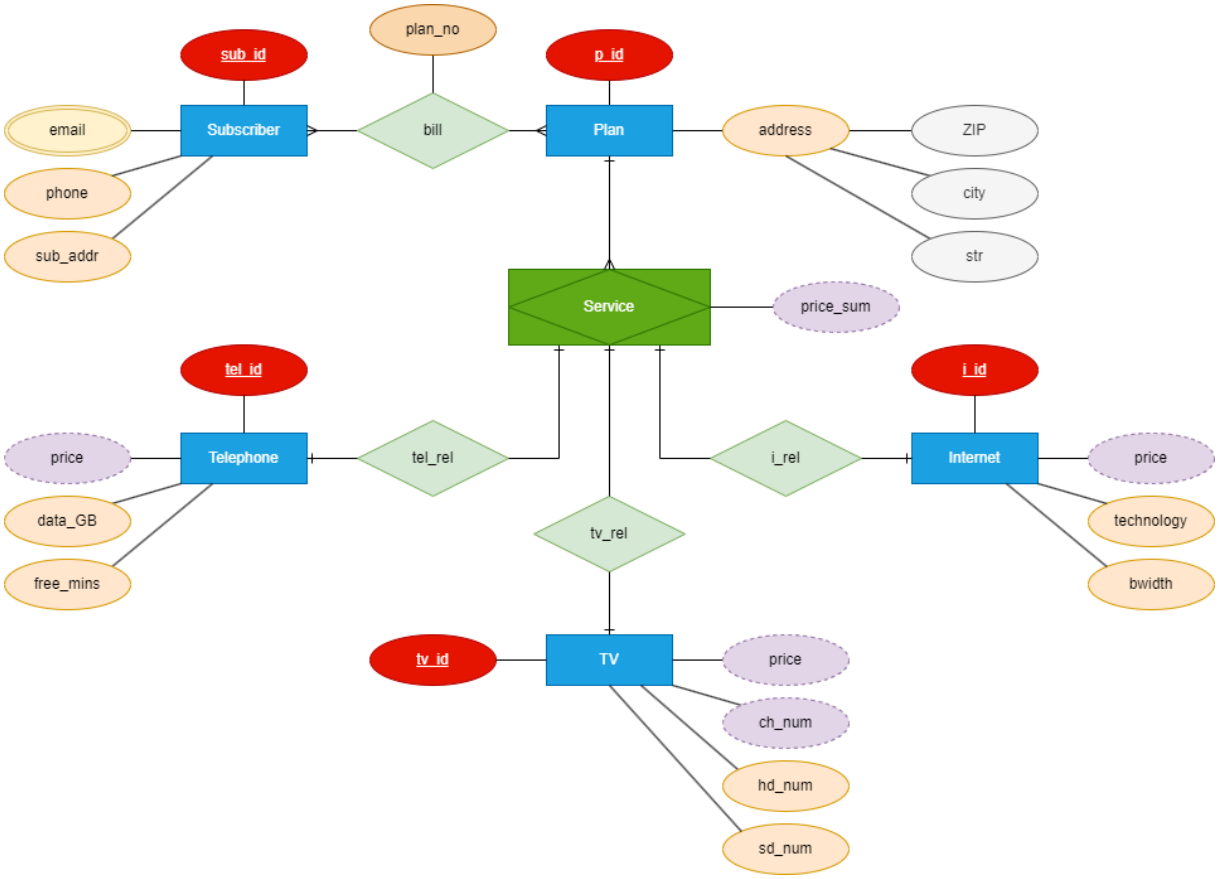
Az egyedek tulajdonságai:

1. Subscriber: előfizető
    a. sub_id: Egy-egy előfizető elsődleges kulcsa
    b. email: Előfizető email címe(i), többértékű.
    c. phone: telefonszám
    d. address: cím
        i. ZIP: irányítószám
        ii. city: város neve
        iii. str: utca neve, ill házszám
2. Bill: az előfizetés – előfizetőt összeköttető kapcsolótáblából jött létre
    a. bill_sub: idegen kulcs az előfizető felé
    b. bill_plan: idegen kulcs az előfizetés felé
    c. plan_no: hányadik előfizetése az adott előfizetőnek
3. Plan: előfizetés
    a. p_id: előfizetés elsődleges kulcsa
    b. address: cím
        i. ZIP: irányítószám
        ii. city: város neve
        iii. str: utca neve, házszám
4. Service: Előfizetést kapcsolja össze a különböző szolgáltatásokat
    a. serv_sum: idegen kulcs, az előfizetésre mutat
    b. tel_serv: idegen kulcs, telefonra mutat
    c. i_serv: idegen kulcs, internetre mutat
    d. tv_serv: idegen kulcs, tv_re mutat
    e. price_sum: összesen fizetendő
5. Internet: internet csomagok listája
    a. i_id: elsődleges kulcs
    b. technology: megmutatja milyen technológiával oldották meg az internetelérést (pl: csavart érpár, optikai kábel)
    c. bwidth: sávszélesség
    d. price: csomag ára
6. Telephone: telefon csomagok listája
    a. tel_id: elsődleges kulcs
    b. data_GB: felhasználható adatmennyiség Gigabyte-ban.
    c. free_mins: ingyenes percek száma
    d. price: csomag ára
7. TV: TV csomagok listája
    a. tv_id: elődleges kulcs
    b. ch_num: csatornák száma összesítve
    c. sd_num: sd csatornák száma
    d. hd_num: hd csatornák száma
    e. price: csomag ára

1. feladat

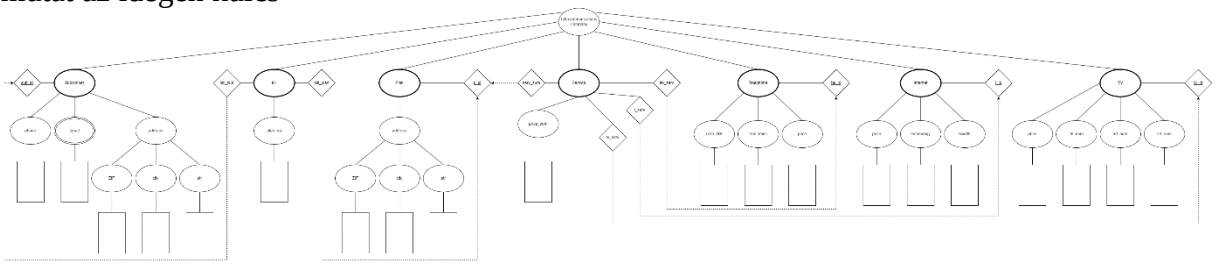
1a) Az adatbázis ER modellje:

- Subscriber – Plan között N:M reláció, mert egy előfizetőnek több előfizetése is lehet.
- Plan – Service között 1:N kapcsolat. Egy címre bejelentett előfizetésben több csomag is lehet ugyanabból a szolgáltatásból.
- Service – Telephone között 1:1 kapcsolat, mert egy szolgáltatás-csomagban csak egy szolgáltatás lehetséges egy fajtából.
- Service – TV között 1:1 kapcsolat, mert egy szolgáltatás-csomagban csak egy szolgáltatás lehetséges egy fajtából.
- Service – Internet között 1:1 kapcsolat, mert egy szolgáltatás-csomagban csak egy szolgáltatás lehetséges egy fajtából.



1b) Az adatbázis konvertálása XDM modellre:

XDM-re konvertáláskor három fontosabb jelölésről beszélhetünk: ellipszis, rombusz és téglalap. Ellipszisek lesznek az egyedi entitásokból, illetve tulajdonságokból. Onnan lehet ezeket megkülönböztetni, hogy az elem, ami az egyedekből jön létre vagy vastag körvonallal jelölt, vagy duplavonallal, emberfüggően. Rombuszok az attribútumokat jelölik, melyek a kulcsok, illetve más fontos elemek. Elsődleges kulcsoknak a nevét alá szokás húzni. A téglalapok jelölik a tényleges adatokat, szöveget, számokat. Többértékű elemet dupla ellipszis mutat. Idegen kulcsok kapcsolatát szaggatot vonallal jelöljük, ahol a nyíl mutatja meg, mire mutat az idegen kulcs



1c) Az XDM modell alapján XML dokumentum készítése:

Az XML dokumentum az XDM modell alapján került kidolgozásra. A gyökérelem a „telecommunications” megnevezést kapta a feladat leírása alapján.

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

<telecommunications xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XSDiiju0z.xsd">


    <!-- Subscribers -->
    <!-- begin -->
    <subscriber sub_id="0001">
        <email>piroskaneni@piroskaneni.hu</email>
        <email>piroskaburner@gmail.com</email>
        <email>piros@freemail.hu</email>
        <phone>+36466748449</phone>
        <address>
            <ZIP>3467</ZIP>
            <city>Ároktő</city>
            <str>Kossuth Lajos út 3</str>
        </address>
    </subscriber>
    <subscriber sub_id="0002">

        <email>istvan@gmail.com</email>
        <email>akiraly@citromail.hu</email>
        <email>istvankiraly@allamalapitas.hu</email>
        <phone>+36204567892</phone>
        <address>
            <ZIP>2435</ZIP>
            <city>Nagylók</city>
            <str>Szent István út 1001</str>
        </address>
    </subscriber>

    <subscriber sub_id="0003">

        <email>geza@gmail.com</email>
        <email>fejedelem@citromail.hu</email>
        <email>honfoglalas@magyarorszag.hu</email>
        <phone>+36204567893</phone>
        <address>
            <ZIP>2434</ZIP>
            <city>Hantos</city>
            <str>Kossuth Lajos út 2</str>
        </address>
    </subscriber>
```

```
    <!-- end -->

    <!-- Bills -->
    <!-- begin -->
    <bill bill_sub="0001" bill_plan="34670001">
        <plan_no>1</plan_no>
    </bill>
    <bill bill_sub="0001" bill_plan="34670002">
        <plan_no>2</plan_no>
    </bill>
    <bill bill_sub="0002" bill_plan="24350001">
        <plan_no>1</plan_no>
    </bill>
    <bill bill_sub="0003" bill_plan="24340001">
        <plan_no>1</plan_no>
    </bill>
    <!-- end -->

    <!-- Plans -->
    <!-- begin -->
    <plan p_id="34670001">
        <address>
            <ZIP>3467</ZIP>
            <city>Ároktő</city>
            <str>Kossuth Lajos út 3</str>
        </address>

    </plan>

    <plan p_id="34670002">
        <address>
            <ZIP>3592</ZIP>
            <city>Nemesbikk</city>
            <str>Posta utca 13</str>
        </address>

    </plan>

    <plan p_id="24350001">
        <address>
            <ZIP>2435</ZIP>
            <city>Nagylók</city>
            <str>Szent István út 1001</str>
        </address>

    </plan>

    <plan p_id="24340001">
        <address>
```

```
            <ZIP>2220</ZIP>
            <city>Vecsés</city>
            <str>Gárdony Géza utca 10</str>
        </address>

    </plan>
    <!-- end -->

    <!-- Services -->
    <!-- begin -->

    <service serv_sum="34670001" tel_serv="5120" i_serv="Cu120"
tv_serv="5050">
        <price_sum>9500</price_sum>
    </service>

    <service serv_sum="34670002" tel_serv="5120" i_serv="Co500"
tv_serv="5050">
        <price_sum>12500</price_sum>
    </service>

    <service serv_sum="24350001" tel_serv="00" i_serv="Op1000">
        <price_sum>19000</price_sum>
    </service>
    <service serv_sum="24340001" tel_serv="5050" i_serv="Co500"
tv_serv="2030">
        <price_sum>12000</price_sum>
    </service>

    <!-- end -->

    <!-- Internet plan types-->
    <!-- begin -->
    <internet i_id="Cu120">
        <technology>copper</technology>
        <bwidth>120</bwidth>
        <price>2000</price>
    </internet>

    <internet i_id="Co500">
        <technology>coaxial</technology>
        <bwidth>500</bwidth>
        <price>5000</price>
    </internet>

    <internet i_id="Op1000">
        <technology>optical</technology>
        <bwidth>1000</bwidth>
        <price>9000</price>
```

```
    </internet>
    <!-- end -->

    <!-- Telephone plan types -->
    <!-- begin -->
    <telephone tel_id="5120">
        <data_GB>5</data_GB>
        <free_mins>120</free_mins>
        <price>3500</price>
    </telephone>

    <telephone tel_id="5050">
        <data_GB>50</data_GB>
        <free_mins>50</free_mins>
        <price>5000</price>
    </telephone>

    <telephone tel_id="00">
        <data_GB>-1</data_GB>
        <free_mins>-1</free_mins>
        <price>10000</price>
    </telephone>
    <!-- end -->



    <!-- TV plan types -->
    <!-- begin -->
    <tv tv_id="2030">
        <ch_num>50</ch_num>
        <sd_num>30</sd_num>
        <hd_num>20</hd_num>
        <price>2000</price>
    </tv>

    <tv tv_id="5050">
        <ch_num>100</ch_num>
        <sd_num>50</sd_num>
        <hd_num>50</hd_num>
        <price>4000</price>
    </tv>

</telecommunications>
```

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

Az XML dokumentumhoz szükséges volt egy ún. XSD, azaz XML Schema Document elkészítése, ami a validációt segíti.
Lépésekre lebontva a következőket hajtottam végre:

1) Elemek, attribútumokat gyűjtöttem, majd egyszerű típusokat rendeltem hozzájuk
2) Simpletype-okat hoztam létre. Ezek végülis egyszerű típusok, viszont különböző kikötéseket tartalmaznak. Pl:
   a)
```
<xs:simpleType name="phoneType">
  <xs:restriction base="xs:string">
    <xs:pattern value="\+\d{11}" />
  </xs:restriction>
</xs:simpleType>
```
   Ahol egy phoneType-ot hozok létre, ami xs:string-en alapszik, viszont olyan kikötéssel, hogy mindenképp szerepelnie kell egy „+” -nak az elején
3) ComplexType-okat létrehoztam, amik különböző simpleType-okat tartalmaznak.
4) Kapcsolatokat hoztam létre
5) Ezt követi maga gyökérelem meghatározása, itt felhasználtam minden eddigi típust, hogy összeálljon a kép.
6) Elsődleges kulcsokat hoztam létre
7) Idegen kulcsokat hoztam létre

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

  <!-- Elements and attributes -->

  <xs:element name="plan_no" type="xs:integer"/>
  <xs:element name="city" type="xs:string"/>
  <xs:element name="str" type="xs:string"/>
  <xs:element name="price_sum" type="xs:integer"/>
  <xs:element name="data_GB" type="xs:integer"/>
  <xs:element name="free_mins" type="xs:integer"/>
  <xs:element name="price" type="xs:integer"/>
  <xs:element name="technology" type="xs:string"/>
  <xs:element name="bwidth" type="xs:string"/>
  <xs:element name="ch_num" type="xs:integer"/>
  <xs:element name="sd_num" type="xs:integer"/>
  <xs:element name="hd_num" type="xs:integer"/>

  <xs:attribute name="sub_id" type="xs:string"/>
  <xs:attribute name="bill_sub" type="xs:string"/>
  <xs:attribute name="bill_plan" type="xs:string"/>
  <xs:attribute name="p_id" type="xs:string"/>
  <xs:attribute name="serv_sum" type="xs:string"/>
  <xs:attribute name="tel_serv" type="xs:string"/>
  <xs:attribute name="i_serv" type="xs:string"/>
  <xs:attribute name="tv_serv" type="xs:string"/>
```

```
<xs:attribute name="tel_id" type="xs:string"/>
<xs:attribute name="i_id" type="xs:string"/>
<xs:attribute name="tv_id" type="xs:string"/>


<!-- Simple types -->

<xs:simpleType name="phoneType">
  <xs:restriction base="xs:string">
    <xs:pattern value="\+\d{11}" />
  </xs:restriction>
</xs:simpleType>

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

<xs:simpleType name="ZIPType">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{4}" />
  </xs:restriction>
</xs:simpleType>

<!-- Complex types -->

<xs:complexType name="addressType">
  <xs:sequence>
    <xs:element name="ZIP" type="ZIPType"/>
    <xs:element ref="city"/>
    <xs:element ref="str"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="subscriberType">
  <xs:sequence>
    <xs:element name="email" type="emailType" maxOccurs="unbounded"/>
    <xs:element name="phone" type="phoneType"/>
    <xs:element name="address" type="addressType"></xs:element>
  </xs:sequence>
  <xs:attribute ref="sub_id" use="required"/>
</xs:complexType>

<xs:complexType name="planType">
  <xs:sequence>
    <xs:element name="address" type="addressType"/>
  </xs:sequence>
  <xs:attribute ref="p_id" use="required"/>
```

```
</xs:complexType>



<xs:complexType name="telephoneType">
  <xs:sequence>
    <xs:element ref="data_GB"/>
    <xs:element ref="free_mins"/>
    <xs:element ref="price"/>
  </xs:sequence>
  <xs:attribute ref="tel_id" use="required"/>
</xs:complexType>

<xs:complexType name="internetType">
  <xs:sequence>
    <xs:element ref="technology"/>
    <xs:element ref="bwidth"/>
    <xs:element ref="price"/>
  </xs:sequence>
  <xs:attribute ref="i_id" use="required"/>
</xs:complexType>

<xs:complexType name="tvType">
  <xs:sequence>
    <xs:element ref="ch_num"/>
    <xs:element ref="sd_num"/>
    <xs:element ref="hd_num"/>
    <xs:element ref="price"/>
  </xs:sequence>
  <xs:attribute ref="tv_id" use="required"/>
</xs:complexType>

<!-- N:M switchboard -->

<xs:complexType name="billType">
  <xs:sequence>
    <xs:element ref="plan_no"/>
  </xs:sequence>
  <xs:attribute ref="bill_plan" use="required"/>
  <xs:attribute ref="bill_sub" use="required"/>
</xs:complexType>

<xs:complexType name="serviceType">
  <xs:sequence>
    <xs:element ref="price_sum"/>
  </xs:sequence>
  <xs:attribute ref="serv_sum" use="required"/>
  <xs:attribute ref="tel_serv" use="optional"/>
  <xs:attribute ref="i_serv" use="optional"/>
```

```
    <xs:attribute ref="tv_serv" use="optional"/>
  </xs:complexType>

  <!-- Telecommunications company -->

  <xs:element name="telecommunications">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="subscriber" type="subscriberType"
maxOccurs="unbounded"/>
        <xs:element name="bill" type="billType" maxOccurs="unbounded"/>
        <xs:element name="plan" type="planType" maxOccurs="unbounded"/>
        <xs:element name="service" type="serviceType" maxOccurs="unbounded"/>
        <xs:element name="internet" type="internetType"
maxOccurs="unbounded"/>
        <xs:element name="telephone" type="telephoneType"
maxOccurs="unbounded"/>
        <xs:element name="tv" type="tvType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>

    <!-- Primary Keys -->

    <xs:unique name="sub_id">
      <xs:selector xpath="subscriber"/>
      <xs:field xpath="@sub_id"/>
    </xs:unique>

    <xs:unique name="p_id">
      <xs:selector xpath="plan"/>
      <xs:field xpath="@p_id"/>
    </xs:unique>
    <xs:unique name="i_id">
      <xs:selector xpath="internet"/>
      <xs:field xpath="@i_id"/>
    </xs:unique>
    <xs:unique name="tel_id">
      <xs:selector xpath="telephone"/>
      <xs:field xpath="@tel_id"/>
    </xs:unique>
    <xs:unique name="tv_id">
      <xs:selector xpath="tv"/>
      <xs:field xpath="@tv_id"/>
    </xs:unique>

    <!-- Foreign keys -->

    <xs:keyref name="bill_subFK" refer="sub_id">
      <xs:selector xpath="bill"/>
```

```
      <xs:field xpath="@bill_sub"/>
    </xs:keyref>
    <xs:keyref name="bill_planFK" refer="p_id">
      <xs:selector xpath="bill"/>
      <xs:field xpath="@bill_plan"/>
    </xs:keyref>

    <xs:keyref name="tel_servFK" refer="tel_id">
      <xs:selector xpath="service"/>
      <xs:field xpath="@tel_serv"/>
    </xs:keyref>
    <xs:keyref name="i_servFK" refer="i_id">
      <xs:selector xpath="service"/>
      <xs:field xpath="@i_serv"/>
    </xs:keyref>
    <xs:keyref name="tv_servFK" refer="tv_id">
      <xs:selector xpath="service"/>
      <xs:field xpath="@tv_serv"/>
    </xs:keyref>
  </xs:element>

</xs:schema>
```

2. feladat

A feladatkiírás szerint Java nyelven készültek a DOM programok. A következőkben részletezem ezeket.

Megjegyzés: A programokban nem használok saját csomagot, mert nem tetszett a Linuxnak, de tökéletesen lefut mind

2a) Adatolvasás:

- A main metódus, miután megkereste a fájlt, beolvassa, illetve egy introduceFile metódussal parse-olja. Ezeket sikeres olvasás után kiírja konzolra, illetve fájlba „XMLiiju0zout.xml” néven.

```
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.io.IOException;

import org.xml.sax.SAXException; //There are import-ant

public class DomReadIIJU0Z {

    public static void main(String[] args) {
        File xmlFile = new File("XMLiiju0z.xml"); // First off, we need the
file we're going to read
        Document doc = parseXmlFile(xmlFile); // This one makes the
documentBuilderFactory and stuff. Looks nicer to
                                              // read, the function is
below

        if (doc == null) { // If no doc, sadness ensues and you can't parse
it. Too bad, program exits.
            System.out.println("The document is null");
            System.exit(-1);
        } else {
            docProcessor(doc); // If doc good, then the fun begins. Let's
start a function.
        }
    }

    private static void docProcessor(Document doc) { // Document processing
woohoo
        doc.getDocumentElement().normalize(); // Normalizing document so it
complies with OSHA and DOM rules
        System.out.println("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
        System.out.println("<" + doc.getDocumentElement().getNodeName() +
                " xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\"XSDiiju0z.xsd\">");
```

```
        NodeList nodeList = doc.getDocumentElement().getChildNodes();
        String indent = "";
        printOutput(nodeList, indent);
        System.out.println("</" + doc.getDocumentElement().getNodeName() +
">"); // Formatting the thingy basically

        writeToFile(doc, "XMLiiju0zout.xml"); // This one saves the new
document into the new file
    }

    private static void writeToFile(Document doc, String fileName) { //
This one does a bunch of do-hickeys to
                                                                    //
save the file. Ultimately it does save it
                                                                    // as
specified in the function call
        try {
            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "no");

            transformer.transform(new DOMSource(doc), new
StreamResult(fileName));

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

    public static Document parseXmlFile(File xmlFile) { // As i said
earlier, this one makes the DocumentBuilders
        Document doc = null;

        try {
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dbBuilder = dbFactory.newDocumentBuilder();
            doc = dbBuilder.parse(xmlFile);
        } catch (ParserConfigurationException | SAXException | IOException
e) {
            e.printStackTrace();
        }
        return doc;
    }

    private static void printOutput(NodeList nodeList, String indent) { /*
```

```
                                                                        *
This one lists everything. Essentially the
                                                                        *
most important in terms of non-IO functions.
                                                                        */
        indent += "\t";

        if (nodeList != null) {
            for (int i = 0; i < nodeList.getLength(); i++) {
                Node node = nodeList.item(i);
                if (node.getNodeType() == Node.ELEMENT_NODE &&
!node.getTextContent().trim().isEmpty()) {
                    processElementNode((Element) node, indent);
                } else if (node instanceof Text) {
                    String value = node.getNodeValue().trim();
                    if (!value.isEmpty()) {
                        System.out.println(indent + node.getTextContent());
                    }
                }
            }
        }
    }

    private static void processElementNode(Element element, String indent)
{
        System.out.print(indent + "<" + element.getNodeName());
        if (element.hasAttributes()) {
            processAttributes(element);
        }
        System.out.println(">");

        NodeList nodeList_new = element.getChildNodes();
        printOutput(nodeList_new, indent);
        System.out.println(indent + "</" + element.getNodeName() + ">");
    }

    private static void processAttributes(Element element) {
        NamedNodeMap attributes = element.getAttributes();
        for (int k = 0; k < attributes.getLength(); k++) {
            Node attribute = attributes.item(k);
            System.out.print(" " + attribute.getNodeName() + "=\"" +
attribute.getNodeValue() + "\"");
        }
    }
}
```

2b) Adatlekérdezés:

- A main az előzőhöz hasonlóan beparse-olja az XML fájlt, majd pedig lekérdezéseket hajt végre. Végül ezek eredményeit kiírja a konzolra. Például kiírja az összes előfizetést TV szolgáltatás nélkül

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

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import java.io.File;
import java.io.IOException;

public class DomQueryIIJU0Z {
    public static void main(String[] args) {
        File xmlFile = new File("XMLiiju0z.xml");
        Document doc = parseXmlFile(xmlFile);

        if (doc != null) {
            doc.getDocumentElement().normalize();

            /*
             * Lists subscribers' sub_id and their first email if they have
more than one
             * plan
             */
            querySubscribersWithMultiplePlans(doc);

            /* Shows the subscriber with the highest bill to pay */
            querySubscriberWithMaxBillPayment(doc);

            /* Shows the plans on Optical Cable technology */
            queryInternetPlansOnOpticalCable(doc);

            /* Shows the specified subscriber's main address */
            querySubscriberPlansAddress(doc, "0002");

            /* Lists every plan without a TV service */
            queryPlansWithoutTV(doc);

            /* Lists every plan and the price_sum */
            listPlansWithTotalPrice(doc);

        } else {
```

```
            System.out.println("The document is null");
            System.exit(-1);
        }
    }

    /* DocumentBuilderFactory and other things */
    private static Document parseXmlFile(File xmlFile) {
        Document doc = null;

        try {
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dbBuilder = dbFactory.newDocumentBuilder();
            doc = dbBuilder.parse(xmlFile);
        } catch (ParserConfigurationException | SAXException | IOException
e) {
            e.printStackTrace();
        }

        return doc;
    }

    /* Queries subscribers who have multiple plans */

    private static void querySubscribersWithMultiplePlans(Document doc) {
        NodeList subscribers =
doc.getDocumentElement().getElementsByTagName("subscriber");

        int maxPlans = 0;
        Element maxPlansSubscriber = null;

        /*
         * This is responsible for the big counting, but hands it down to
the
         * countPlansForSubscriber() function. That function gives a number
back and
         * this one does a maximum selection on it.
         */

        for (int i = 0; i < subscribers.getLength(); i++) {
            Element subscriber = (Element) subscribers.item(i);
            String subId = subscriber.getAttribute("sub_id");
            int planCount = countPlansForSubscriber(doc, subId);

            if (planCount > maxPlans) {
                maxPlans = planCount;
                maxPlansSubscriber = subscriber;
            }
        }
```

```
        /* This just writes it to the console. Nothing more. */
        if (maxPlansSubscriber != null) {
            String subId = maxPlansSubscriber.getAttribute("sub_id");
            NodeList emails =
maxPlansSubscriber.getElementsByTagName("email");
            String firstEmail = (emails.getLength() > 0) ?
emails.item(0).getTextContent() : "No Email";
            System.out.println("Subscriber ID: " + subId + ", First Email:
" + firstEmail);
        }
    }

    /*
     * This one is responsible for counting the plans. Counts every
bill_sub
     * attribute in the bills. Gives the number back to the function above.
     */

    private static int countPlansForSubscriber(Document doc, String subId)
{
        NodeList bills =
doc.getDocumentElement().getElementsByTagName("bill");
        int planCount = 0;

        for (int i = 0; i < bills.getLength(); i++) {
            Element bill = (Element) bills.item(i);
            String billSub = bill.getAttribute("bill_sub");

            if (subId.equals(billSub)) {
                planCount++;
            }
        }

        return planCount;
    }

    /*
     * This one queries the subscriber with the highest amount to pay. It
does that
     * by counting the summing the price_sums within the bills where
bill_sub
     * matches up. Then of course writes it out on the console.
     */
    private static void querySubscriberWithMaxBillPayment(Document doc) {
        NodeList bills =
doc.getDocumentElement().getElementsByTagName("bill");
        String maxSubscriberId = "";
        int maxPayment = Integer.MIN_VALUE;
```

```
        for (int i = 0; i < bills.getLength(); i++) {
            NodeList query = bills.item(i).getChildNodes();
            int payment = 0;

            for (int j = 0; j < query.getLength(); j++) {
                if (query.item(j).getNodeName().equals("price_sum")) {
                    payment +=
Integer.parseInt(query.item(j).getTextContent());
                }
            }

            if (payment > maxPayment) {
                maxPayment = payment;
                maxSubscriberId =
bills.item(i).getAttributes().getNamedItem("bill_sub").getNodeValue();
            }
        }

        NodeList subscribers =
doc.getDocumentElement().getElementsByTagName("subscriber");
        for (int i = 0; i < subscribers.getLength(); i++) {
            if
(subscribers.item(i).getAttributes().getNamedItem("sub_id").getNodeValue().
equals(maxSubscriberId)) {
                System.out.println("Highest paying customer: ");
                printOutput(subscribers.item(i).getChildNodes(), "");
                break;
            }
        }
    }

    /*
     * Very simple. Looks for "optical" in the internet plans and writes
its ID out
     * onto the console.
     */

    private static void queryInternetPlansOnOpticalCable(Document doc) {
        NodeList internetPlans =
doc.getDocumentElement().getElementsByTagName("internet");
        System.out.println("Internet plans on Optical cable:");

        for (int i = 0; i < internetPlans.getLength(); i++) {
            Element internetPlan = (Element) internetPlans.item(i);
            String technology =
internetPlan.getElementsByTagName("technology").item(0).getTextContent();

            if (technology.equals("optical")) {
                String iId = internetPlan.getAttribute("i_id");
```

```
                System.out.println(iId);
            }
        }
        System.out.println("-----------------------------------------------
--------------");
    }

    /*
     * In this one, you can specify which subscriber you want to look for
and then
     * it queries that subscriber along with their plan.
     */
    private static void querySubscriberPlansAddress(Document doc, String
targetSubId) {
        NodeList bills =
doc.getDocumentElement().getElementsByTagName("bill");

        for (int i = 0; i < bills.getLength(); i++) {
            Element bill = (Element) bills.item(i);
            if (bill.getAttribute("bill_sub").equals(targetSubId)) {
                String planId = bill.getAttribute("bill_plan");
                Element plan = getPlanElementById(doc, planId);

                if (plan != null) {
                    displayAddress(plan);
                }
            }
        }
    }

    /*
     * Helper function for the one above. This is where the magic happens
     * essentially. It queries the plans where the sub_id is identical to
the one we
     * want to look for.
     */
    private static Element getPlanElementById(Document doc, String planId)
{
        NodeList plans =
doc.getDocumentElement().getElementsByTagName("plan");

        for (int i = 0; i < plans.getLength(); i++) {
            Element plan = (Element) plans.item(i);
            if (plan.getAttribute("p_id").equals(planId)) {
                return plan;
            }
        }

        return null;
```

```
    }

    /*
     * Also a helper for the one above. This one lists the address of the
     * subscriber.
     */

    private static void displayAddress(Element plan) {
        NodeList addressNodes = plan.getElementsByTagName("address");
        if (addressNodes.getLength() > 0) {
            Element address = (Element) addressNodes.item(0);
            String zip =
address.getElementsByTagName("ZIP").item(0).getTextContent();
            String city =
address.getElementsByTagName("city").item(0).getTextContent();
            String street =
address.getElementsByTagName("str").item(0).getTextContent();

            System.out.println("Subscriber Address: ZIP " + zip + ", City:
" + city + ", Street: " + street);
        }
    }

    /*
     * This one looks for plans without any TV subscription. Basically if
it doesn't
     * find a "tv_serv" attribute in the <plan> tag, it lists that plan.
     */

    private static void queryPlansWithoutTV(Document doc) {
        NodeList services =
doc.getDocumentElement().getElementsByTagName("service");

        for (int i = 0; i < services.getLength(); i++) {
            Element service = (Element) services.item(i);
            if (!service.hasAttribute("tv_serv")) {
                String planId = service.getAttribute("serv_sum");
                System.out.println("Plan without TV: " + planId);
            }
        }
    }

    /* This one sums the total price of plans up and displays them. */
    private static void listPlansWithTotalPrice(Document doc) {
        NodeList services =
doc.getDocumentElement().getElementsByTagName("service");

        for (int i = 0; i < services.getLength(); i++) {
            Element service = (Element) services.item(i);
```

```
            String planId = service.getAttribute("serv_sum");
            int totalPrice =
Integer.parseInt(service.getElementsByTagName("price_sum").item(0).getTextC
ontent());
            System.out.println("Plan " + planId + " - Total Price: " +
totalPrice);
        }
    }

    /* This one prints everything onto the screen */

    public static void printOutput(NodeList nodeList, String indent) {
        indent += "\t";

        if (nodeList != null) {
            for (int i = 0; i < nodeList.getLength(); i++) {
                Node node = nodeList.item(i);
                if (node.getNodeType() == Node.ELEMENT_NODE &&
!node.getTextContent().trim().isEmpty()) {
                    System.out.print(indent + "<" + node.getNodeName());
                    if (node.hasAttributes()) {
                        for (int k = 0; k <
node.getAttributes().getLength(); k++) {
                            Node attribute = node.getAttributes().item(k);
                            System.out.print(" " + attribute.getNodeName()
+ "=\"" + attribute.getNodeValue() + "\"");
                        }
                        System.out.println(">");
                    } else {
                        System.out.println(">");
                    }

                    NodeList nodeList_new = node.getChildNodes();
                    printOutput(nodeList_new, indent);
                    System.out.println(indent + "</" + node.getNodeName() +
">");
                } else if (node instanceof Text) {
                    String value = node.getNodeValue().trim();
                    if (value.isEmpty()) {
                        continue;
                    }
                    System.out.println(indent + node.getTextContent());
                }
            }
        }
    }
}
```

2c) Adatmódosítás:

- A main elsőként meghívja a parseXmlFile függvényt, ami megnyitja, illetve parse-olja az XML fájlt. Ezt követően elvégzi a módosításokat a modifyValues függvényben és kiírja azokat az egész fájlt a konzolra a printXml függvénnyel. Például: minden irányítószámot átváltoztatunk „3333”-ra.

```
import java.io.File;
import java.io.IOException;

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.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class DomModifyIIJU0Z {
    public static void main(String[] args) {
        File xmlFile = new File("XMLiiju0z.xml");
        Document doc = parseXmlFile(xmlFile);

        if (doc == null) {
            System.out.println("The document is null");
            System.exit(-1);
        } else {
            doc.getDocumentElement().normalize();
        }

        /*This function applies the modifications*/
        modifyValues(doc.getDocumentElement());

        /*Print the modified XML to the screen*/
        printXml(doc);
    }

    /*Introduces file just like in the previous programs */
    public static Document parseXmlFile(File xmlFile) {
        Document doc = null;

        try {
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dbBuilder = dbFactory.newDocumentBuilder();
            doc = dbBuilder.parse(xmlFile);
```

```
        } catch (ParserConfigurationException | SAXException | IOException
e) {
            e.printStackTrace();
        }
        return doc;
    }


    /*This is where the magic happens*/
    public static void modifyValues(Element element) {
        NodeList childNodes = element.getChildNodes();

        for (int i = 0; i < childNodes.getLength(); i++) {
            Node node = childNodes.item(i);

            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element childElement = (Element) node;

                /*Increase the prices of internet plans by 2000*/
                if ("internet".equals(childElement.getNodeName())) {
                    int currentPrice =
Integer.parseInt(childElement.getElementsByTagName("price").item(0).getText
Content());
                    int newPrice = currentPrice + 2000;
                    childElement.getElementsByTagName("price").item(0).setT
extContent(Integer.toString(newPrice));
                }

                /*Adds "lol" at the end of every email address lol*/
                if ("email".equals(childElement.getNodeName())) {
                    childElement.setTextContent(childElement.getTextContent
() + "lol");
                }

                /*Changes every ZIP to 3333*/
                if ("ZIP".equals(childElement.getNodeName())) {
                    childElement.setTextContent("3333");
                }

                /*Reduces HD TV channel number by 15*/
                if ("tv".equals(childElement.getNodeName())) {
                    int currentHdNum =
Integer.parseInt(childElement.getElementsByTagName("hd_num").item(0).getTex
tContent());
                    int newHdNum = currentHdNum - 15;
                    childElement.getElementsByTagName("hd_num").item(0).set
TextContent(Integer.toString(newHdNum));
                }
```

```
                /*Adds "eeeeeeeee" at the beginning of every street name*/
                if ("str".equals(childElement.getNodeName())) {
                    childElement.setTextContent("eeeeeeeee" +
childElement.getTextContent());
                }

                /*Increase plan_no by 1000, so everyone has thousands of
plans ehehehehe*/
                if ("plan_no".equals(childElement.getNodeName())) {
                    int currentPlanNo =
Integer.parseInt(childElement.getTextContent());
                    int newPlanNo = currentPlanNo + 1000;
                    childElement.setTextContent(Integer.toString(newPlanNo)
);
                }

                /* Actually apply these modifications to targeted
elements*/
                modifyValues(childElement);
            }
        }
    }
    public static void printXml(Document doc) {
        try {
            /*Transformer factory let's go */
            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");

            /*Printy printy :3*/
            DOMSource source = new DOMSource(doc);
            StreamResult consoleResult = new StreamResult(System.out);
            transformer.transform(source, consoleResult);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2d) Adatírás:

- A main létrehozza az új fájlt „telecommunications” gyökérelemmel. Minden elemet egyenként létrehoz, majd hozzáragaszt az xml fájlhoz. Ezt követően elmenti egy fájlba az újonnan írt dokumentumot

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

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.io.FileWriter;
import java.util.Arrays;
import java.util.List;

public class DomWriteIIJU0Z {

    public static void main(String[] args) {
        try {
            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance(); // blah blah blah

       // DocumentBuilderFactory. Same as

       // before
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document document = builder.newDocument();
            Element rootElement =
document.createElement("telecommunications");
            rootElement.setAttribute("xmlns:xsi",
"http://www.w3.org/2001/XMLSchema-instance");
            rootElement.setAttribute("xsi:noNamespaceSchemaLocation",
"XSDiiju0z.xsd");
            document.appendChild(rootElement);

            /*
             * Tedious processing of the different elements. They all
follow the same model,
             * so we'll take a look at one
             */

            processSubscribers(document, rootElement, "0001",
                    Arrays.asList("piroskaneni@piroskaneni.hu",
"piroskaburner@gmail.com", "piros@freemail.hu"),
```

```
                    "+36466748449", "3467", "Ároktő", "Kossuth Lajos út
3");
            processSubscribers(document, rootElement, "0002",
                    Arrays.asList("istvan@gmail.com",
"akiraly@citromail.hu", "istvankiraly@allamalapitas.hu"),
                    "+36204567892", "2435", "Nagylók", "Szent István út
1001");
            processSubscribers(document, rootElement, "0003",
                    Arrays.asList("geza@gmail.com",
"fejedelem@citromail.hu", "honfoglalas@magyarorszag.hu"),
                    "+36204567893", "2434", "Hantos", "Kossuth Lajos út
2");

            processBills(document, rootElement, "0001", "34670001", 1);
            processBills(document, rootElement, "0001", "34670002", 2);
            processBills(document, rootElement, "0002", "24350001", 1);
            processBills(document, rootElement, "0003", "24340001", 1);

            processPlans(document, rootElement, "34670001", "3467",
"Ároktő", "Kossuth Lajos út 3");
            processPlans(document, rootElement, "34670002", "3592",
"Nemesbikk", "Posta utca 13");
            processPlans(document, rootElement, "24350001", "2435",
"Nagylók", "Szent István út 1001");
            processPlans(document, rootElement, "24340001", "2220",
"Vecsés", "Gárdony Géza utca 10");

            processServices(document, rootElement, "34670001", "5120",
"Cu120", "5050", 9500);
            processServices(document, rootElement, "34670002", "5120",
"Co500", "5050", 12500);
            processServices(document, rootElement, "24350001", "00",
"Op1000", null, 19000);
            processServices(document, rootElement, "24340001", "5050",
"Co500", "2030", 12000);

            processInternetTypes(document, rootElement, "Cu120", "copper",
120, 2000);
            processInternetTypes(document, rootElement, "Co500", "coaxial",
500, 5000);
            processInternetTypes(document, rootElement, "Op1000",
"optical", 1000, 9000);

            processTelephoneTypes(document, rootElement, "5120", 5, 120,
3500);
            processTelephoneTypes(document, rootElement, "5050", 50, 50,
5000);
            processTelephoneTypes(document, rootElement, "00", -1, -1,
10000);
```

```
            processTVTypes(document, rootElement, "2030", 50, 30, 20,
2000);
            processTVTypes(document, rootElement, "5050", 100, 50, 50,
4000);

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

    /*
     * So in the declaration we have all the elements we need the program
to make,
     * except Document and rootElement. Those are mandatory to have.
Basically the
     * function creates a bunch of new Elements, that it later appends to
the root.
     * Pretty simple once you understand it.
     */
    private static void processSubscribers(Document document, Element
rootElement, String sub_id,
            List<String> emailList, String phone, String zip, String city,
String str) {
        Element sub = document.createElement("subscriber");
        sub.setAttribute("sub_id", sub_id);

        Element emailE = document.createElement("email");
        for (String s : emailList) {
            Element temp = createElement(document, "email", s);
            emailE.appendChild(temp);
        }
        sub.appendChild(emailE);

        Element phoneE = createElement(document, "phone", phone);
        sub.appendChild(phoneE);

        Element addressE = document.createElement("address");
        Element zipE = createElement(document, "ZIP", zip);
        Element cityE = createElement(document, "city", city);
        Element strE = createElement(document, "str", str);

        addressE.appendChild(zipE);
        addressE.appendChild(cityE);
        addressE.appendChild(strE);

        sub.appendChild(addressE);
        rootElement.appendChild(sub);
```

```
    }

    /* Same as before */

    private static void processBills(Document document, Element
rootElement, String bill_sub, String bill_plan,
            int plan_no) {
        Element bill = document.createElement("bill");
        bill.setAttribute("bill_sub", bill_sub);
        bill.setAttribute("bill_plan", bill_plan);

        Element planNo = createElement(document, "plan_no",
String.valueOf(plan_no));
        bill.appendChild(planNo);

        rootElement.appendChild(bill);
    }
    /* Same as before */

    private static void processPlans(Document document, Element
rootElement, String p_id, String zip, String city,
            String str) {
        Element plan = document.createElement("plan");
        plan.setAttribute("p_id", p_id);

        Element addressE = document.createElement("address");
        Element zipE = createElement(document, "ZIP", zip);
        Element cityE = createElement(document, "city", city);
        Element strE = createElement(document, "str", str);

        addressE.appendChild(zipE);
        addressE.appendChild(cityE);
        addressE.appendChild(strE);

        plan.appendChild(addressE);
        rootElement.appendChild(plan);
    }

    /*
     * These are all the same, but different elements. Really no difference
in logic
     * here.
     */

    private static void processServices(Document document, Element
rootElement, String serv_sum, String tel_serv,
            String i_serv, String tv_serv, int price_sum) {
        Element service = document.createElement("service");
        service.setAttribute("serv_sum", serv_sum);
```

```
        service.setAttribute("tel_serv", tel_serv);
        service.setAttribute("i_serv", i_serv);
        if (tv_serv != null) {
            service.setAttribute("tv_serv", tv_serv);
        }

        Element priceSum = createElement(document, "price_sum",
String.valueOf(price_sum));
        service.appendChild(priceSum);

        rootElement.appendChild(service);
    }

    private static void processInternetTypes(Document document, Element
rootElement, String i_id, String technology,
            int bwidth, int price) {
        Element internet = document.createElement("internet");
        internet.setAttribute("i_id", i_id);

        Element technologyE = createElement(document, "technology",
technology);
        Element bwidthE = createElement(document, "bwidth",
String.valueOf(bwidth));
        Element priceE = createElement(document, "price",
String.valueOf(price));

        internet.appendChild(technologyE);
        internet.appendChild(bwidthE);
        internet.appendChild(priceE);

        rootElement.appendChild(internet);
    }

    private static void processTelephoneTypes(Document document, Element
rootElement, String tel_id, int data_GB,
            int free_mins, int price) {
        Element telephone = document.createElement("telephone");
        telephone.setAttribute("tel_id", tel_id);

        Element dataGB = createElement(document, "data_GB",
String.valueOf(data_GB));
        Element freeMins = createElement(document, "free_mins",
String.valueOf(free_mins));
        Element priceE = createElement(document, "price",
String.valueOf(price));

        telephone.appendChild(dataGB);
        telephone.appendChild(freeMins);
        telephone.appendChild(priceE);
```

```
        rootElement.appendChild(telephone);
    }

    private static void processTVTypes(Document document, Element
rootElement, String tv_id, int ch_num, int sd_num,
            int hd_num, int price) {
        Element tv = document.createElement("tv");
        tv.setAttribute("tv_id", tv_id);

        Element chNum = createElement(document, "ch_num",
String.valueOf(ch_num));
        Element sdNum = createElement(document, "sd_num",
String.valueOf(sd_num));
        Element hdNum = createElement(document, "hd_num",
String.valueOf(hd_num));
        Element priceE = createElement(document, "price",
String.valueOf(price));

        tv.appendChild(chNum);
        tv.appendChild(sdNum);
        tv.appendChild(hdNum);
        tv.appendChild(priceE);

        rootElement.appendChild(tv);
    }

    /*
     * This one simplifies element creation, reduces it to one command,
instead of
     * three.
     */
    private static Element createElement(Document document, String tagName,
String textContent) {
        Element element = document.createElement(tagName);
        element.setTextContent(textContent);
        return element;
    }

    /* This one creates the output. Same as in the DOMRead program
essentially. */
    private static void printDocument(Document document) {
        try {
            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");

            /*
```

```
            * This is the line that actually creates the new file named:
"XMLiiju0z1.xml".
            * Note that the original is called: "XMLiiju0z.xml", meaning i
put a "1" before
            * the file extension. You will know it's the program's
creation if it doesn't
            * have any comments
            */
           transformer.transform(new DOMSource(document), new
StreamResult("XMLiiju0z1.xml"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```