**DOM**

. A dokumentum kezel˝o nyelv´enek azonban nemcsak a lek´erdez´esre kell szor´ıtkoznia, hiszen a dokumentum tartalma az id˝ok sor´an m´odosulhat, b˝ovu¨lhet. Emiatt szu¨ks´eg van az adatelemek m´odos´ıt´as´ara, t¨orl´es´ere ´es u´j adatelemek beszu´r´as´ara. A procedur´alsi kezel˝o felu¨letek k¨oz¨ott a k´et legfontosabb szabv´any a SAX

´es a DOM API felu¨let. A SAX [**?**] (Simple API for XML) kezel˝o felu¨let egy egyszeru˝bb funcki´ok¨ort biztos´ıt. A SAX mu˝k¨od´esi elve a k¨ovetkez˝o: Az

´ertelmez˝o szekvenci´alisan ´atolvassa az XML dokumentumot. Ha a doku- mentumban ´ep´ıt˝o elemhez (p´eld´aul elem, sz¨oveg, elemjellemz˝o) ´er, akkor egy esem´eny v´alt´odik ki. A feldolgoz´o motor az esem´eny bek¨ovetkez´esekor megk- eresi a esem´enyhez rendelt kezel˝o rutint. A kezel˝o rutin konkr´et alakj´at a programoz´o hat´arozza meg. A megh´ıvott rutin azonban nem hat vissza a fel- dolgoz´as alatt ´all´o dokumentumra. Emiatt a SAX felu¨let az XML dokumen- tumok olvas´as mu˝veleteihez haszn´alatos. Term´eszetesen az esem´enykezel˝o rutinok l´etrehozhatnak egy u´j XML dokumentumot, melynek tartalma a be-

olvasott dokumentumb´ol sz´armaztat´odik, de az m´ar egy u´j, m´asik XML

dokumentumot jelent. A SAX f˝o el˝onye a hat´ekony v´egrehajt´as, hiszen

egy egyszeru˝ erkezete.

szekvenci´alis ´atfut´assal gyorsan felt´arhat´o a dokumentum.

Egy t¨obb funkci´ot, a m´odos´ıt´asi mu˝veleteket is lehet˝ov´e t´ev˝o kezel˝o felu¨let a DOM API. A DOM [**?**] (Document Object Model) l´enyeges tulajdons´aga hogy itt a kezel˝o felu¨let a mem´ori´aban fel´ıti a teljes dokumentum objektu- mot, s a rendelkez´esre ´all´o met´odusokon keresztu¨l elv´egezhet˝o a tartalom

´atolvas´asa ´es m´odos´ıt´asa is. A mem´ori´aban l´ev˝o dokumentum k´ep teh´at v´altozhathat´o, s szu¨ks´eg eset´en f´ajlba is ki´ırhat´o. Az kimeneti ´allom´any lehet azonos a bemeneti ´allom´annyal, de lehet t˝ole ku¨l¨onb¨oz˝o ´allom´any is. A DOM modellben az XML dokumentumot a m´ar megismert fa reprezent´aci´oval ´ırj´ak le. A DOM API teh´at az XML fa manipul´as´ara szolg´al´o elemeket tartalmaz. Az objektum jelleg arra urtal, hogy mind a teljes fa, mint annak elemei

egy-egy objektumk´ent kezelhet˝o. ´Igy a teljes f´at reprezent´al´o dokumentum

mellett megjelennek az elemeket, a sz¨ovegr´eszekete, az elemjellemz˝oket le´ır´o objektumok is. Az elemek k¨oz¨otti tartalmaz´asi rel´aci´o az objektumok hier- archi´aj´ara k´epz˝odik le.

A fa kezel´esi logik´aja alapvet˝oen navig´aci´os jellegu˝. Ez azt jelenti, hogy a fa feldolgoz´asa sor´an egy feldolgoz´asi poz´ıci´o jelenik meg, amely megmu- tatja, hogy a fa melyik r´esze,mely objektuma a feldolgoz´as alatt ´all´o elem. Az elem v´alt´askor alapvet˝oen az aktu´alis elem szomsz´edaira lehet ´atl´epni,

navig´alni. A m´odos´ıt´as jellegu˝ mu˝veletekn´el az aktu´alis elem kornyez¨ ete´t

lehet m´odos´ıtani. A nem helyhez k¨ot¨ott mu˝veletek, mint p´eld´aul egy u´j

elem l´etrehoz´asa, a teljes dokumentumot le´ır´o objektumhoz k¨ot˝odnek. A DOM API teh´at oszt´alyok gyu¨jtem´enye, melyben a defin´ı´alt adattagok az elemek jellemz˝oit ´ırj´ak le, a defini´alt met´odusok a navig´aci´ora ´es a tartalom kezel´esre szolg´alnak. A programoz´o feladata olyan programk´odot ´ırni, mely- ben a p´eld´anyos´ıtja a kezel˝o oszt´alyokat ´es elv´egzi a fa m´odos´ıt´as´at a ren- delkez´esre ´all´o met´odusok seg´ıts´eg´evel.

A DOM modell mu˝k¨od´esi elv´et vizsg´alva, elmodnhat´o, hogy ez nagy- obb v´egrahajt´asi k¨olts´eggel j´ar, mint a SAX modell. Ennek oka, hogy a DOM els˝o l´ep´esben a ´atolvassa a megadott bemeneti XML dokumentumot, majd a mem´ori´aban fel´ep´ıti a hozz´a kapcsol´od´o obejktum hierarchi´at. A fa fel´ep´ıt´eshez a DOM az SAX API-t haszn´alja fel. A DOM egyik kritikus pontja, hogy a rendelkez´esre ´all´o mem´oria v´eges, ez´ert a hat´ekonyan feldolgo- zhat´o XML dokumentum m´eret is korl´atos. A mem´oria korl´atok kikeru¨l´es´ere sz´amos olyan javaslat van, mely nem olvassa be egyszerre a teljes dokumen- tumot, ehelyett csak a feldolgoz´as alatt ´all´o dokumentum r´eszlet keru¨l be a mem´ori´aba.

# Az XML-fa DOM objektum modelje

Az XML dokumentumokhoz az el˝oz˝o fejezetben ismertetett fa struktu´r´aval adhat´o meg a logikai szerkezetet. A DOM modell objektum hierarchi´aja is alapvet˝oen arre a logikai fa szerkezetre ´epu¨l, kieg´esz´ıtve n´eh´any imple- ment´aci´o-fu¨gg˝o elemmel. A f´aban az al´abbi csom´opont t´ıpusokat ku¨l¨onb¨ozteti meg:

* dokumentum: a teljes f´at reprezent´alja

dokumentum s´ema le´ır´as: a fa szerkezet´ere vonatkoz´o megk¨ot´esek, in- tegrit´asi szab´alyok csom´opontja

*•*

* tartalomelem csom´opont: egy ´altal´anos u¨res vagy ¨osszetett elem csom´opont
* sz¨oveg csom´opont
* megjegyz´es csom´opont
* elem jellemz˝o csom´opontja
* elem hivatkoz´as csom´opont

Az egyes elemek k¨oz¨ott tartalmaz´asi rel´aci´o ´all´ıthat´o fel. A DOM ´ertelmez˝o, feldolgoz´o a bemeneti XML dokumentumhoz el˝o´all´ıtja a fenti elemeket tar- talmaz´o XML-f´at. Mintak´ent vegyu¨k az al´abbi XML dokumentumot:

<?xml version="1.0" ?>

<!-- mintap´elda -->

<konyvek xmlns="[www.lib.org/books"](http://www.lib.org/books)>

<konyv id="1001">

<cim> Tornado a godorben </cim>

<ibook:isbn xmlns:ibook=["www.book.org/codes](http://www.book.org/codes)"> 1156511-568-89 </ibook:isbn>

<ar pnem="Ft"> 2455 </ar>

<benn/>

<kiado idref="K354" />

<szerzo> Kiss P´eter </szerzo>

<szerzo> Kovacs Lajos </szerzo>

</konyv>

</konyvek>

Az elk´esz´ıtett DOM-fa fel´ep´ıt´ese a k¨ovetkez˝o:

dokumentum

|- direkt´ıva // <?xml version="1.0" ?>

|

|- megjegyz´es // <!-- mintap´elda -->

|

|- elem (gy¨ok´er) // <konyvek xmlns="...">

|- n´evt´er // xmlns="[www.lib.org/books](http://www.lib.org/books)"

|- sz¨oveg // ¨ures!

|- elem // <konyv id="1001">

|- elemjellemz¨o // id="1001"

| |- sz¨oveg // 1001

|- sz¨oveg // ¨ures!

|

|- elem // <cim> Tornado a godorben </cim>

| |- sz¨oveg // Tornado a godorben

|- sz¨oveg // ¨ures!

|

|- elem // <ibook:isbn ... </ibook:isbn>

| |- n´evt´er // xmlns:ibook=["www.book.org/codes"](http://www.book.org/codes)

| |- sz¨oveg // 1156511-568-89

|- sz¨oveg // ¨ures!

|

|- elem // <ar pnem="Ft"> 2455 </ar>

| |- jellemz¨o // pnem="Ft"

| | |-sz¨oveg // Ft

| |- sz¨oveg // 2455

|- sz¨oveg // ¨ures!

|

|- elem // <benn/>

|- sz¨oveg // ¨ures!

|

|- elem // <kiado idref="K354" />

| |-hivatkoz´as // idref="K354"

| |- sz¨oveg // K354

|- sz¨oveg // ¨ures!

|

|- elem // <szerzo> Kiss P´eter </szerzo>

| |-sz¨oveg // Kiss P´eter

|-sz¨oveg // ¨ures!

|

|- elem // <szerzo> Kovacs Lajos </szerzo>

|-sz¨oveg // Kovacs Lajos

M´ar a p´elda alapj´an is ´erz´ekelhet˝o, hogy eg´esz kis XML dokumentumok eset´eben is el´eg tereb´elyes DOM-fa j¨on l´etre. A feldolgoz´as sor´an a f´aban, ezen csom´opontok ment´en lehet navig´alni. A m´odos´ıt´asokat az aktu´alis csom´oponthoz kapcsol´od´oan lehet elv´egezni.

A modellben az al´abbi navig´aci´os l´ep´esek ´allnak rendelkez´esre:

* mozg´as a gyerekek fel´e
* navig´aci´o a testv´er csom´opontok fel´e
* ´atl´ep´es a szu¨l˝o csom´opontra

# DOM API elemei

A DOM API szabv´any interface-eket defin´al, s nem oszt´alyokat. Ez´altal csak az alap funkcionalit´as biztos´ıt´asa van el˝o´ırva, szabad utat engedve a b˝ov´ıt´eseknek. A DOM API interface gyu¨jtem´enye mag´aba foglalja a kliens programban p´eld´anyos´ıthat´o DOM-fa kezel˝o objektumok le´ır´as´at. A DOM API egyr´eszt tartalmazza az egyes csom´opont t´ıpusakat megval´os´ıt´o inter- faceket, m´asr´eszt tov´abbi ¨osszetettebb funkci´ot ell´at´o elemek is megjelentek a szabv´anyban. ´Igy p´eld´aul az API tartalmaz csom´oponthalmazt reprent´al´o interface-t is. Ez az interface alkalmazhat´o t¨obbek k¨oz¨ott az egy adott csom´oponthoz tartoz´o gyerek csom´opontokk t´arol´as´ara.

A k¨ovetkez˝okben a DOM API fontosabb interface elemeit tekintju¨k ´at.

* *Document* : a dokumentum csom´opont megtestes´ıt˝oje. Adattagok:
  + documentElement: a fa gy¨ok´er eleme, Element t´ıpusu´
  + doctype : s´ema l´ır´o, DocType t´ıpusu´

A dokumentum objektum szerepe a teljes dokumentum egys´egbe z´ar´asa

´es az alkot´o elemek teljes k¨oru˝ kezel´ese. Emiatt ide kapcsol´odnak az

´altal´anos elem l´etrehoz´o met´odusok is. Az interface f˝obb met´odusai:

* + Element createElement (String elemn´ev): u´j elem objektum l´etrehoz´asa
  + Text createTextNode(String sz¨oveg): sz¨oveg csom´opont l´etrehoz´asa
  + Comment createComment (String sz¨oveg): megjegyz´es csom´opont l´etrehoz´asa
  + Attr createAttribute (String jellemz¨o neve): elemjellemz˝o l´etrehoz´asa
  + NodeList getElementsbyTagName (String elemn´ev): a fa megadott n´evvel rendelkez˝o csom´opontjainak ¨osszegyu¨jt´ese
* *Node*: ´altal´anos fa csom´opont. Fontosabb adattagok:
  + nodeType: csom´opont t´ıpusa (pl. elem, sz¨oveg), t´ıpusa: short
  + nodeName: csom´opont neve, t´ıpusa String
  + nodeValue: csom´opont ´ert´eke, t´ıpusa String
  + parentNode: szu¨l˝o csom´opont, t´ıpusa Node
  + childNodes: gyerek csom´opontok halmaza, t´ıpusa: NodeList
  + nextSibling: testv´er csom´opont, t´ıpusa: Node
  + Attributes: jellemz˝ok halmaza, t´ıpusa: NamedNodeMap
  + ownerDocument: befoglal´o dokumentum objektum, t´ıpusa: Doc- ument

A objektumhoz tartoz´o met´odusok:

* + Node removeChild(Node elem) : a megadott gyermek csom´opont kit¨orl´ese a f´ab´ol
  + Node appendChild(Node elem) : a megadott csom´opont beszu´r´asa a f´aba az aktu´alis csom´opont al´a gyermekk´ent
  + Node replaceChild(Node elemuj, Node elemregi) : a megadott gy- ermek csom´opont helyettse´ıt´ese egy m´asik elemmel
  + Node insertBefore(Node elemuj, Node elemregi) : a megadott csom´opont beszu´r´asa a f´aba az aktu´alis csom´opont al´a gyermekk´ent, a megadott l´etez˝o gyermek el´e

*CharacterData*: ´altal´anos sz¨ovegr´esz, a Node interface-b˝ol sz´armaztatott. K´et adattagja van:

*•*

* + data: sz¨oveges adat le´ır´esa, t´ıpusa String

length: az sz¨oveg hossza, t´ıpusa long

A sz¨ovegkezel˝o met´odusok list´aja:

* + String substringData (long kezdet, long hossz): sz¨ovegr´esz kiemel´ese
  + deleteData (long kezdet, long hossz): sz¨ovegr´esz kit¨orl´ese
  + insertData (long kezdet, String sz¨oveg): sz¨ovegr´esz beszu´r´asa amegadott poz´ıci´ora
* *Attr* : elemjellemz˝o le´ır´asa, a Node interface-b˝ol sz´armaztatott. Adattagok:
  + name: az jellemz˝o azonos´ıt´o neve, t´ıpusa String
  + specified: van-e ´ert´eke a jellemz˝onek vagy sem, t´ıpusa: Boolean
  + value: a jellemz˝o ´ert´eke, t´ıpusa: String Az objektumhoz nem tartozik saj´at met´odus.

*Element* : tartalom elem csom´opontja, a Node interface-b˝ol sz´armaztatott. Ide tartoz´o speci´alis met´odusok:

*•*

* + String getAttribute (String n´ev): a jellemz˝o ´ert´ek´enek lek´erdez´ese a jellemz˝o neve alapj´an

setAttribute (String n´ev, String ´ert´ek): a megadott elemjellemz˝o

´ert´ek´enek be´all´ıt´asa

* + removeAttribute(String n´ev): megadott elemjellemz˝o elt´avol´ıt´asa a f´ab´ol
  + Attr getAttributeNode (String n´ev): a megadott n´evvel rendelkez˝o elemjellemz˝o csom´opont objektum ´atv´etele
  + NodeList getElementsbyTagName(String n´ev): a megadott n´evvel rendelkez˝o lesz´armazott csom´opontok list´aj´anak el˝o´all´ıt´asa
* *NodeList* : csom´opontok halmaza, egy adattagja ´es egy met´odusa van:
  + length : a tartalmazott csom´opontok darabsz´ama, t´ıpusa: long
  + Node item(long index) : a megadott sorsz´amu´ csom´opont lek´er´ese

*DOMException*: DOM kezel˝o felu¨let ´altal gener´alt hiba. Egyetlen at- tribu´tuma van, melynek neve

*•*

* + code: hibak´od, short t´ıpusu´

# DOM API program fel´ep´ıt´ese

A mintaprogramban mindenek el˝ott l´etre kell hozni egy olyan objektumot, amely alkalmas a dokumentum DOM modell alapu´ kezel´es´ere. Ez a feldolgoz´o objektum a DocumentBuilder oszt´alyhoz tar- tozik. Mivel igen sok egyedi param´etert kell be´all´ıtani, a DocumentBuilder oszt´alyt nem lehet k¨ozvetlenu¨l a new oper´atorral l´etrehozni, ehelyett egy m´asik objektumot h´ıvunk meg a DocumentBuilder oszt´aly p´eld´anyos´ıt´as´ara. Ez a seg´ed objektum a DocumentBuilderFactory oszt´alyhoz tartozik. A DocumentBuilderFactory t´ıpusu´ objektum newDocumentBuilder met´odusa

hozza l´etre az ig´eny´elt DocumentBuilder t´ıpusu´ objektumot. A dolgok tov´abbi ´erdekess´ege, hogy a DocumentBuilderFactory oszt´alyu´ objektum l´etrehoz´asa sem a new oper´atorhoz kapcsol´odik. Ekkor a DocumentBuilderFactory oszt´aly statikus newInstance() met´odusa szolg´al a p´eld´anyos´ıt´asra.

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance(); DocumentBuilder db = dbf.newDocumentBuilder();

A l´etrej¨ott DOM ´ertelmez˝o m´ar alkalmas az ig´enylet DOM fa el˝o´all´ıt´as´ara. Ehhez a DocumentBuilder t´ıpusu´ objektum parse() met´odus´at kell megh´ıvni. A parse() met´odus az ig´enyelt Document objektumot adja vissza:

Document dom = db.parse(new File(args));

A k´es˝obbiekben szu¨ks´eg lehet arra, hogy a dokumentum bet¨olt´esekor a feldolgoz´o ellen˝orizze az XML dokumentum s´em´aj´anak helyess´eg´et is. En- nek sor´an az XML dokumentum ´allom´any aktu´alis fel´ep´ıt´es´et ¨osszeveti egy megadott s´ema le´ır´assal. Az ´ertelmez˝o csak akkor fogadja el az XML doku- mentumot, ha az megfelel a megadott s´ema integrit´asi elemeinek. A s´ema ellen˝orz´est a

dbf.setValidating(true);

DocumentBuilderFactory objektumnak sz´ol´o met´odush´ıv´assal lehet ak- tiviz´alni.

Mivel a ford´ıt´onak ismernie kell az oszt´alyok, met´odusok szignatu´r´aj´at, a felhaszn´alt API elemekt deklar´alni kell a program elej´en. A Java k¨ornyezetben erre az import utast´as szolg´al. A mint´ainkban az al´abbi be´all´ıt´asokra van szu¨ks´eg:

import com.sun.org.apache.xerces.internal.impl.xs.dom.DOMParser;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException; import org.w3c.dom.\*;

import org.xml.sax.SAXException;

* SAXException: a dokumentum ´atolvas´asa sor´an keletkezett hiba

# Minta feladat

Mintafeladatk´ent egy olyan DOMalkalamz´ast mutatunk be, melyben meg- tal´alhat´o az adatkezel´es n´egy legfontosabb alapmu˝velete:

* adott kulcsu´ rekord keres´ese
* adott felt´etelt teljes´ıt˝o rekordok keres´ese
* u´j rekord felvitele
* adott kulcsu´ rekord t¨orl´ese
* adott tulajdons´agu´ rekordok tartalm´anak m´odos´ıt´asa

A minta XML dokumentum egy k¨onyvt´ar le´ır´o ´allom´any, melyben az egyes k¨onyv le´ır´o elemek lesznek a rekordok. Minden rekord k´et mez˝ot tar- talmaz, melyeket befoglalt elemekkel val´os´ıtunk meg. A k´et mez˝o a k¨onyv c´ıme ´es ´ara. Egy minta XML dokumentumot szeml´etet az al´abbi minta:

<?xml version="1.0" ?>

<konyvek>

<konyv>

<cim>Google1</cim><ar>20233</ar>

</konyv>

<konyv>

<cim>Boomi</cim><ar>2233</ar>

</konyv>

</konyvek>

A programban az al´abbi egyedi met´odusok szerepelnek:

* query1(Document dom): a fa teljes tartalm´anak kilist´az´asa

query2(Document dom, int ar) : a megadott ´arn´al dr´ag´abb k¨onyvek kilist´az´asa

*•*

update1(Document dom, String nev, int ar) : a megadott c´ımu˝ k¨onyvek

*•*

´ar´anak ´at´ır´asa a megadott ´ar ´ert´ekre

delete1 (Document dom, String nev, int ar) : a megadott c´ımu˝ ´es ´aru´ k¨onyvek kit¨orl´ese

*•*

insert1 (Document dom, String nev, int ar) : megadott c´ımmel ´es ´ert´ekkel

u´j k¨onyv egyed l´etrehoz´asa

package xml;

import com.sun.org.apache.xerces.internal.impl.xs.dom.DOMParser; import java.io.\*;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory; import javax.xml.parsers.ParserConfigurationException; import org.w3c.dom.\*;

import org.xml.sax.SAXException;

public class XMLDOM { public XMLDOM() {

}

public void xmain(String args) {

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance(); try {

DocumentBuilder db = dbf.newDocumentBuilder(); Document dom = db.parse(new File(args)); query1(dom);

query2(dom, 3000); update1(dom, "Boomi",1000); delete1(dom,"Boomi",3233); insert1(dom,"Lulu",3001); query1(dom);

}catch(ParserConfigurationException pce) { pce.printStackTrace();

}catch(SAXException se) {

se.printStackTrace();

}catch(IOException ioe) { ioe.printStackTrace();

}

}

public void query1(Document dom) {

Node root = dom.getDocumentElement(); printtree(root," ");

}

public void printtree(Node elem, String pre) { String eredm = pre;

int p;

if (elem.getNodeType()==Node.ELEMENT\_NODE) { eredm = eredm.concat(elem.getNodeName()); System.out.println(eredm);

for (p=0; p<elem.getChildNodes().getLength(); p++) { Node elemuj = elem.getChildNodes().item(p); printtree(elemuj,pre.concat(" "));

}

for (p=0; p<elem.getAttributes().getLength();p++) { Node elemuj = elem.getAttributes().item(p); printtree(elemuj,pre.concat(" "));

}

}

if (elem.getNodeType()==Node.TEXT\_NODE) { String st = elem.getTextContent();

st = st.replace(’\n’,’ ’); st = st.trim();

if (st.length()>0) {

eredm = eredm.concat("\""); eredm = eredm.concat(st); eredm = eredm.concat("\""); System.out.println(eredm);

}

}

if (elem.getNodeType()==Node.ATTRIBUTE\_NODE) { String st = elem.getNodeName();

eredm = eredm.concat("["); eredm = eredm.concat(st); eredm = eredm.concat("=");

eredm = eredm.concat(elem.getNodeValue()); eredm = eredm.concat("]"); System.out.println(eredm);

}

}

public void query2 (Document dom, int minar) { int p,q,r;

int ear=0;

String ecim=null; Node elem = null;

Node root = dom.getDocumentElement();

for (p=0; p<root.getChildNodes().getLength(); p++) { elem = root.getChildNodes().item(p);

if (elem.getNodeName().compareTo("konyv") == 0) {

for (q=0; q<elem.getChildNodes().getLength(); q++) { Node elemuj = elem.getChildNodes().item(q);

if (elemuj.getNodeType()==Node.ELEMENT\_NODE) {

if (elemuj.getNodeName().compareTo("ar") == 0) {

for (r=0; r<elemuj.getChildNodes().getLength(); r++) Node elemujabb = elemuj.getChildNodes().item(r) if (elemujabb.getNodeType()==Node.TEXT\_NODE) {

ear = Integer.parseInt(elemujabb.getTextCont

}

}

}

if (elemuj.getNodeName().compareTo("cim") == 0) {

for (r=0; r<elemuj.getChildNodes().getLength(); r++) Node elemujabb = elemuj.getChildNodes().item(r) if (elemujabb.getNodeType()==Node.TEXT\_NODE) {

ecim = elemujabb.getTextContent();

}

}

}

}

}

if (ear > minar) { System.out.println(ecim);

}

}

}

}

public void update1 (Document dom, String nev, int nov) { int p,q,r,kell=0;

int ear=0;

String ecim=null; Node elem = null;

Node root = dom.getDocumentElement();

for (p=0; p<root.getChildNodes().getLength(); p++) { elem = root.getChildNodes().item(p);

if (elem.getNodeName().compareTo("konyv") == 0) { kell = 0;

for (q=0; q<elem.getChildNodes().getLength(); q++) { Node elemuj = elem.getChildNodes().item(q);

if (elemuj.getNodeType()==Node.ELEMENT\_NODE) {

if (elemuj.getNodeName().compareTo("cim") == 0) {

for (r=0; r<elemuj.getChildNodes().getLength(); r++) Node elemujabb = elemuj.getChildNodes().item(r) if (elemujabb.getNodeType()==Node.TEXT\_NODE) {

ecim = elemujabb.getTextContent();

if (ecim.compareTo(nev) == 0) { kell=1;

}

}

}

}

}

}

if (kell==1) {

for (q=0; q<elem.getChildNodes().getLength(); q++) { Node elemuj = elem.getChildNodes().item(q);

if (elemuj.getNodeType()==Node.ELEMENT\_NODE) {

if (elemuj.getNodeName().compareTo("ar") == 0) { for (r=0; r<elemuj.getChildNodes().getLength();

Node elemujabb = elemuj.getChildNodes().ite if (elemujabb.getNodeType()==Node.TEXT\_NODE) ear = Integer.parseInt(elemujabb.getTextC

System.out.println(ear);

String ujszov = String.valueOf(ear+nov); elemujabb.setNodeValue(ujszov);

}

}

}

}

}

}

}

}

}

public void insert1 (Document dom, String nev, int ear) { Node root = dom.getDocumentElement();

Node ujelem = dom.createElement("konyv"); root.appendChild(ujelem);

Node ujelem2 = dom.createElement("cim");

ujelem2.setTextContent(nev); ujelem.appendChild(ujelem2);

Node ujelem3 = dom.createElement("ar");

ujelem3.setTextContent(String.valueOf(ear)); ujelem.appendChild(ujelem3);

root.appendChild(ujelem);

}

public void delete1 (Document dom, String nev, int ear) { int p,q,r,kell=0;

String ecim=null; Node elem = null;

Node root = dom.getDocumentElement();

for (p=0; p<root.getChildNodes().getLength(); p++) { elem = root.getChildNodes().item(p);

if (elem.getNodeName().compareTo("konyv") == 0) { kell = 0;

for (q=0; q<elem.getChildNodes().getLength(); q++) { Node elemuj = elem.getChildNodes().item(q);

if (elemuj.getNodeType()==Node.ELEMENT\_NODE) {

if (elemuj.getNodeName().compareTo("cim") == 0) {

for (r=0; r<elemuj.getChildNodes().getLength(); r++) Node elemujabb = elemuj.getChildNodes().item(r) if (elemujabb.getNodeType()==Node.TEXT\_NODE) {

ecim = elemujabb.getTextContent(); if (ecim.compareTo(nev) == 0) {

kell=1;

}

}

}

}

}

}

if (kell==1) {

for (q=0; q<elem.getChildNodes().getLength(); q++) { Node elemuj = elem.getChildNodes().item(q);

if (elemuj.getNodeType()==Node.ELEMENT\_NODE) {

if (elemuj.getNodeName().compareTo("ar") == 0) { for (r=0; r<elemuj.getChildNodes().getLength();

Node elemujabb = elemuj.getChildNodes().ite if (elemujabb.getNodeType()==Node.TEXT\_NODE) if (ear == Integer.parseInt(elemujabb.g

kell = 2;

}

}

}

}

}

}

}

if (kell == 2) { root.removeChild(elem); p--;

}

}

}

}

}