Debreceni Egyetem

Informatika Kar

MARC rekordok feldolgozása Java alkalmazás segítségével

Témavezető:
Dr. habil. Boda István
tanszékvezető egyetemi docens

Készítette:
Kovács Krisztián
Informatikus Könyvtáros

Debrecen
2010

Tartalomjegyzék

1. Bevezetés

Az egyetemi tanulmányaim folyamán az egyik könyvtári gyakorlatomat a Debreceni Egyetem Egyetemi és Nemzeti Könyvtár Informatikai központjában végeztem. Itt került előtérbe a probléma melynek megoldására készítettem el a programomat.

1.1. Debreceni Egyetem Egyetemi és Nemzetközi Könyvtár

Az Egyetemi Könyvtár gyűjtőköre 1952-től kottákkal is bővült, ettől az időponttól számíthatjuk tehát a Könyvtár kiemelkedő értéket képviselő zenei külön gyűjteményének megalapozását. A mára már mintegy százezer kötetet tartalmazó kottatár az új kiadású kották mellett - **Hankiss János** professzor áldozatos gyűjtőmunkájának köszönhetően - egy olyan, a 19. században, ill. a 20. század elején megjelentetett **régi kottagyűjteményt** is magába foglal, melyek között nagy zeneszerzők jelentős műveinek első kiadásai is megtalálhatók. A kották mellé az évtizedek folyamán (1956-tól) rendkívül értékes külföldi és hazai kiadású hanganyag-gyűjteménnyel gazdagodott a könyvtár. A gyűjtemény széles és átfogó képet nyújt a zeneirodalom egészéről, az egyes zenetörténeti korszakok, stílusirányzatok legjelentősebb képviselőiről. Évezredek zenéjével ismerkedhetünk meg itt: az antik zenétől a kortárs szerzők műveiig, tematikailag a gyűjtemény zömét kitevő komolyzenei felvételektől a szépirodalmi, archív, könnyűzenei, népzene, jazz, stb. hangdokumentumokig.

A Zeneműtár 2005 végéig a volt **Egyetemi Templomban** működött, majd 2006. januárja óta az újonnan épült **Kenézy Élettudományi Könyvtár** földszintjén nyert elhelyezést **Zeneműtár- és Médiatár** néven, ahol a gazdag zenei gyűjtemény használata kiegészül a zenei, művészeti folyóiratok helyben olvasásának s a gondosan válogatott kézikönyvtár tanulmányozásának lehetőségével, jó feltételeket teremtve a tudományos kutatómunkához is. A Zeneműtár- és Médiatár elsődleges feladatának tekinti a széleskörű zenei művelődés iránti igény kielégítését, a zenei oktatás és nevelés, valamint a zenekutatás segítését, a muzsikusok, színházi énekesek és kórustagok zeneművekkel való ellátását, de video-, DVD- és könnyűzenei felvételei által segíteni szeretné olvasóit szabadidejük értékes és hasznos eltöltésében is.

1.2. A megoldandó probléma bemutatása

A DEENK Médiatárának internetes oldalán a DVD-k listába rendezését eddig mindig kézzel végezték, az interneten eddig megjelent listát kézzel készítették. A médiatár számítógépes rendszerében Marc[1] rekordok formájában a teljes lemez állomány nyilván van tartva az összes adattal együtt, ami rendelkezésre áll. A program alap feladata annyi volt, hogy ebből a Marc rekordokat tartalmazó állományból kikeresse és listákba rendezze a különböző feltételeknek megfelelően a DVD-ket. Az alkalmazásnak cím, rendező neve és szereplők nevei alapján kellett megalkotnia a listákat. A különböző listáknak tartalmaznia kellett a dokumentumonként a címet, szereplőket, rendezőt és a dokumentum azonosítóját és ezen négy adatcsoportnak a megfelelő listán belül abc rendben kellett szerepelnie. Első gondolatban teljes kész html oldalakat készített a program a különböző rendszerezett dokumentumokról betűrendben szétbontva. Ez a hozzáállás nem volt jó, ezért a végső verzióban csak html kódrészletet hoz létre és ezt egy php motor alakítja a megfelelő formára, mivel ez jobban megfelelt a weblapkészítők elvárásainak.

Ezt kiegészítve később kértek egy RSS-es hírszolgáltatást nyújtó részt is.

[1] MARC - MAchine Readable Cataloging / Géppel olvasható katalógus

2. Fejlesztői környezet bemutatása

Egyetemi tanulmányaim során több programozási nyelvel is megismerkedtem, ezek közül a Java nyelv lett az általam legkedveltebb. A program egyik fontos eleme az, hogy a könyvtár szervergépén hetente időzítve futtatják. Ez annyit tett a számomra, hogy a legjobb választás a Java nyelv a program megírásához, mivel ez a könyvtári szervereken könnyen alkalmazható és beállítható.

A programkód elkészítéséhez NetBeans-t használtam, mivel a Java weboldala is ezt ajánlotta. Másrészről pedig ezt a fejlesztői eszközt használtam már előtte is.

2.1. Java

2.1.1. Java történet

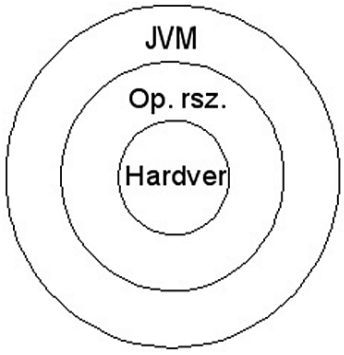
1991-ben a Sun Microsystems egy 13 főből álló fejlesztői csoportot állított fel „Green Team” fantázianévvel. A csoport feladata olyan háztartási eszközök készítése volt, amelyek beépített mikrocsippel rendelkeznek, s képesek az egymás közötti kommunikációra. Kezdetben a C++ nyelvvel próbálkoztak, de gyorsan kiderült, hogy ez a nyelv nem alkalmas a feladatra, ezért a csapat egyik tagja (név szerint James Gosling) egy új nyelvet dolgozott ki. Az új nyelv a C egy továbbfejlesztett verziója. A nyelv a tűzkeresztségben az „Oak” (Tölgy) nevet kapta, állítólag azért, mert kifejlesztője dolgozószobájának ablakából egy hatalmas tölgyre látott rá. A nyelvvel szemben támasztott alapvető elvárás volt, hogy platform független legyen, hiszen a háztartási eszközök gyártói bármikor lecserélhették mikroprocesszoraikat egy olcsóbb, jobb modellre.

A kezdeti sikereknek köszönhetően a projekt új lendületet kapott, újabb embereket vontak be a munkába, így a létszám 70 főre duzzadt, s felvette a „FirstPerson” nevet. A biztató eredmények ellenére sem sikerült nagyobb megrendelést szerezniük, így 1993 tavaszára a projekt végveszélybe került. Ekkor a vezetők (John Gage, James Gosling, Bill Joy, Patrick Naughton, Wayne Rosing és Eric Schmidt) három napra elvonultak a világ elől, s határoztak a továbbiakról. Döntésük eredménye, hogy meg kell próbálkozni az internettel!

Az grafikus böngészők megjelenésének köszönhetően egyre nagyobb népszerűségnek örvendő Internet számára tökéletesen alkalmas volt az Oak technológia, hiszen platformfüggetlenségének köszönhetően nem okozott számára gondot a hálózatba kapcsolt gépek inhomogenitása. 1995 elejére a csapat kifejlesztett egy grafikus böngészőt „Webrunner" néven. Később ez a böngésző lett az őse HotJava böngészőnek, s itt jelent meg a Java kabalafigurájaként ismert Duke figura. A Sun megpróbálta levédeni az Oak nevet, de kiderült, hogy ezt már használják egy programozási nyelv elnevezéseként, így új nevet kellett találni. A nyelv kifejlesztésének ideje alatt elfogyasztott kávék származási országának emléket állítva kapta a technológia a Java nevet. A csapat külön Internet-címet kapott, ahol lehetővé tették a források ingyenes letölthetőségét, s így bárki számára a tesztelési lehetőséget. A letöltések száma rohamosan növekedett, s 1995 novemberében már letölthető volt a nyelv béta verziójának forráskódja és fejlesztőkészlete. Néhány év alatt a Java a programozók egyik legfontosabb eszköze, a hálózat nyelve és operációs rendszere lett. Mindez azoknak a lelkes programozóknak köszönhető, akik korán felismerték a platformfüggetlenség iránti igényt, s meglátták a benne rejlő lehetőséget.

2.1.2. A Java platformfüggetlensége

Napjaink - magasabb szintű programozási nyelven írt - alkalmazói szoftverei hardverfüggetlenek. Ez kicsit szemléletesebben azt jelenti, hogy egy adott operációs rendszerre írt felhasználói programnak nem kell törődnie azzal, hogy milyen hardver van a gépünkben. Nem kell ismerettel rendelkeznie a memória vagy a háttértároló típusáról, gyártójáról. A hardverek típus- és gyártó specifikus részeit a gépen futó operációs rendszer elrejti a felhasználói programok elől, egy egységes felületet biztosítva ezáltal a programozóknak. A Java viszont ennél is tovább megy, hiszen még egy réteget húz az operációs rendszerre, így elfedve a rendszerek közötti különbségeket. Ezt a réteget Java Virtuális Gépnek (JVM[2]) nevezték el. (**Ábra 2**)



Ábra 2 Platformok rétegződése

[2] JVM – Java Virtual Machine (Java Virtuális Gép)

A JVM-nek, illetve a mögötte rejlő koncepciónak köszönhetően létrejött egy új, magasabb absztrakciós szintű platform (Java platform), amely lehetővé teszi, hogy az egyszer már megírt programunkat bármely más architektúrán futtassuk. Innen származik a Java alapfilozófiáját megfogalmazó **„Írd meg egyszer, futtasd bárhol!”** mondat. Természetesen egy platform csak akkor válhat Java platformmá, ha az adott szoftver, vagy hardver gyártója beépítette a Java programok futtatásához szükséges JVM-et.

2.1.3. A Java alkalmazásai, lehetőségei

Egy pár példa, melyből kitűnik a Java technológia[3] sokszínűsége. Az első és legfontosabb tény, hogy minden jelentősebb operációs rendszerhez ingyenesen elérhető legalább egy JVM, ennek köszönhetően *egy általunk megírt és lefordított program gond nélkül futtatható* Solarisos, Windowsos, Linuxos, AS/400-as, stb. gépeken (PC-n és más architektúrán egyaránt). Sokan a Java-t az appletek programozásaként ismerik, de a ez csak egy nagyon kis szelete a technológiának. A lehetőségeknek szinte csak a képzelet szabhat határt. Egyszerű eszközök segítségével készíthetünk párhuzamos programokat, adatbázis-kezelő alkalmazásokat (JDBC[4]), dinamikus weboldalakat (Servlet, JSP), vagy a napjainkban oly divatos XML feldolgozókat illetve web-szolgáltatásokat. Van Java-ban írt web-böngésző, web-szerver, de még operációs rendszer is.

A Java nem állt meg a személyi számítógépek által futtatható alkalmazásoknál. Létezik JVM mobiltelefonokra, PDA-kra, amelyek – bár szűkebb eszközkészlettel rendelkeznek – képesek Java programok használatára. Ugyanakkor kifejlesztettek Java-kódot futtató processzort, valamint plasztik kártyát (JavaCard) is, amelynek mérete megegyezik egy átlagos bankkártya méretével. A budapesti 56-os villamos végállomásán a vezetők gyűrűjüket egy rögzített eszköz elé tartják, hogy az regisztrálja a megérkezést. A rögzített eszköz és az elétartott gyűrű egyaránt Java kódot futtat, ennek a technológiának a segítségével végezvén el a kommunikációt. Emellett egyre több gyártó jelenti be Java-kód futtatására alkalmas háztartási eszközét is. Ebből látszik, hogy a fejlődés még nem állt meg. A technológiát egyre szélesebb körben alkalmazzák, s fogják alkalmazni a jövőben is.

[3] A Java nemcsak egy nyelv, hanem egy technológia elnevezése is egyben, amely a Java nyelvet, mint eszközt használja céljai megvalósítására
[4] JDBC - Java Database Connectivity: Java adatbázis hozzáférés támogatás

2.1.4. Eszközök és verziók

A Java-hoz is léteznek RAD[5] eszközök, amelyekkel - nevüknek megfelelően – gyorsabban készíthetjük el a kódokat, de egy ilyen fejlesztőkörnyezet használata magában rejti azt a veszélyt, hogy elveszünk a menüpontok között, s „nem látjuk a fától az erdőt”, vagyis egy fellépő hiba esetén nem tudjuk eldönteni, hogy pontosan hova is nyúljunk. A RAD eszközöknek megvan az előnyük, de ezeket csak akkor tudjuk igazán kihasználni, ha már tisztában vagyunk a részletekkel[6].

Amellett, hogy Sun meghatározta a Java nyelvi szabályainak leírását (szintaktikáját), létrehozott egy API[7]-t, amely a szoftverek által felhasználható osztályokat tartalmazza. Ezeket az osztályokat, valamint a hozzájuk tartozó segédprogramokat (fordító, futtató, stb.) adják ki JDK[8] név alatt, különböző verziószámokkal (1.1, ...,1.2.2, ..., 1.4). Ez a verziószám azonban nem felletethető meg egyértelműen a Java-platform verziószámával. Egy kiadás dátumát is feltüntető összehasonlítást mutat a Táblázat 1. Az 1.2-es platform új nevet kapott (Java 2). Ez a jelenleg használt platform, a JDK fejlődése viszont nem állt meg. Újabb és újabb osztályok kerülnek be, igazodván az újabb igényekhez. Meg kell említeni még egy betűszót, ez a JRE[9], amely egy lebutított JDK-nak fogható fel. Ez az eszköz nem tartalmaz fordító, csak futtató környezetet, amely akkor lehet hasznos, ha valakinek, például a megrendelőnknek, nincs szüksége a meglévő szoftver fejlesztésére, csak annak futtatására. Ekkor helytakarékossági és biztonsági szempontból is érdemesebb egy JRE-t telepíteni, amelynek verziószáma analóg módon növekszik a JDK verziószámával együtt.

Kiadás	Platform	JDK
1995.	1.0	1.0
1996.	1.1	1.1
	Java 2	1.2
		1.3
		1.4
		…

Táblázat 1

[5] RAD – Rapid Application Development (Gyors Alkalmazás-fejlesztés)
[6] A Sun oldaláról (www.sun.com) ingyenesen letölthető a Forte nevű fejlesztőkörnyezet (RAD)
[7] API – Application Programming Interface
[8] JDK - Java Development Kit: Java fejlesztői környezet
[9] JRE – Java Runtime Enviroment :Java Futtató Környezet

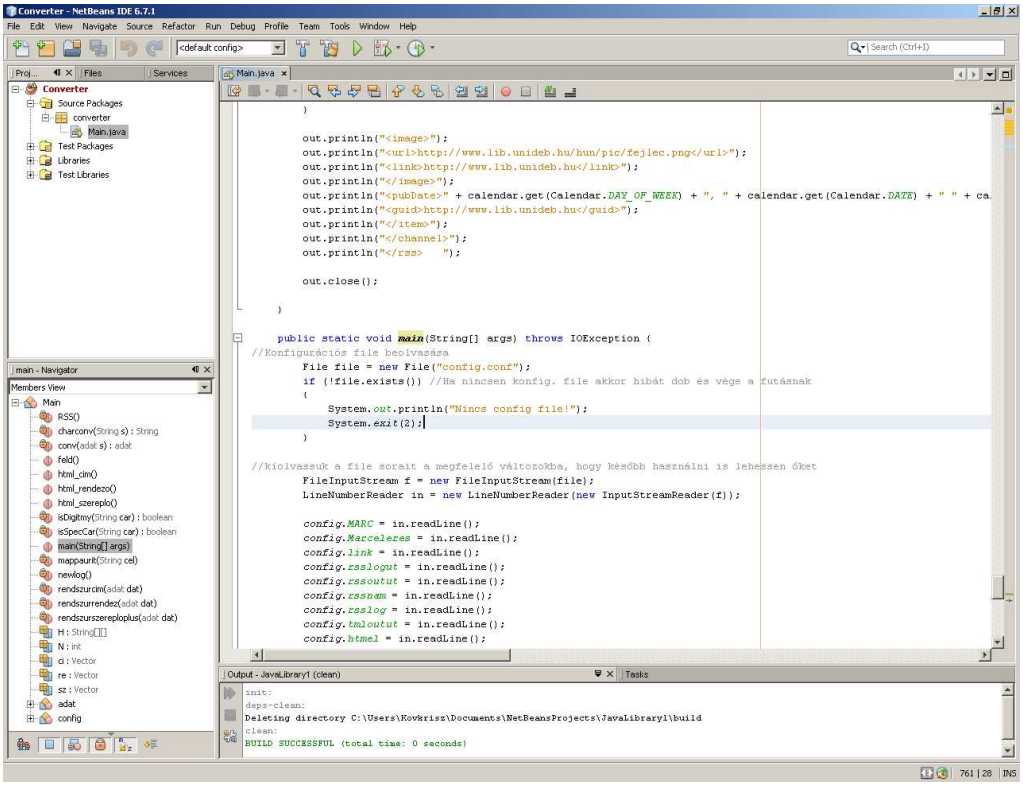
2.1.5. Objektum-orientáltság

A nyelv egyik fő tulajdonsága, az objektum-orientáltság, a programozási stílusra és a nyelv struktúrájára utal. Az OO[10] fontos szempontja, hogy a szoftvert „dolgok” (objektumok) alapján csoportosítja, nem az elvégzett feladatok a fő szempont. Ennek alapja, hogy az előbbi sokkal kevesebbet változik, mint az utóbbi, így az objektumok (az adatokat tartalmazó entitások) jobb alapot biztosítanak egy szoftverrendszer megtervezéséhez. A cél az volt, hogy nagy fejlesztési projekteket könnyebben lehessen kezelni, így csökken az elhibázott projektek száma.

[10] OO - Objektum-orientáltság

2.2. NetBeans

A NetBeans[11] egy nyílt forráskódú integrált fejlesztői környezet (IDE[12]), amely a Java nyelven alapul. A program grafikus fejlesztőfelületet kínál a különböző alkalmazások, appletek vagy akár JavaBeanek elkészítéséhez. Segítségével könnyebben, gyorsabban tudjuk fejleszteni saját programjainkat.



1. Kép NetBeans képernyőkép

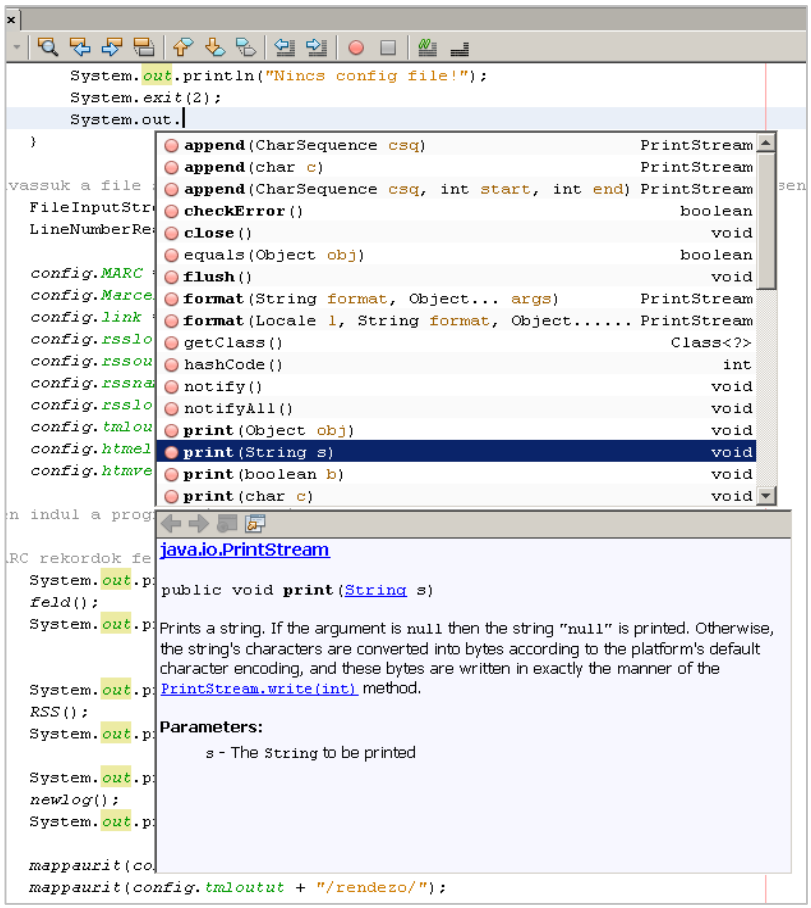
A Sun Microsystems 2000 júniusában hozta létre a NetBeans nyílt forráskódú projectet. Olyan fejlesztői környezet, amely lehetővé teszi számunkra, hogy programokat írjunk, fordítsunk, teszteljünk, hibakeresést végezzünk az alkalmazásokban. Java nyelven íródott, de bármilyen más programozási nyelvet is támogat (C/C++, PHP, Ajax, stb.). A NetBeans IDE szoftvert számos modullal bővíthetjük és futtathatjuk különböző operációs

[11] A NetBeans aktuális verziója letölthető a http://netbeans.org/ oldalról
[12] IDE - Integrated Development Environment: integrált fejlesztői környezet

rendszereken is (Linux, Windows, Mac OS X, Solaris). A termék teljesen ingyenes és semmilyen korlátozás sincs érvényben a használatára vonatkozóan. Jelenlegi legfrissebb verzió az 6.8-es, a program fejlesztésénél a 6.5-ös verziót használtam.

2.2.1. Saját tapasztalatok

Nagyon egyszerűen kezelhető fejlesztői környezet, szinte a programozó kezébe adja a különböző programozási eszközöket. Gyorsan létre lehet vele hozni felületeket, kereteket, panelokat. Igaz ennek a programnak a fejlesztésénél ezeket a lehetőségeket nem használtam ki, mivel egy szerveren futó programról van szó. A kódok megírásánál a parancsokat felajánlja és bőséges információt ad mindegyikről. A csomagok, fájlok, módszerek, eljárások a bal oldali ablakban teljes mértékben átláthatóak, hozzáférhetőek.



2. Kép NetBeans parancslista és -leírás

A programozóknak nagy segítséget jelent, ha minden felhasználható eszközről és parancsról talál információt egy gyorsan elérhető és kereshető helyen. A NetBeans Help-je bőségesen kielégíti a kívánalmakat, de érdemes hozzá az angol tudás, mivel angol nyelven írták és mint a programhoz úgy a helphez sincs magyar fordítás.

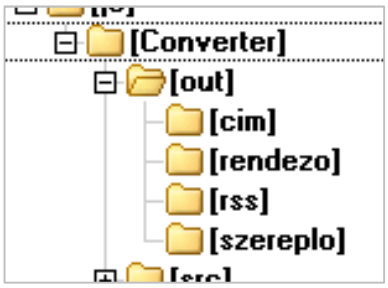
Véleményem szerint napjaink egyik legjobb programfejlesztő eszköze. Megvan benne valószínűleg az összes eszköz a különböző területekre szánt programok fejlesztéséhez.

3. Fejlesztői dokumentáció

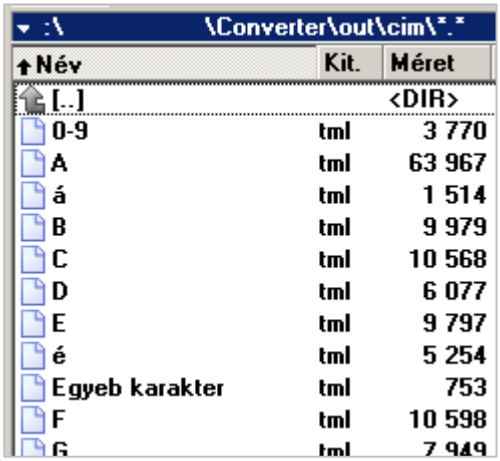
3.1. A program feladatai

A programnak egy fő és egy mellékfeladata van. Ezen feladatok a programban – amit majd később részletesen is leírok – lineárisan, az-az egymás után hajtódnak végre. A projektvezető kérése volt a minél egyszerűbb kialakítás és a teljes szöveges felületen való kezelhetőség.

Fő feladata, hogy elkészítse a MARC rekordokból álló szöveges állományból a cím, rendező és szereplő nevek alapján betűrendbe rendezett elemeket a médiatár weblapja számára. Mely elemeket a web-programozóik könnyen tudnak kezelni. Ezért lesz a program végeredménye egy három könyvtárból (a mellékfeladatnak is van egy munkamappája, ezért négy mappa készül) és a könyvtárakban sok fájlból álló adatrendszer (**3. Kép**). A mappákban tml kiterjesztésű filék vannak (**4. Kép**). Azért kapta ezt a kiterjesztést, mert a tartalmuk html kódrészletek, de nem teljes és magában nem értelmes html kód, saját ötlet alapján csonkoltam a html kiterjesztést és így lett tml. (A fájlok pontos tartalmáról és a felhasznált adatokról később)



3. Kép Mapparendszer



4. Kép Minta fájlok

A mellékfeladata a programnak annyi, hogy az újonnan a rekordok közé kerülő lemezek címéből készít egy RSS fájlt, amit a web-szerver el tud küldeni a hírszolgáltatást

igénylőknek. Ezt a feladatot egy log fájl segítségével készíti el, mely minden egyes lefutáskor frissül és az rss mappában található. Természetesen ide képződik le a kész RSS fájl is.

3.2. Felhasznált külső csomag

Miért is jó nekem, hogy előre megírt API-t használok? Elsősorban az a nagy előnye van, hogy rengeteg dolog kivan dolgozva, a hibalehetőségek le vannak kezelve és ezekkel már nem kell foglalkoznom, csak fel kell használnom a saját programomban.

3.2.1. MARC4J[13]

A MARC4J célja, hogy egy könnyen kezelhető API-t nyújtson MARC[14] és MARCXML[15]-t használóknak Javában. A MARCXML veszteségnélküli átalakítást tesz lehetővé MARC és XML között, de más formátumokat is tud kezelni.
A MARC4J csomag tartalma:

- Egy könnyen kezelhető interfész amellyel nagy rekordkészleteket is egyszerűen kezelhetünk.
- Olvasó és író eszközök MARC és MARCXML-hez.
- Előre elkészített eljárásalapú MARCXML-t használó XSLT stíluslapok.
- Egy MARC rekord tárgymodell (mint DOM az XML-nél) memóriában való szerkesztése.
- Adat átalakítási megoldások: MARC-8 ANSEL, ISO5426 vagy ISO6937, támogatott még UCS/Unicode-ba és természetesen visszafelé is.
- Egy előolvasó mely a feljegyzésekben található hibákat kezelni tudja.
- Implementációs XML támogatás JAXP és SAX2-vel, nagy teljesítményű XML interfész.
- Támogatás MARC és MARCXML közötti átalakításokhoz.

[13] Az API verziói letölthetőek a http://marc4j.tigris.org webhelyről
[14] MARC - MAchine Readable Cataloging / Géppel olvasható katalógus
[15] MARCXML - Congress' Network Development és MARC Standards Office fejlesztése; rugalmasan ötvözi a MARC rekordok adattárolási módszereit az XML Shema lehetőségeivel

- Nagyfokú integráció JAXP, DOM és SAX2 interfészekhez.
- Könnyen integrálható más XML-ekbe a DOMOT, XOM, JDOM vagy DOM4J interfészeket használva.
- Parancssori utasításkészlet MARC és MARCXML átalakításokhoz.
- Javadoc dokumentáció.

3.3. A programot alkotó függvények

A program és részeinek könnyebb megértése érdekében a programösszetevőkön a meghívásuk és futásuk sorrendjében fogok végighaladni. Az alfejezetek címei a programban található metódusok nevei, és a számozás a meghívások egymásba ágyazottságát hívatott szemléltetni.

3.3.1. Main()

A függvény elején a program futásához szükséges adatokat olvastatom be egy külső fájlból (config.conf).

A config fájl

A filt a program az indítási mappában keresi alaphelyzetben.

config.conf

```
1  dvd.ok.marc
2  D:\j\Converter\
3  http://webpac.lib.unideb.hu:8082/WebPac/CorvinaWeb?action=cclfind&resultview=longlong&ccltext=idno+
4  D:\j\Converter\out\rss\
5  D:\j\Converter\out\rss\
6  rss.xml
7  rss.log
8  D:\j\Converter\out\
9
10 .tml
```

5. Kép A config.conf tartalma

A fájl soronkénti áttekintése és a lehetséges beállítások megismerése:

1. A feldolgozandó Marc rekord neve
2. A feldolgozandó Marc rekord elérési útja

3. Az internetes elérés alap linkje (erről majd később részletesen írok a program további részletezése folyamán, hogy miért szükséges)
4. Az RSS log-jának elérési útja
5. A kész RSS helye (ide kerül az eredmény)
6. Az RSS fájl neve
7. Az RSS log-jának fájl neve
8. A kimeneti fájlok mappáinak elérési útja (Ezen a könyvtáron belül kell lennie a "cim","rendezo","szereplo" könyvtáraknak, ezekbe pakolja a program a kész állományokat)
9. -A kimeneti fájlk nevének előtagja (Nem szükséges)
10. -A kimeneti fájlk kiterjesztése (Ez sem szükséges, de érdemes megadni)

A program a működés biztosítása érdekében a fájl hiányában a program hibaüzenettel leáll és nem fut bele a bonyolult feladatok elvégzésébe, mivel ezeket csak a megfelelő beállításokkal tudja elvégezni. A beállítások hiányában akár végtelenciklusba is futhatna.

A következő kis rész megnyitja és kiolvassa a fájl tartalmát, az-az a beállításokat és információkat a feldolgozáshoz.

```
754
755     public static void main(String[] args) throws IOException {
756 //Konfigurációs file beolvasása
757         File file = new File("config.conf");
758         if (!file.exists()) //Ha nincsen konfig. file akkor hibát dob és vége a futásnak
759         {
760             System.out.println("Nincs config file!");
761             System.exit(2);
762         }
```

6. Kép A Main eleje

A feldolgozás soronként halad a config fájlon. Az adatsorokat a globálisan létrehozott config nevű adatcsoportba helyezi, ahonnan a program ismételten elérheti az éppen kellő információkat.

```
62      public static class config {
63
64          static String MARC;
65          static String Marceleres;
66          static String link;
67          static String rssnam;
68          static String rsslog;
69          static String rssoutut;
70          static String rsslogut;
71          static String tmloutut;
72          static String htmel;
73          static String htmveg;
74      }
```

7. Kép Config változók létrehozása

A main függvény további részében sorra meghívódnak a program feldolgozó rész egységei egymás után. A futás figyelhetősége érdekében az egyes részek meghívása előtt és után az elvégzett részről rövid szöveges üzeneteket íratok ki a képernyőre.

```
782         System.out.println("feldolgozas...");
783         feld();
784         System.out.println("Feldolgozás OK");
785
786
787         System.out.println("RSS képzés...");
788         RSS();
789         System.out.println("RSS OK");
790
791         System.out.println("RSSLOG kiirása...");
792         newlog();
793         System.out.println("RSSLOG kiirása OK");
794
795         mappaurit(config.tmloutut + "/cim/");
796         mappaurit(config.tmloutut + "/rendezo/");
797         mappaurit(config.tmloutut + "/szereplo/");
798
799         System.out.println("TML filek képzés...");
800         System.out.println("    cím szerint...");
801         html_cim();
802         System.out.println("    rendező szerint...");
803         html_rendezo();
804         System.out.println("    szereplő szerint...");
805         html_szereplo();
806         System.out.println("TML-ek kiirása OK");
807         System.out.println(".........END.........");
808
809         in.close();
```

8. Kép Hívások

Az első meghívott programrész a *feld()*, amely rész az adatok feldolgozásáért felelős. Ezt követi az *RSS()* ami a hírszolgáltatást készíti el. Következő lépés az RSS loggolás mely a *newlog()* ….. segítségével készül el. Végezetül csak a rendezett adatok fájlokba írása marad melyet a *html_cim()*, *html_rendezo()* és a *html_szereplo()* végez, de még előtte elő kell készíteni a mappákat nekik, törölni kell a tartalmukat, ezt a *mappaurit()* végzi el.

Lássuk ezeket részletesen is.

3.3.1.1. feld()

Ebben a részben sok programkódot fogok bemutatni, ezért csak darabokban jelenítem meg és magyarázom el az adott részt.[16]

A *feld()* metódus lényegében annyit csinál, hogy megnyitja a rekordokat tartalmazó fájlt és a MARC4J csomag eszközeivel az egyes rekordelemek adatai közül a megfelelőeket kimásolja és meghívja a további programrészeket.

```
324
325         //A FELDOLGOZÓ EGYSÉG
326         public static void feld() throws FileNotFoundException {
327             //Akonfigban található címről megpróbálja a program betölteni a marc rekordok
328             File file = new File(config.MARC);
329             if (!file.exists()) //ha nincs meg a file hibaüzenetet ad és kilép
330             {
331                 System.out.println("Nincs meg a MARC file!");
332                 System.exit(2);
333             }
334
335             //Betöltöm a marc rekordokat tartalmazó filet
336             InputStream in = new FileInputStream(file);
337             MarcReader reader = new MarcStreamReader(in);
338
```

9. Kép *feld()* első rész

A kódrészlet eleje egy egyszerű fájlmegnyitás olvasásra. Természetesen a hiányzó fájl okozta hibákat rögtön le is kezeltem, nehogy problémákat okozzon. A program érdekes része a látható kódrészlet utolsó soránál kezdődik. Itt lehet látni először a MARC4J csomag felhasználását. Ha megnézzük a kódot lehet látni, hogy nem sorokkal dolgozik, hanem teljes rekordokkal, de ezt a későbbiek folyamán még megfigyelhetjük.

[16] A teljes programkód egyben megtalálható a mellékletben.

```
339         while (reader.hasNext()) //a MARC4J csomagot használva feldolgozom az adatokat
340         {
341             adat dat = new adat();
342             Record record = reader.next();
343 
344             if (record.find("245") != null) //megkeresem a rekordban a címet és letárolom
345             {
346                 DataField cim = (DataField) record.getVariableField("245");
347                 if (cim != null) {
348                     dat.cim = cim.toString();
349                 }
350             }
351             if (record.find("100") != null) // megkeresem benne a rendezőt és letárolom
352             {
353                 DataField rendezo = (DataField) record.getVariableField("100");
354                 if (rendezo != null) {
355                     dat.rendezo = rendezo.toString();
356                 }
```

10. Kép *feld()* második rész

A metódus végéig egy *while* ciklus fut, ezzel a megoldással lehet a legkönnyebben megoldani, hogy addig fusson, ameddig van még elem a fájlban. A *reaider.hasNext()* utasítás mindaddig igaz értéket ad vissza ameddig van következő elemünk. Ez is az importált csomagban van kidolgozva.

Létrehozok egy *dat* nevű változót amelynek a tartalmi elemeit már előzőleg globálisan definiáltam (11. Kép). Látható, hogy öt *string* típusú változót tartalmaz. Ezekbe a változókba gyűjti ki a program a keresett információkat.

```
54      public static class adat {
55 
56          String cim;
57          String rendezo;
58          String szereplo;
59          String ID;
60          String sszam;
61      }
```

11. Kép A globális *adat* változó tartalma

A következő sor egy *Record* típusú, *record* nevű változóba beleteszi a következő rekordot és csak ezt az egy rekordot kezeli a továbbiakban. Ezek után következik öt *if()* függvény, amelyek kikeresik a megfelelő adatokat a rekordból. Mind megegyezik, csak a változók másak, ezért csak az elsőt vezetem végig.

A *record.find()* függvény megkeresi a zárójelek között megadott azonosítójú adatelemet a rekordban és visszatér a tartalmával. Ha nem talál ilyen azonosítót, akkor *null* értéket ad vissza, ekkor nem teljesül a függvény feltétele és nem fut le. Amikor megtalálta az adatrészt azt átrakja egy változóba. A MARC4J csomag fildeket használ az adatok átadásánál; mielőtt bár mit is csinálunk az adatokkal kel még egy utolsó ellenőrzés, hogy nem hagyták e üresen az adatmezőt a rekord készítésekor. Amennyiben üres elemet találunk a későbbi problémák elkerülése miatt nem hajtunk végre semmit, a megfelelő adat csoport megtartja a kezdeti tartalmát, az-az a null értéket. Ha nem üres a mező stringé alakítás után belekerül a *dat* adatcsoport megfelelő változójába. Ez a folyamat megy végig mind az öt *if* függvényben csak más-más adatokkal. Az utolsó adatcsoport keresésének a végén akad egy kis különbség; a sorszám nem stringként kerül átadásra, hanem a MARC4J csomag egyik saját típusaként, ami a *Data()*.

```
378                dat = conv(dat);   //elküldöm a letárolt adatokat a convertáló fügvénynek
379
380
381                if (dat.cim != null) {
382                    rendszurcim(dat);
383                    if (dat.rendezo != null) {
384                        rendszurrendez(dat);
385                    }
386                    if (dat.szereplo != null) {
387                        rendszurszereploplus(dat);
388                    }
389                }
```

12. Kép *feld()* harmadik rész

A *while* ciklus végét láthatjuk a 12. Kép-en. Az első sor ami látszik egy *conv()* nevű rész amit egy kicsit később fogok részletezni. Látható, hogy megkapja a dat tartalmát változóként és a visszatérési értéke is a *dat* változóba kerülnek.

Hogy ne keljen ehhez a kis részlethez visszatérni, elmagyarázom a kódrészlet végét is. Mint látható a kódrész vége három egymásba ágyazott *if*-ből áll. A függvények, ha a megadott változók tartalmaznak adatot akkor meghívják a megfelelő rendező beszúrást végző metódusokat. Az adatokon itt végződnek utoljára olyan ellenőrzések, amelyek adatok létezésére vonatkoznak, ezek a beszúrásos rendezés miatt lényegesek. Ha például nincsen cím adatunk, akkor egyik meghívás sem fut le, de ha címűnk van de rendezőnk, vagy szereplőink nincsenek, akkor az ennek megfelelő részek nem futnak le, hogy ne legyen üres elemekkel feltöltött listánk.

3.3.1.1.1. conv()

A dolgozatírás előtt amint a program kódot tekintettem át úgy éreztem a programnak ez a leg bonyolultabb, de lehet, hogy csak a legáttekinthetetlenebb része.

```
134     private static adat conv(adat s) {
135
136         StringBuffer sb = new StringBuffer();
137
138         int i = 0;
139         if (s.cim.indexOf("245", i) != -1) {
140             i = i + 8;
141             int j = s.cim.indexOf("$", i);
142             sb.append(s.cim.substring(i, j));
143             i = j + 2;
144
145         }
146         s.cim = charconv(sb.toString());
147
148
149         StringBuffer sb2 = new StringBuffer();
150
151         if (s.rendezo == null) {
152             sb2.append("");
153         } else {
154             int k = 0;
155             if (s.rendezo.indexOf("100", k) != -1) {
156                 k = k + 8;
157                 sb2.append(s.rendezo.substring(k, s.rendezo.length()));
158             }
159         }
160         s.rendezo = charconv(sb2.toString());
```

13. Kép *conv()* első rész

A rekordokból kiolvasott adatelemek még tartalmaznak a program céljának tekintetében felesleges és hátrányos elemeket. Például az átvett adatelemekben még benne van az adat azonosítókódja, és egyéb felesleges karakterek. Sajnos vannak hibái az általam használt MARC4J verzió karakterkód átalakító részének, ezért erre is megoldást kellett találni. Erről részletesen később. A *conv()* metódus is több részre osztható, mivel külön-kölün elkülönítve dolgozik az egyes adatelemekkel. Az első adatelem a cím amin módosításokat végez.

A címen csak annyit kell változtatni, hogy az elejéről eltávolítom az adatazonosítót, a végéről pedig a „$” jelet, ami az adatelem végét hivatott jelölni a Marc rekordban. Ezt egy StringBuffer segítségével oldom meg. Megvizsgálom, hogy az *s.cim* –ben egyáltalán benne van e a kód, ha megtalálom akkor megjegyzem a végét egy változóban, majd megkeresem a

záró karaktert és annak a helyét is rögzítem egy másik változóban. A két változó segítségével a *substring(i,j)* paranccsal kiemelem a karaktereket a két változóval megjelölt részről a *StringBufferbe*. Újra vizsgálom a stringet a második változó értékétől, hogy nincs e még egy cím benne, amit ki kellene emelni. Amennyiben sikeresen kiléptünk a ciklusból a StringBuffert újra stringgé alakítjuk a *toString* segítségével és az értéket közvetlenül a *charconv()* parancsnak adjuk át. A következő fejezet fog erről a algoritmusról szólni. visszatérési értékét az eredeti adat helyére töltöm, hogy ne legyen olyan sok változó és kevesebb memóriát használjon. Igaz ez egy két rekorddal való munkánál nem sokat jelent, de amikor már több ezres a tételek száma meglátszanak az ilyen kis apróságok is.

A következő részben a program a *rendezo* változót rendezi hasonló módon mint ahogy azt előzőleg leírtam. Annyi a különbség csak, hogy itt nincs „$” a végén, így a string vége lesz az átemelendő karakterek vége, így csak az elejét csonkoljuk. Végül ezt a változót is átadom a *charconv()* –nak.

```
161             StringBuffer sb3 = new StringBuffer();
162             StringBuffer sb31 = new StringBuffer();
163             StringBuffer sb32 = new StringBuffer();
164             StringBuffer sb33 = new StringBuffer();
165
166             if (s.szereplo == null) {
167                 sb3.append("");
168             } else {
169                 int l = 0;
170                 if (s.szereplo.indexOf("511", l) != -1) {
171                     l = l + 8;
172                     sb3.append(s.szereplo.substring(l, s.szereplo.length()));
173                 }
174                 if (sb3.indexOf("Szereposztâas") != -1) {
175                     l = sb3.indexOf(":");
176                     sb31.append(sb3.substring(l + 1, sb3.length()));
177                 } else {
178                     sb31.append(sb3);
179                 }
180                 if (sb31.indexOf("Elîoadâok:") != -1) {
181                     l = sb31.indexOf(":");
182                     sb32.append(sb3.substring(l + 1, sb3.length()));
183                 } else {
184                     sb32.append(sb31);
185                 }
186                 if (sb32.indexOf("[et al.]") != -1) {
187                     l = sb32.indexOf("[et al.]");
188                     sb32.delete(l, l + 8);
189                 }
190             }
191             s.szereplo = charconv(sb32.toString());
```

14. Kép *conv()* második rész

A következő kódrészletben, amit láthatunk, folytatódik az adatok feldolgozása. A szereplők adatcsoport a médiatár Marc rekordjaiban egy kicsit bonyolult összetételű, ezért ezt kezelni kell. A Marc rekordok 511-es adatelemében – ez a kódja a szereplőknek a Marcban – megtalálhatóak különböző kulcsszavak, amikre nekünk nincsen szükségünk. Ezek a szereposztás, előadók és az „et al.” kifejezés. Ezeket egymás után a már bemutatott módszerrel megkeresem és eltávolítom a szöveges változóból. Így nem maradnak meg csak a nevek, mivel másra nincs szükség. Végül ez a változó is a *charconv()* metódushoz kerül.

```
193             StringBuffer sb4 = new StringBuffer();
194
195             if (s.ID == null) {
196                 sb4.append("");
197             } else {
198                 int m = 0;
199                 if (s.ID.indexOf("001", m) != -1) {
200                     m = m + 4;
201                     sb4.append(s.ID.substring(m, s.ID.length()));
202                 }
203             }
204             s.ID = sb4.toString();
205
206             return s;
207         }
```

15. Kép *conv()* harmadik rész

Végül a már ismert módon elvégzi az algoritmus az ID-n is a módosításokat, de ezt nem kell a *charconv()* használatával átvizsgálni nem megfelelő karakterek után, mivel ez biztosan nem tartalmaz ékezetes karaktereket.

Az utolsó sorban lehet látni, hogy visszaadom az *s*-t. A kódból lehet látni hogy az s változó több másik változót foglal magába. Ez a már előzőleg elmondott saját típus létrehozása által lehetséges. Ugyancsak ez teszi lehetővé, hogy szimpla visszatérési értékként több változó értéket adjunk vissza. Ezzel a módszerrel nem kell a globális változók újra és újra törlésével és felülírásával foglalkozni, ezzel egy kicsit egyszerűbbé lehetett tenni a program írását.

3.3.1.1.1.1. charconv()

A *charconv* egy rekurzívan működő függvény, amely a szöveges típusú változókban a MARC4J hibája által létrejött karakterhibákat javítom. Mielőtt a kódot részletesen ismertetném először két globális változót kell ismertetnem. Ezeket használom a karakterek cseréjéhez.

```
32  //Vizsgált, cserélt elemek száma a convert f.-hez.
33      static int N = 16;
34  //Vizsgált értékek a carconv-hoz
35      static String H[][] = {{"âa", "âe", "âo", "âu", "èo", "îo", "èu", "îu", "âi", "$a",
36                              {"á", "é", "ó", "ú", "ö", "ő", "ü", "ű", "í", " ", " ", " ",
```

16. Kép Globálisan létrehozott változók a karaktercserélő algoritmushoz

A globálisan létrehozott *N* változó adja meg a globális *H* tömb elemeinek a számát, ennek a lényegessége a 17. Képen látható kód leírásából egyértelműen megérthető lesz. A *H* tömb tartalmazza az első sorában azokat a karakter kombinációkat, amiket ki szeretnék cserélni a szöveges elemekben, a második sora azokat az elemeket tartalmazza, amire cserélni kell őket. Ezek természetesen párosítva vannak nem véletlen a sorrend.

```
105     //A stringek karaktereit konvertáló rekurzív függvény
106     private static String charconv(String s) {
107         String out = null;
108         String a = null;
109         StringBuffer sb = new StringBuffer();
110         if (s != null) {
111             sb.append(s);
112             int x = sb.length();
113             for (int i = 0; i < x - 1; i++) {
114                 a = sb.substring(i, i + 2);
115                 for (int j = 0; j < N; j++) {
116                     if (H[0][j].equalsIgnoreCase(a)) {
117                         sb.delete(i, i + 2);
118                         sb.insert(i, H[1][j]);
119                         x--;
120                     }
121                 }
122             }
123             out = sb.toString();
124         }
125         return out;
126     }
```

17. Kép A *charconv()* függvény

Az algoritmus megkap egy stringet, megvizsgálja, hogy nem üres e. Amennyiben nem az, az egész belekerül egy StringBufferbe a könnyű kezelhetőség érdekében, és hogy ne keljen állandóan a változókat cserélgetni. Egy *for* ciklussal karakterenként végighaladunk az egészen és közben a karakterpárokat egy másik *for* és egy *if* ciklus segítségével végigellenőrzöm, hogy nincs e egyezés a már előre létrehozott és a megfelelő karakterpárokkal feltöltött globális H tömb első sorának elemeivel. Amennyiben egyezést találok törlöm a két karaktert a StringBufferből és a helyükre a H tömb második sorából a megfelelő elemet szúrom be a helyükre. Mivel így csökken egy karakterrel a StringBuffer hossza ezért az elején meghatározott hosszt csökkenteni kell, hogy ne okozzon problémát a túlfutás. A végén vissza alakítjuk egy string típusú változóba és visszatérési értéknek adjuk ezt meg.

A program futása visszatér a meghívó osztályhoz és jön a következő rész.

3.3.1.1.2. rendszurcim()

Mielőtt erre a részre térnék előtte, a megértést megkönnyítendő pár sort kell írnom három globálisan létrehozott vektor típusú változóról.

```
73      // A használt vektorok létrehozása
74      static Vector ci = new Vector(); //Cím szerint rendezett vektor
75      static Vector re = new Vector(); //Rendező szerint rendezett vektor
76      static Vector sz = new Vector(); //Szereplők szerint rendezett vektor
77
```

18. Kép Vektorok létrehozása

A program tervezése alatt több megoldáson is gondolkoztam, hogy mivel lehetne a legkönnyebben megoldani azt a problémát, hogy könnyen és gyorsan lehessen az elemeket rendezni. A legjobbnak a vektorok alkalmazása tűnt. A későbbiekben látható lesz a felhasznált módszer is. A három vektorban lesznek eltárolva a megfelelően rendezett elemek. Majd később ezekből a vektorokból olvassa ki az egyik eljárás a fájlok tartalmának létrehozásánál felhasznált adatokat.

```
206         // Rendezetten beszurja a megadott adatokat egy vektorba (az elem helyét lineáris ke
207         private static void rendszurcim(adat dat) {
208             int i = 0;
209             if (ci.isEmpty()) {
210                 ci.addElement(dat);
211             } else {
212                 Iterator it = ci.iterator();
213                 while (it.hasNext()) {
214                     adat a = (adat) it.next();
215                     if (dat.cim.compareToIgnoreCase(a.cim.toString()) >= 1) {
216                         i++;
217                     }
218                 }
219                 ci.insertElementAt(dat, i);
220             }
221         }
```

19. Kép A rendezőbeszúrást végrehajtó kód (cím)

A ciklus megkapja az előzőleg a Marc rekordból kigyűjtött, használható formába öntött adatokat. létrehozok egy számlálót, ami majd a vektorban található elemeket fogja számlálni. Megnézi, hogy van-e elem a vektorban, ha nincs beleteszi a megkapott adatokat, és ez lesz az első behelyezett elem a vektorban. Amennyiben talál benne elemet létrehoz egy iterátort, amelynek segítségével gyorsan meg lehet találni a beszúrandó elem helyét. Egy *while* ciklust futtatok addig, ameddig van következő eleme a vektornak és közben minden egyes elemet vizsgálok, a *compareToIgnoreCase()* segítségével, hogy az elem cím változója kisseb vagy nagyobb e mint a beszúrandó elem cím változója. Ha a beszúrandó elem cím változója megegyezik vagy nagyobb, mint a vektor aktuális eleme akkor megtaláltuk a beszúrás helyét, amelyet a számláló jelöl nekünk, mivel ameddig nem teljesültek a feltételek addig ahányszor léptettük a keresést mindig növeltük az értékét egyel. Amint végigfutott a *while* ciklus az adatok beszúródnak a vektorba a jelölt helyre, pontosabban az index elé..

Most a dolgozatírás közben találtam egy kis hibát a programkódban, amely a program futási idejére van hatással. Amint megtaláltuk, a beszúrandó elem helyét utána felesleges tovább futni a vektoron. Mivel ha egy több ezer elemről beszélünk és az első helyre kell berakni az elemet, akkor felesleges végigfutni az egészen. Ezt a problémát egy megszakítással lehetne kezelni, de az eredeti, és használt programkód bemutatása a cél.

Ez a módszer egy rendezett vektort eredményez, és nem kell utólag a rendezéssel bajlódni. Ez a megoldás sokkal energiatakarékosabb, mint az utólagos rendezés.

3.3.1.1.3. rendszurrendez()

A rendezőket a megfelelő vektorba rendező algoritmus teljesen megegyezik a címeket kezelő algoritmussal. Ezért nem írom le ismét a kódot részletesen, csak láthatóvá teszem a kódot (20. Kép).

```
223         //rendező beszúrás a rendezők vektorba
224         private static void rendszurrendez(adat dat) {
225             int o = 0;
226             if (re.isEmpty()) {
227                 re.addElement(dat);
228             } else {
229                 Iterator it = re.iterator();
230                 while (it.hasNext()) {
231                     adat a = (adat) it.next();
232                     if (dat.rendezo.compareToIgnoreCase(a.rendezo.toString()) > 0) {
233                         o++;
234                     }
235                 }
236                 re.insertElementAt(dat, o);
237             }
238         }
```

20. Kép A rendezőbeszúrást végrehajtó kód (rendező)

3.3.1.1.4. rendszurszereploplus()

Ez a kód a szereplőket rendezi a vektorba. Mivel egy-egy Marc rekordhoz több szereplő is tartozik és az összes nevet rekordonként egy stringben tárolom, ez egy kicsit bonyolultabb, mint az eddigiek.

```
240     //rendező beszúrás a szereplák közé vektorba
241     private static void rendszurszereploplus(adat dat) {
242         int t = 0;
243         for (int i = 0; i < dat.szereplo.length() - 1; i++) //A szerepltömb méretét meghatár
244         {
245             String a = dat.szereplo.substring(i, i + 2);      //Atadom az a stringnek a kara
246             if (a.equals(";") || a.equals(",")) {
247                 t++;         // Összehasonlítom a keresett karakterekkel, ha valamelyiket me
248             }
249         }
250         t++;                                              //A tömb méretének egyel nagyobbnak
251         String szerepl[] = new String[t];                   //A szereplő tömb létrehozása
```

21. Kép A rendezőbeszúrást végrehajtó kód (szereplő) 1

Az első *for* ciklus, ami a metódusban található meghatározza, hogy hány szereplőt tartalmaz a string. Végighalad a szöveges állományon és megvizsgálja a karaktereket, hogy nem elválasztó karakterek e. Amennyiben talál egy elválasztó karakter lépteti a számlálót és fut tovább. Amikor végigért a stringen még meg kel egyel növelni az elemek számát, mivel az utolsó elválasztó karakter után még áll egy név. Az elemek számával létrehozok egy tömböt.

```
252     if (t == 0) {
253         szerepl[0] = dat.szereplo;
254     } else {
255         int k = 0;
256         for (int i = 0; i <= t; i++) //A szereplők szétszedése a tömbbe
257         {
258             int j = dat.szereplo.indexOf(";", i);          //A j-be belerakom a ";"
259             if (j != -1) //Ha nincs a karaktersorban ";" a fügvény -1 értéket ad vis
260             {                                              //Amennyiben van benne:
261                 szerepl[k] = dat.szereplo.substring(i, j); //A szerepl halmaz k-adik
262                 k++;                                       //Növelem a k-t hogy a köv.
263                 i = j + 2;                                    //i-nek átadom a j+1 ér
264             } else //Ha nincs benne ";" akkor ez jön:
265             {
266                 j = dat.szereplo.indexOf(",", i);          //Megkeresem a "," karakt
267                 if (j == -1) {
268                     break;                          //A biztonság kedvéért megnézem v
269                 }                                                          //Inne
270                 szerepl[k] = dat.szereplo.substring(i, j); //A szerepl halmaz k-adik
271                 k++;
272                 i = j + 2;
273             }
274
275             for (int in = 0; in < t; in++) {
276                 StringBuffer sb = new StringBuffer();
277                 sb.append(szerepl[in].toString());
278                 if (sb.substring(0, 1).equals(" ")) {
279                     sb.delete(0, 1);
280                     char f = sb.charAt(0);
281
282                     szerepl[in] = sb.toString();
283                 }
284             }
285         }
286     }
```

22. Kép A rendezőbeszúrást végrehajtó kód (szereplő) 2

Ha nincs csak egy elem, amit a tömbbe kell rakni, akkor ezt megteszi az első *if* és jön a vektorba helyezés, amit a következő kép kódján lehet majd látni. Több elem esetén jön egy for ciklus, amelyben megtörténik a nevek különbontása. A ciklus addig fut, ameddig meg nem találtuk az utolsó elemet is és bele nem helyeztük. Egy változóba belehelyezem az első elválasztó karakter indexét. Ha már nincs több érték, vagy egyáltalán nincs benne ilyen

elválasztó karakter, akkor elkezdem keresni a másodikat. Találatok esetén a program belerakja a szereplő tömb azon helyére a string részt, amely a megkeresett elválasztó karakterek között volt. Végül az utolsó *for* ciklusban eltávolításra kerül a tömbben lévő nevek elejéről a szóköz, ha van.

```
288     //A szereplők vektor tényleges feltöltése:
289     for (int j = 0; j < t; j++) //Annyiszor kell ismételni a futását egy alaprekordra
290     {
291         int i = 0;
292         dat.szereplo = szerepl[j];                          //Beviszem a módosított
293         if (dat.szereplo == null) {
294             dat.szereplo = "";            //Ha a dat.szereplo nem tartalmazna semmit,
295         }
296         if (sz.isEmpty()) {
297             sz.addElement(dat);               //Ha üres a vektorom, akkor beszúrom a
298         } else //egyébként
299         {
300             Iterator it = sz.iterator();                    //Iterátorképzés
301             while (it.hasNext()) //Ez a ciklus keresi meg a megfelelő elemet ami elé
302             {
303                 adat a = (adat) it.next();
304                 if (dat.szereplo.compareToIgnoreCase(a.szereplo.toString()) >= 1) {
305                     i++;
306                 }
307             }
308             sz.insertElementAt(dat, i);                     // Beillesztem a megfelel
309         }
310     }
```

23. Kép A rendezőbeszúrást végrehajtó kód (szereplő) 3

Az eddigi kódrész előkészítette az elemeket a vektorba való beszúráshoz. A 23. képen látható *for* ciklus végzi el a beszúrást. A ciklus addig ismétli magát, ahány elem van a tömbben, mivel minden elemet a vektorba kell pakolni. A tömbből betöltöm az aktuális *dat.szereplo* változóba az egyik nevet és ezt ismétlem a tömb végéig. Ez azért fontos, hogy a vektorban megtalálhatóak legyenek ugyanazok az adatok minden név mellett, amit behelyezünk a vektorba. Meg kell vizsgálni, hogy az éppen aktuális elem a tömbben nem üres elem e, mert ez később gondot okozhat, ha üres elem kicserélem egy szóközre.

Ettől a ponttól kezdve a metódus ugyan azt végzi mint az előzőekben ismertetett *rendszurcim()* és *rendszurrendez()*. Ezt a részt nem írom le ismét.

Ezzel a résszel befejeződött az adatok vektorokba helyezése, a program innentől csak a vektorokban található adatokkal foglalkozik. Lezárult az adatfeldolgozó rész.

3.3.1.2. RSS()

Mi az RSS és miért jó a médiatárnak.

Az **RSS** webes együttműködésre szolgáló XML állományformátumok családja, mely megkíméli a felhasználókat attól, hogy az ilyen megoldást használó weboldalakat rendszeresen kelljen látogatniuk az új tartalom ellenőrzése miatt, vagy levélben kelljen értesítést kapniuk erről. Egy feed-olvasóként vagy aggregátorként ismert program képes ellenőrizni az RSS-t használó weboldalakat a felhasználó helyett és képes megjeleníteni a frissített cikkeket. Az RSS-t gyakran frissülő szájtok használják, az oldalon megjelenő új tartalom rövid összefoglalójának terjesztésére. Sok szempontból a hírlevél Web2.0-es utódjának tekinthető.

```
618   //Az RSS file létrehozása
619   private static void RSS() throws FileNotFoundException, IOException {
620       String oldrss = null;
621       Vector oldrsslistt = new Vector();
622       boolean van = false;
623
624       //Beállítom az idő változót
625       SimpleTimeZone ect = new SimpleTimeZone(1, "ECT");
626       ect.setStartRule(Calendar.APRIL, 1, Calendar.SUNDAY, 2 * 60 * 60 * 1000);
627       ect.setEndRule(Calendar.OCTOBER, -1, Calendar.SUNDAY, 2 * 60 * 60 * 1000);
628       Calendar calendar = new GregorianCalendar(ect);
629       Date Most = new Date();
630       calendar.setTime(Most);
631
632       Iterator it = ci.iterator();
633
634       File file = new File(config.rsslogut + "" + config.rsslog);
635       if (!file.exists()) {
636           System.out.println("Nincs előző log file.");
637           System.out.println("Minden elemet berakok a hírekbe...");
638
639           PrintStream out = new PrintStream(new FileOutputStream(config.rssoutut + "" + config.rssnam))
640
641           out.println("<?xml version=" + (char) 34 + "1.0" + (char) 34 + "?>");
642           out.println("<rss version=" + (char) 34 + "2.0" + (char) 34 + ">");
643           out.println("<channel>");
```

24. Kép RSS-t készítő kód 1

A rész megértéséhez egy kicsit érteni kell az RSS működését. A hírcsatorna információkat az olvasóprogram a weboldalról csak akkor tölti le, ha annak a létrehozási dátuma frissebb az utolsó letöltöttnél. Ezen áll vagy bukik a működésének sikeressége.

A program beállítja magának a naptárinformációkat, lekérdezi a futtató gép dátumát, óráját, aktuális percét. A későbbiekben ezt fogja felhasználni a szolgáltatás létrehozási dátumának beállításakor.

A RSS fájlt a legegyszerűbb módon hoztam létre; megnéztem egy mintát és annak az adatelemeit íratom ki egy általam készített fájlba, természetesen a megfelelő saját információval feltöltve.

A dátum betöltése után látható, hogy itt is a vektorok iterátoraival fogunk dolgozni. A következő sor megpróbálja megnyitni az előző programfuttatás által létrehozott log fájlt, amely tartalmazza az eddig már betöltött és az előző RSS-ekben is már szereplő tételek azonosítóját. Ennek a fájlnak az elérését a config fájlban adhattuk meg még a futtatás előtt és az abból nyert információk alapján a config globális változócsoport információit használjuk.

Amennyiben nem találja a fájlt a megadott helyen, erről értesít minket rövid szöveges üzenetben és elkészíti a teljes cím vektorból az RSS fájlt. Ehhez létrehoz egy új fájlt a megadott helyre és abba kezdi bepakolni a szöveges sorokat. (A teljes kódot nem fogom itt megjeleníteni, mivel az nagyon sok lenne; csak a lényegi részeket illesztem be.)

```
655             out.println("<item>");
656             out.println("<title>Ezek az új dokumentumaink.</title>");
657             out.println("<link>http://www.lib.unideb.hu</link>");
658
659             while (it.hasNext()) // Ebben a kis ciklusban bepakolom az rss file-be a megfelelő elemeket
660             {
661                 adat b = (adat) it.next();
662                 out.println("<description><a href=" + config.link + "" + b.ID.toString().trim() + ">"
663                         + b.cim + "</a>" + b.rendezo + "</description>");
664             }
665
666             out.println("<image>");
667             out.println("<url>http://www.lib.unideb.hu/hun/pic/fejlec.png</url>");
668             out.println("<link>http://www.lib.unideb.hu</link>");
```

25. Kép RSS-t készítő kód 2

A kiemelt algoritmus részletben a *while* ciklus érdekes a számunkra, a többi sor csak a szokásos módon a szöveges fájlhoz hozzáírok egy sort ismételgeti.

A ciklusban a már korábban is látott - ameddig van következő elem addig fut – kialakítás működik. Ez segít a vektoron való gyors végighaladásban. Az adatokat beszúró sort meg kellett tördelnem, hogy el lehessen olvasni itt, de így ez nem működik! Az eredeti

forráskódban egy sorként szerepel. A sor érdekesen épül fel. Vannak benne fix elemek és sok konstans is. A „*<description>" + b.cim + "" + b.rendezo + "</description>");*” rész amely beszúrja a címet, lezárja a linket, beszúrja a szerzőt és lezárja a sort. Amikor olvassuk az RSS ebből csak annyi látszik, hogy van egy cím, amire ha rákattintunk megnyitja a dokumentumhoz tartozó információs oldalt a könyvtári szerverről; és még utána ki van írva a rendező is.

Viszont ha már van előző rss log fájl, akkor a következő rész jön.

```
682         FileInputStream f = new FileInputStream(file);
683         LineNumberReader in = new LineNumberReader(new InputStreamReader(f));
684         oldrss = in.readLine();
685
686         StringTokenizer st = new StringTokenizer(oldrss, ", ");
687         while (st.hasMoreTokens()) {
688             oldrsslistt.add(((String) st.nextToken()).trim());
689         }
690
691         PrintStream out = new PrintStream(new FileOutputStream(config.rssoutut + "" + config.rssnam));
692
693         out.println("<?xml version=" + (char) 34 + "1.0" + (char) 34 + "?>");
694         out.println("<rss version=" + (char) 34 + "2.0" + (char) 34 + ">");
```

26. Kép RSS-t készítő kód 3

Betöltöm az előző futtatásnál létrejött log fájl tartalmát egy változóba. Ezt fel kell dolgozni és olyan formába hozni, hogy gyorsan és egyszerűen lehessen benne keresni. Ezt a következő három soros kis ciklus végzi el. Ehhez az *RSS()* metódus elején létrehoztam egy *oldrsslist* nevű vektort. Ezután minden marad az előző ciklusrész bemutatásánál bemutatottak szerinte egészen a *while* ciklusig.

```
711   while (it.hasNext()) // megvizsgálom h van e a logban az éppen beszurni kívánt elem,
712   {
713       van = false;
714       adat a = (adat) it.next();
715       Iterator it2 = oldrsslistt.iterator();
716       while (it2.hasNext()) {
717           String b = (String) it2.next();
718           if (a.ID.equals(b)) {
719               van = true;
720           }
721       }
722       if (!van) {
723           out.println("<description><a href=" + config.link + "" + a.ID + ">" + a.cim +
724       }
725   }
```

27. Kép RSS-t készítő kód 4

Az első *while* ciklus a cím vektoron halad végig. Minden egyes elemet a vektorból kiemelünk egy változóba és a következő *while* ciklus segítségével, ami a log fájl elemeit tartalmazza ellenőrizzük, hogy az előző futásnál már az Marc rekordok között volt e az elem. Ha igen akkor az első *while* ciklusban a kővetkező elemre lépünk, és ez megy újra és újra. Amikor olyan elemet találunk, amely azonosítójának nincsen párja a logban található azonosítók között, akkor a már ismertetett sor szúródik be az elem adataival. Az RSS fájl vége megegyezik az előzőekben leírtakkal.

Ez a hírszolgáltatás a Kenézi könyvtárnál a következő címen érhető el: http://media.lib.unideb.hu/hun/lists/generalt_dvd/rss/rss.xml

3.3.1.3. newlog()

Ez azért szükséges, mert az RSS szolgáltatásnál csak az új dokumentumokat kell a szolgáltatásban kiküldeni. A log fájl tartalmazza a már egyszer feldolgozott dokumentumok azonosítóját.

```
387      //a log file létrehozása
388      private static void newlog() throws FileNotFoundException {
389          PrintStream out = new PrintStream(new FileOutputStream(config.rsslogut + "" + config.rsslog));
390          Iterator it = ci.iterator();
391
392          while (it.hasNext()) {
393              adat a = (adat) it.next();
394              String ID = a.ID;
395              out.print(ID);
396              out.print(", ");
397          }
398          out.close();
399      }
```

28. Kép Loggoló kód

A metódus nagyon egyszerű. Létrehoz egy fájlt egy előre meghatározott helyen és névvel, Az egyik vektor elemein végighaladva azokból az ID-ket kiírja a fájlba és rak egy elválasztójelet közéjük. A végén, amikor elfogytak az elemek lezárja a fájlt. Ezzel készen van a log fájlunk, amiben az összes feldolgozott elem benne van.

3.3.1.4. mappaurit()

Ez a kis metódus azért volt fontos, mert amikor elkezdjük létrehozni a mappákba a fájlokat és azok már léteznek a program egyszerűen a már létező fájlok végéhez fűzi a sorokat.

```
39      private static void mappaurit(String cel) {
40          File[] lista = new File(cel).listFiles();
41          if (lista.length != 0) {
42              int i = lista.length;
43              for (int j = 0; i > j; j++) {
44                  String cel2 = (lista[j].getPath());
45                  new File(cel2).delete();
46              }
47          }
```

29. Kép Mappák ürítését végző eljárás

A metódus megkapja argumentumként a mappa címét és kilistázza annak tartalmát. Amennyiben a lista tartalmaz elemeket, az-az a mappa nem üres megnézi hány elem is van benne. A fájlokról készített lista egy tömbben van tárolva ezért egy *for* ciklus segítségével végigolvasom elemenként a tömböt és a kiolvasott címek és neveket segítségével törlöm a mappából az elemet.

Ezt a részt annyiszor kell meghívni ahány mappát ki akarok üríteni.

```
780             mappaurit(config.tmloutut + "/cim/");
781             mappaurit(config.tmloutut + "/rendezo/");
782             mappaurit(config.tmloutut + "/szereplo/");
```

30. Kép Törlés hívása

3.3.1.5. html_cim(), html_rendezo(), html_szereplo()

Ezt a három részegységet egyszerre tárgyalom mivel teljesen megegyeznek, csak a feldolgozott vektoruk más. Akár egybe is lehetne őket vonni, bár ez egy kicsit nagy kuszaságot okozhatna a változók meghatározásánál.

Ehhez a részhez tartoznak azok az elvárások, hogy minden különböző karakterrel kezdődő elem más-más fájlba kerüljön. Ezt egy kis trükkel értem el.

```
401     //a cim szerinti kiiratás
402     public static void html_cim() throws FileNotFoundException {
403         String xx = "cim";
404         String car = null, carold = "z";
405         Iterator it = ci.iterator();
406 
407         for (int i = 0; it.hasNext(); i++) {
408             adat b = (adat) it.next();
409             car = b.cim.substring(0, 1);
410 
411             if (isDigitmy(car) == true) {
412                 PrintStream out = new PrintStream(new FileOutputStream(
413                         config.tmloutut + "/cim/" + config.htmel + "0-9" + config.htmveg, true));
414                 carold = car;
415 
416                 out.println("<tr>");
417                 out.println("<td><a href=" + config.link + "" + b.ID + ">" + b.cim + "</a>"
418                         + b.cim + " </td>");
419                 out.println("<td>" + b.rendezo + " </td>");
420                 out.println("<td>" + b.szereplo + " </td>");
421                 out.println("<td>" + b.sszam + " </td>");
422                 out.println("</tr>");
423             } else {
424                 if (isDigitmy(carold) == true) {
425                     PrintStream out1 = new PrintStream(new FileOutputStream(
426                             config.tmloutut + "/cim/" + config.htmel + "0-9" + config.htmveg, true));
427                     out1.close();
428                     carold = car;
429                 }
430                 if (isSpecCar(car) == true) {
```

31. Kép Tml-ek képzése 1

A metódusban az első ciklus egy *for*, ami végig lépked a vektor elemein és segít az elemek feldolgozásában. Minden egyes elem első karakterét megvizsgálom, ehhez ki kell őket emelni egy egyszerű *substring()* paranccsal. Az első *if*-en belül elhelyezet *isDigitmy()* metódus meghívásával történi, melynek visszatérési értéke igaz vagy hamis lehet, ha igaz akkor a vizsgált karakter szám. (Az *isDigitMy()*-ról később) A ciklusba lépés után létrehozzuk a config fájlban beállított értékek szerint az aktuális karakterrel kezdődő fájlt. Ez számoknál a „0-9” nevet viseli. Ameddig a karakterek, amiket az elemek elejéről másolunk le megegyeznek újra és újra megnyitjuk a létrehozott fájlt és a végére írjuk a megfelelő sorokat.

Amikor a karakter már nem szám és az előző még az volt, akkor a fájlt és a kimenetet lezárom.

A fájlba írt sorok egy html kódolású táblázat celláiban jeleníti meg az elemeket. Ezeket egy PHP motor dolgozza fel majd a szerveren és ezekből építi fel a weboldalakat amikhez készült.

Amint nem számokról beszélünk minden különböző karaktert újabb és újabb fájlba kell elhelyezni. De először a speciális karaktereket kezelem le. Speciális karakterek a mi szempontunkból minden más, ami nem szám és betű.

```
502   if (isSpecCar(car) == true) {
503       PrintStream out = new PrintStream(new FileOutputStream(config.tmloutut + "/r
504       carold = car;
505
506       out.println("<tr>");
507       out.println("<td><a href=" + config.link + "" + b.ID + ">" + b.cim + "</a>"
508       out.println("<td>" + b.rendezo + " </td>");
509       out.println("<td>" + b.szereplo + " </td>");
510       out.println("<td>" + b.sszam + " </td>");
511       out.println("</tr>");
512   } else {
513       if (isSpecCar(carold) == true) {
514           PrintStream out1 = new PrintStream(new FileOutputStream(config.tmloutut
515           out1.close();
516           carold = car;
517       }
```

32. Kép Tml-ek képzése 2

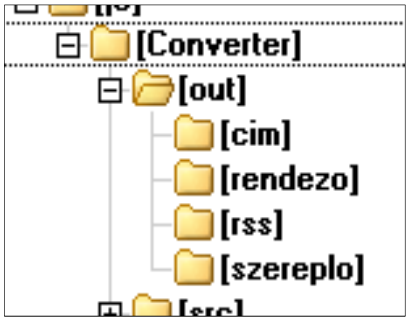
Az *if* ciklus feltételében az *isSpecCar()* metódus meghatározza, hogy speciális karakter e az éppen feldolgozásra váró elem első karaktere. A metódusról később írok. Amennyiben a karakter valamely speciális karakter, akkor létrehozzuk, vagy módosításra megnyitjuk az egyéb karakter nevű fájlt. Majd a már ismertetett módon hozzáadjuk a sorokat. Amennyiben a következő elem első karaktere nem speciális karakter lezárom a fájlt és a kimenetet.

```
519         if (!carold.equals(car)) {
520             PrintStream out1 = new PrintStream(new FileOutputStream(config.tmlo
521 
522             out1.close();
523 
524             PrintStream out = new PrintStream(new FileOutputStream(config.tmlou
525 
526             out.println("<tr>");
527             out.println("<td><a href=" + config.link + "" + b.ID + ">" + b.cim
528             out.println("<td>" + b.rendezo + " </td>");
529             out.println("<td>" + b.szereplo + " </td>");
530             out.println("<td>" + b.sszam + " </td>");
531             out.println("</tr>");
532         } else {
533             PrintStream out = new PrintStream(new FileOutputStream(config.tmlou
534 
535             out.println("<tr>");
536             out.println("<td><a href=" + config.link + "" + b.ID + ">" + b.cim
537             out.println("<td>" + b.rendezo + " </td>");
538             out.println("<td>" + b.szereplo + " </td>");
539             out.println("<td>" + b.sszam + " </td>");
540             out.println("</tr>");
541         }
```

33. Kép Tml-ek képzése 3

Amennyiben a karakter se nem szám, se nem speciális karakter a 31. képen látható rész következik. Vizsgáljuk, hogy a következő elem kezdőbetűje megegyezik-e az előzővel; ha nem akkor lezárom az előző fájlt és a kimenetet és megnyitom az új kezdőbetűvel nevesített fájlt. Majd beleteszem az első elem adataival a megfelelő sorokat. Amikor egyezés van az előző elem és az aktuális elem kezdőbetűje között egyszerűen csak a fájl végéhez fűzöm a sorokat és továbblépek a következő elemre.

Mind a három metódus, amelyek a tml fájlokat készítik, szerkezetileg teljesen megegyeznek, csak a használt vektoruk és a célmappájuk más. A célmappájuk előre meghatározott és nem módosítható csak az elérési útja a config fájl módosításával. A fixen felhasznált mappáknak léteznie kell, a program csak a fájlokat hozza létre a mappaszerkezetet nem. A config fájlban megadott elérési út végén a 32. képen látható mappaszerkezetnek kell lennie.



34. Kép Mappa fa

A mellékletekben megtalálható egy elkészült fájl tartalma.

3.3.1.5.1. isDigitmy()

Ez a kis algoritmus eldönti, hogy a kapott karakter szám vagy nem.

```
79  private static boolean isDigitmy(String car) {
80      if (car == null) {
81          System.out.println(car);
82          return false;
83      }
84      String Digit[] = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "0"};
85      int k = 10;
86      for (int n = 0; n < k; n++) {
87          if (car.equals(Digit[n])) {
88              return true;
89          }
90      }
91      return false;
92  }
```

35. Kép isDigitmy()

Ha a kapott karakter „üres” akkor egyértelműen nem szám ezért a visszatérési érték hamis. Egy string típusú tömbben felsorolom a számjegyeket és a továbbiakban azt vizsgálom, hogy a kapott karakter megegyezik e valamely elemmel a tömbben. Egyezés esetén a visszatérési érték igaz, minden egyéb esetben hamis.

3.3.1.5.2. isSpecCar()

A speciális karaktereket vizsgáló metódus. Hasonló elven alakítottam ki, mint a számokat vizsgáló metódust.

```
94  //Speciális karakterek vizsgálata
95  private static boolean isSpecCar(String car) {
96      String SpecCar[] = {"'", ";", ",", "+", "-", "%", ".", "!", "=", "@", "#", "/", "" + (char) 34};
97      int k = 13;
98      for (int i = 0; i < k; i++) {
99          if (car.equals(SpecCar[i])) {
100             return true;
101         }
102     }
103     return false;
104 }
```

36. Kép isSpacCar()

Egy tömbben összegyűjtöm a lehetséges speciális karaktereket. Ezek a jelek, amiket kigyűjtöttem elvileg lefedik a teljes lehetséges készletet, ami előfordulhat a rekordokban.

Egyéb karakterek valamilyen hibára utalnak, ezért ezekkel nem kell foglalkozni, mivel a MARC4J nem engedi, hogy megjelenjenek.

A metódusban egy *for* ciklus segítségével végignézem, hogy van e egyezés a tömb elemeivel, ha nincsen akkor nem speciális karakter, tehát a visszatérési érték hamis. Ha van egyezés a visszatérési érték igaz. Ezzel a nem túl bonyolult kis megoldással sok problémát sikerült megoldani. Ameddig ez nem volt benne a programban bizonyos elemeknél rossz állományba pakolta a sorokat. Ez megoldotta a problémát.

4. Záró gondolatok

A program megírásánál az a cél vezérelt, hogy bemutassam a programomat és eny programozáshoz kevésbé értő számára is érthetővé tegyem a kódot. Nem tértem ki a dolgozatban a Marc rekordok szerkezetére, tulajdonságaira, és az utófeldolgozást végző PHP kódra sem, mivel ezek más témakörhöz tartoznak és nem ezek bemutatása volt a célom.

Ez a program egy folyamatosan fejleszthető, fejlesztendő, a könyvtár web fejlesztésénél hasznos segítség. Remélhetőleg az engem követő diákok további fejlesztéseket fognak rajta végezni. Fejlesztési irány lehet a szemantikus világháló iránya, de ezt nem az én dolgom eldönteni.

5. Köszönetnyilvánítás

Ez úton szeretnék köszönetet mondani témavezetőmnek, Dr. Boda István tanár úrnak, hogy lehetőséget biztosított a dolgozatom megírásához, valamint segítő tanácsaiért.

Köszönetet mondanék még Balázs Lászlónak, aki a program megírását adta feladatként.

6. Irodalom jegyzék

[1.] http://hu.wikipedia.org

[2.] http://www.java.com

[3.] ELTE (Jegyzet) - Java nyelv programozóknak

[4.] http://netbeans.org/

[5.] http://marc4j.tigris.org/

[6.] Nyékyné Gaizler Judit (szerk.): J2EE Útikalauz Java programozóknak

7. Melléklet

U.tml tartalma

```
<tr>
    <td><a href=http://webpac.lib.unideb.hu:8082/WebPac/CorvinaWeb?action=cclfind&
    resultview=longlong&ccltext=idno+bibDEK403067>U-571</a>U-571 </td>
    <td>Mostow, Jonathan </td>
    <td>Matthew McConaughey </td>
</tr>
<tr>
    <td><a href=http://webpac.lib.unideb.hu:8082/WebPac/CorvinaWeb?action=cclfind&
    resultview=longlong&ccltext=idno+bibKLT00386943>Un ballo in maschera</a>Un ballo
    in maschera </td>
    <td>Verdi, Giuseppe (1813-1901) </td>
<td>James Levine, vez.  </td>
</tr>
<tr>
    <td><a href=http://webpac.lib.unideb.hu:8082/WebPac/CorvinaWeb?action=cclfind&
    resultview=longlong&ccltext=idno+bibDEK00667676>Ungarn, Siebenbürgen, Kroatien-
    Slawonien</a>Ungarn, Siebenbürgen, Kroatien-Slawonien </td>
    <td> </td>
    <td> </td>
</tr>
<tr>
    <td><a href=http://webpac.lib.unideb.hu:8082/WebPac/CorvinaWeb?action=cclfind&
    resultview=longlong&ccltext=idno+bibKLT00329278>Utolsó belövés</a>Utolsó
    belövés </td>
    <td>Hall, Vondie Curtis </td>
    <td>Tupac Shakur  </td>
</tr>
```