Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleš Urbas

Uporaba XML zapisa za implementacijo podatkovne zbirke

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Tomaž Dobravec

Ljubljana, 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00310/2012
Datum: 14.05.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ALEŠ URBAS**

Naslov: **UPORABA XML ZAPISA ZA IMPLEMENTACIJO PODATKOVNE ZBIRKE**

A DATABASE IN XML FORMAT

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomskem delu opišite format zapisa XML ter pomen datotek DTD, XSLT in podobno; omenjene datoteke opišite preko dobrih zgledov. V nadaljevanju izdelajte koncept XML baze - datoteka zapisana v XML formatu, ki hrani vse podatke o bazi (sheme, tabele, strukture in podatke). V programskem jeziku Java izdelajte preprosto orodje za delo z XML bazami, ki naj omogoča povezavo na bazo, izdelavo baze in tabel, vnos in prikaz podatkov v tabelah in izvajanje sql stavkov.

Mentor:

doc. dr. Tomaž Dobravec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aleš Urbas, z vpisno številko **63060365**, sem avtor diplomskega dela z naslovom:

Uporaba XML zapisa za implementacijo podatkovne zbirke

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Tomaža Dobravca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. september 2012 Podpis avtorja:

ZAHVALA

Zahvalil bi se rad vsem sošolcem in prijateljem, ki so mi tako ali drugače pomagali pri moji študijski poti. Hvala tudi izvrstnemu mentorju doc. dr. Tomažu Dobravcu, ki me je usmerjal skozi diplomsko delo. Posebna zahvala pa gre moji družini, ki me je skozi vsa leta mojega šolanja spodbujala, pomagala, ter mi stala ob strani. Iz srca HVALA vsem naštetim !

Kazalo

Seznam kratic in simbolov

XML - Extensible Markup Language
W3C - World Wide Web Consortium
DTD - Document Type Definition
XSD - XML Schema Definition
XSL - Extensible Stylesheet Language
XSLT - Extensible Stylesheet Language Transformations
RSS - Rich Site Summary
SOAP - Simple Object Access Protocol
XHTML - Extensible HyperText Markup Language
PCDATA - Parsed Character Data
CDATA - Character Data
UTF-8 - Unicode Transformation Format 8-bit
UTF-16 - Unicode Transformation Format 16-bit
IDE - Integrated Development Environment
JVM - Java Virtual Machine
DOM - Document Object Model
IDE - Integrated Development Environment
JDBC driver - Java Database Connectivity driver

Povzetek

Cilj diplomske naloge je bil v programskem jeziku Java razviti program s pomočjo katerega bomo lahko upravljali z datotekami tipa XML, kot s podatkovno bazo. Napisan program omogoča ustvarjanje nove podatkovne baze ali pa odpiranje že obstoječe. Prav tako ima možnost pregleda shem in tabel znotraj datoteke XML. Sheme lahko dodajamo ali pa brišemo. Znotraj shem lahko dodajamo nove tabele ali pa se odločimo za urejanje in brisanje že obstoječih. Razvit program prav tako omogoča poizvedbo stavkov SQL. V diplomski nalogi so tudi iz teoretičnega vidika razložene tehnologije: XML, DTD, shema XML, XLink, XPointer, DOM razčljenjevalnik in Java.

Ključne besede:
XML, podatkovna baza, DTD, DOM, shema XML, Java, NetBeans, XLink, XPointer.

Abstract

The aim of the thesis is to develop a program in the Java programming language with which we can manage files of XML type as the database. Developed program can create a new database or open an existing one. It also has the possibility to review schemes and tables within XML. Schemes can be added or deleted. We can also add new tables to schemes or choose to edit or delete the existing ones. Developed program also have an option of SQL queries. The thesis also explains the theoretical point of view on next technologies: XML, DTD, XML Schema, XLink, XPointer, DOM and Java.

Key words:
XML, database, DTD, DOM, XML Schema, Java, NetBeans, XLink, XPointer.

Poglavje 1

Uvod

Skozi stoletja se je družba in gospodarstvo zelo spreminjalo. Danes poznamo družbo, ki se ji reče informacijska družba. Informacijska družba se ukvarja s storitvami in vsem kar je povezano z njimi. Zbiranje, hranjenje, obdelava in distribucija podatkov [2]. Vsi ti podatki so tako ali drugače shranjeni v podatkovnih bazah. Z večjimi podatkovnimi bazami se ukvarjajo večja podjetja, saj so ti sistemi zelo zapleteni. V diplomski nalogi se ne bomo ukvarjali z zapletenimi in velikimi podatkovnimi bazami. Namesto tega si bomo pogledali razvit program, ki omogoča branje in pisanje celotne podatkovne baze v eno samo datoteko XML. Program omogoča osnovne operacije večjih podatkovnih baz (pregled in ustvarjanje tabel, dodajanje zapisov v tabele, poizvedbe itd.), ne potrebuje nobene namestitve na računalnik, ampak se samo zažene in program že lahko uporabljamo za branje, urejanje in ustvarjanje podatkovne baze.

V diplomski nalogi so najprej predstavljene tehnologije XML, DTD, shema XML, XSL, XLink, XPointer. Sledi predstavitev odprtokodnega integriranega razvojnega okolja NetBeans, programskega jezika Java in DOM standarda. V nadaljevanju je bolj podrobno razložen razvit program, njegove podatkovne strukture, algoritmi in njegova uporaba. V zaključku naloge pa so zapisane možne izboljšave programa.

Poglavje 2

Tehnologije in orodja

2.1 Splošno o XML

XML je kratica za angleški izraz e**X**tensible **M**arkup **L**anguage (razširljiv označevalni jezik), ki je definiran z meta jezikom SGML (**S**tandard **G**eneralized **M**arkup **L**anguage). XML je v skladu s specifikacijo opredeljeno s strani World Wide Web Consortium (W3C), dosegljiv na http://www.w3.org. To pomeni, da je XML standard. [8]
XML [10] je enostaven, vsesplošen, ter spreminja mnogo vidikov računalništva, prevsem na področju komuniciranja med aplikacijo in strežnikom (arhitektura odjemalec-strežnik). Razdeljen je na tri dele:

- Podatkovni - vanj shranjujemo podatke v neki obliki z želenimi značkami.
- Deklarativni - pri dodajanju novih podatkov vidimo kaj kakšna značka predstavlja.
- Predstavitveni - oblikujemo izpis podatkov.

XML je namenjen transportu, ter shranjevanju podatkov in ima vrsto dobrih lastnosti, zato je po svetu zelo razširjen. Ima silno preprosto in pregledno zgradbo, ter omogoča integracijo strukturiranih podatkov iz več virov v logične in preprosto pregledne podatke. Podatki so sámo opisni in so lahko sprejeti in procesirani brez potrebe po še dodatnih komentarjih. Zapisani podatki, ki jih vsebuje XML, so brez težav preneseni skozi skripto ali druge programske jezike, s pomočjo XML

objektnega modela, v poljubno aplikacijo. Odkar obstaja XML pa do danes, se je razvilo že več sto jezikov, ki temeljijo na razširljivem označevalnem jeziku, npr. RSS, SOAP, XHTML, ...

2.2 Pravila dokumenta XML

Tako kot vsak programski jezik ima tudi XML določena pravila, ki se jih moramo držati. Te so v primerjavi s programskimi jeziki izredno enostavna in se jih hitro naučimo. Dokument XML je besedilna datoteka, ki upošteva vsa ta pravila in ima končnico `.xml`.

2.2.1 Deklaracija XML

Dokument XML ni nujno, je pa zaželjeno deklarirati. Deklaracija se mora nahajati v prvi vrstici dokumenta. Trenutno je edina verzija 1.0, vendar v prihodnosti obstaja možnost novejših različic. V deklaraciji lahko prav tako zapišemo vrsto kodiranja dokumenta (UTF-8, UTF-16, ISO-8859-1,...).

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

Zapis deklaracije dokumenta XML

2.2.2 Značke

Poznamo tri vrste značk:

- **Začetna značka (angl. starting tag)**:
  Začetna značka je značka, ki se začne z znakom <, sledi ji ime značke, konča pa se z znakom >. Pravilen in nepravilen zapis značke je lepo razviden iz naslednjih dveh primerov.

```
Nepravilno:   <ime<
Pravilno:     <ime>
```

- **Zaključna značka (angl. ending tag) in velike in male črke (angl. case sensitive)**:
  Zaključna značka obvezno stoji za začetno značko. Obe znački morata biti zapisani popolnoma enako, saj XML razlikuje med velikimi in malimi črkami. Edina razlika med njima je znak / pred imenom značke.

```
<!-- Nepravilna uporaba zakljucne znacke -->
<ime>Marko<ime>
<Priimek>Skok</priimek>
<!-- Pravilna uporaba zakljucne znacke -->
<ime>Marko</ime>
<Priimek>Skok</Priimek>
```

Zaključna značke in velike in male črke

- **Prazna značka (angl. empty tag)**:
  Kot že ime samo pove, je značka, ki ne vsebuje nobenega elementa (več o elementih v naslednjem poglavju). Za zapis prazne značke imamo dve možnosti:
  1) `<posta></posta>`
  2) `<posta/>`
  Razlike med prvim in drugim zapisom ni nobene, tako da lahko uporabimo bodisi prvi, bodisi drugi zapis.

2.2.3 Elementi in atributi

Element XML predstavlja vse od začetne značke pa do zaključne značke (vključno z njima). Element lahko vsebuje [10]:

- besedilo
- atribute
- druge elemente
- kombinacijo vsega naštetega.

```
<trgovina>
   <izdelek kategorija="prehrana">
     <znamka>Ljubljanske mlekarne</znamka>
     <ime>Jogurt 1.6 MM</ime>
     <cena>0,95</cena>
   </izdelek>
   <izdelek kategorija="pijaca">
     <znamka>Pivovarna Lasko</znamka>
     <ime>Pivo Zlatorog Klub</ime>
     <cena>1,22</cena>
   </izdelek>
</trgovina>
```

Uporaba različnih elementov

V zgornjem primeru elementi, ki se začnejo z značkami `<znamka>`, `<ime>` in `<cena>` vsebujejo besedilo. Elementa `<trgovina>` in `<izdelek>` vsebujeta ostale elemente. Element `<izdelek>` poleg ostalih elementov, vsebuje še atribut `kategorija`.

Vsak dokument XML mora vsebovati natanko en korenski element (angl. root element). To je element, kateremu se začetna značka nahaja na začetku dokumenta, zaključna značka pa na koncu dokumenta XML in vsebuje vse ostale elemente. Vsi ostali elementi, ki jih je sicer lahko poljubno mnogo, so njegovi podrejeni elementi (angl. child elements). Pri zadnjem zgledu imamo korenski element, ki se začne z značko `<trgovina>`, ter ostale elemente, ki so podrejeni korenskemu elementu in se začnejo z značkami `<izdelek>`, `<znamka>`, `<ime>`, `<cena>`.

Elemente moramo pravilno gnezditi. Preden se element, ki vsebuje besedilo, ne zaključi, ne moremo začeti z novim.

```
<!-- Nepravilno gnezdenje elementov -->
<Student>
        <ime>Marko<priimek>
        </ime>Klepec</Student>
</priimek>
```

```
<!-- Pravilno gnezdenje elementov -->
<Student>
        <ime>Marko</ime>
        <priimek>Klepec</priimek>
</Student>
```

Gnezdenje elementov

Vsak element znotraj XML dokumenta lahko vsebuje atribut. Atribut mora biti naveden v začetni znački in je lahko v eni izmed dveh oblik [10]:

1. ime=”vrednost”
2. ime='vrednost'

Med zapisoma ni nobene razlike, vendar se v praksi večkrat uporabi prvi zapis. V spodnjem zgledu je primer uporabe atributa `fakulteta` znotraj elementa `<student>`.

```
<student fakulteta="FRI">Samo Hrast</student>
```

V praksi se je priporočeno izogibati atributom, če je le to mogoče. Atribute nadomestimo z elementi [10]. Zakaj ?

- Atribute je težje brati in vdrževati kot elemente.
- Atributi so težje razširljivi, kot elementi.
- Atribut ne more vsebovati več vrednosti, element ima to opcijo.
- Atribut ne more vsebovati drevesne strukture, element lahko.

Ko shranjujemo metapodatke (podatke o podatkih) je te smiselno zapisati v atribut. Same podatke pa je bolj smiselno shranit kot element. V spodnjem primeru nam `format="jpg"` predstavlja metapodatek, zato ga je smiselno shranit kot atribut in ne kot element.

```
<slika format="jpg">avtomobil</slika>
```

V spodnjem primeru smo najprej shranili `kategorijo` kot atribut, kasneje pa kot element. Kot smo že omenili, je priporočljivo shranjevanje podatkov v elemente.

```
<!-- kategorija kot atribut -->
<izdelek kategorija="prehrana">
        <znamka>Ljubljanske mlekarne</znamka>
        <ime>Jogurt 1.6 MM</ime>
        <cena>0,95</cena>
</izdelek>
<!-- kategorija kot element -->
<izdelek>
        <kategorija>prehrana</kategorija>
        <znamka>Ljubljanske mlekarne</znamka>
        <ime>Jogurt 1.6 MM</ime>
        <cena>0,95</cena>
</izdelek>
```

Kategorija kot atribut/element

XML vsebuje nekaj omejitev glede samih imen elementov. Te ne smejo :

- vsebovati presledkov ali znakov `!"#$&'()*+,/;<=>?@[]^‘|~{}`,

```
<emso stevilka}, <nasl()v>, <@les>, ...
```

- se začeti s številko,

```
<5ime>, <13priimek>, <64nintendo>, ...
```

- se začeti s črkami XML.

```
<xmlNaslov>, <XMLpotrdi>, <XmLIme>, ..
```

Razen zgoraj naštetih omejitev, drugih omejitev pri imenih XML elementov ni. XML ne pozna nobenih rezerviranih besed, tako da si lahko popolnoma prilagodimo imena. Obstajajo pa nasveti s katerimi sebi in drugim olajšamo delo. Priporočljivo je, da so imena elementov kratka in enostavna, brez potrebe po dodatnih komentarjih. Med imeni, namesto pike, minusa ali dvopičja, raje uporabimo podčrtaj.

```
<!-- Nepriporocljivo -->
<znamka.telefona>Samsung</znamka.telefona>
<operacijski-sistem>Windows</operacijski-sistem>
<!-- Priporocljivo -->
<znamka_telefona>Samsung</znamka_telefona>
<operacijski_sistem>Windows</operacijski_sistem>
```

Imena elementov

2.2.4 Komentarji

XML komentarji so enaki HMTL komentarjem. Pred komentar zapišemo `<!--`, potem sledi komentar, zaključimo pa z `-->`. Velikokrat v dokumentih XML komentarji niso potrebni, saj že same značke povejo veliko (zadosti), kdaj pa so vseeno priročni in jih je pametno pisati.

```
<!-- Nov avtomobil -->
<avto>
        <!-- Znamka avtomobila -->
        <znamka>Audi</znamka>
        <!-- Ime modela -->
        <ime>A5</ime>
        <!-- Letnik prve registracije -->
        <letnik>2011</letnik>
```

```
        <!-- Moc motorja v KM -->
        <moc>205</moc>
</avto>
```

Uporaba komentarjev

2.2.5 Uporaba prepovedanih znakov

Znotraj elementov XML znaka `&` in `<` nista dovoljena, zato ju moramo zapisati drugače. Npr. znak `<` ima vlogo začetne oz. zaključne značke in je rezerviran znak, zato ga moramo v dokumentu XML zapisati kot `<`. Znak `&` pa zapišemo kot `&`. Tukaj so še ostali znaki, ki pa jih ni nujno zapisati drugače, je pa priporočljivo, v izogib problemov, ki morda lahko kasneje nastanejo. Znak `>` je priporočljivo zapisati kot `>`, znak `'` kot `&apos;`, znak `''` pa zapišemo kot `&quot;`.

```
<!-- Prepovedan zapis -->
<obvestilo>Ce je placa < 1000 potem</obvestilo>
<risanka>Tom & Jerry</risanka>
<!-- Pravilen zapis -->
<obvestilo>Ce je placa &lt; 1000 potem</obvestilo>
<risanka>Tom &amp; Jerry</risanka>
```

Prepovedani znaki

2.2.6 Pravilno oblikovan dokument XML

Dokumentu XML pravimo, da je pravilno oblikovan (angl. well formed), ko ustreza vsem napisanim pravilom v podpoglavjih 2.2.2 - 2.2.5.

2.2.7 Veljaven dokument XML

Dokument XML je veljaven, ko je pravilno oblikovan, ter hkrati ustreza pravilom DTD.

2.3 Splošno o DTD

DTD je angleška kratica za **D**ocument **T**ype **D**efinition. Z jezikom XML ne moremo do potankosti opisati kako naj bi dokument XML izgledal, zato nam je v pomoč DTD. Z njegovo pomočjo lahko točno določimo kateri elementi in atributi se smejo oz. ne smejo uporabljati. Določimo jim lahko ali je nek atribut obvezen ali ne, privzeto vrednost itd. S pomočjo DTD-ja lahko poljubna aplikacija preveri veljavnost podatkov pridobljenih iz zunanjega sveta. Prav tako se lahko podjetja med seboj dogovorijo za nek skupen, standarden DTD, ki se ga držijo in s tem omogočijo lažjo izmenjavo podatkov med aplikacijami [4].

2.4 Pravila DTD

DTD ima, tako kot XML, posebna pravila, ki se jih je potrebno držati. Ta so prav tako enostavna in hitro naučljiva.

2.4.1 Deklaracija DTD

V dokumentu XML je potrebno deklarirati DTD. To naredimo takoj po deklaraciji XML-ja, ter pred zapisom ostalih podatkov. Imamo dve možnosti kako deklarirati DTD:

- Znotraj dokumenta XML:

```
<!DOCTYPE KorenskiElement[deklaracijaElementov]>
```

- V ločeni datoteki:

```
<!DOCTYPE KorenskiElement SYSTEM "ImeDatoteke">
```

2.4.2 Elementi in atributi

Elementi so glavni gradniki dokumentov XML, zato je njihova deklaracija znotraj DTD do potankosti opredeljena. Elementom lahko določimo točen vrstni red kako si sledijo eden za drugim, kateri element je podrejen element, prazen element itd. V naslednjem zgledu imamo zapis različnih elementov, ki vsebujejo podatke v DTD, ter njihov zapis v XML-ju. Zgled, ki se nahaja v drugi oz. sedmi vrstici kode nam pove, da imamo v dokumentu XML element z imenom `risanka`, kateri vsebuje besedilo oz. po angleško **P**arsed **C**haracter **DATA**. Primer v tretji in osmi vrstici je zelo podoben prejšnjemu primeru, s to razliko, da element z imenom `priimek` vsebuje besedilo, ki ne bo posredovano razčlenjevalniku (angl. parser). CDATA je angleška okrajšava za **C**haracter **DATA**. Predzadnji primer v četrti oz. deveti vrstici nam pove, da imamo v dokumentu XML element `posta`, ki ne vsebuje nobenih podatkov in je prazen (angl. empty). Če želimo, da odstranimo vso sintakso preverjanja elementa, uporabimo besedo `any`, tako kot smo to storili v peti oz. deseti vrstici.

```
1  <!-- Zapis elementa v DTD -->
2  <!ELEMENT risanka (#PCDATA)>
3  <!ELEMENT priimek (#CDATA)>
4  <!ELEMENT posta EMPTY>
5  <!ELEMENT vzdevek ANY>
6  <-- Primer zapisa v XML -->
7  <risanka>Tom &amp; Jerry</risanka>
8  <priimek>Balon</priimek>
9  <posta></posta>
10 <vzdevek>CP3</vzdevek>
```

Zapis različnih elementov

Omenili smo že, da lahko elementom določimo tudi vrstni red kako si sledijo eden za drugim, ter njegove podrejene elemente. V naslednjem primeru, smo v drugi vrstici kode določili korenski element `<avto>`, njegove podrejene elemente pa zapisali v oklepaj. Podrejeni elementi si morajo nujno slediti v enakem vrstnem redu, kot so zapisani v oklepajih. V nasprotnem primeru dokument XML ni veljaven.

V naslednjih vrsticah smo še določili, da podrejeni elementi vsebujejo besedilo.

```
<!-- Zapis elementov v DTD -->
<!ELEMENT avto (znamka,model,letnik,kilometri,moc)>
<!ELEMENT znamka (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT letnik (#PCDATA)>
<!ELEMENT kilometri (#PCDATA)>
<!ELEMENT moc (#PCDATA)>
<!-- Primer zapisa v XML -->
<avto>
   <znamka>Audi</znamka>
   <model>A5</model>
   <letnik>2011</letnik>
   <kilometri>14000</kilometri
   <moc>205</moc>
</avto>
```

Vrstni red elementov

DTD nam tudi ponuja opcijo, da lahko bolj natančno opredelimo kolikokrat se lahko pojavi podrejeni element [4]. Izbiramo lahko med štirimi opcijami, ki so opisane v naslednjem zgledu.

```
<!-- Podrejeni element znamka
se mora pojaviti tocno: -->
<!-- enkrat -->
<!ELEMENT avto (znamka)>
<!-- enkrat ali veckrat -->
<!ELEMENT avto (znamka+)>
<!-- nobenkrat ali veckrat -->
<!ELEMENT avto (znamka*)>
<!-- nobenkrat ali enkrat -->
<!ELEMENT avto (znamka?)>
```

Podrejeni elementi

V DTD-ju imamo tudi možnost deklaracije ali (angl. or). V spodnjem primeru imamo za vrsto goriva tri opcije, med katerimi izberemo samo eno.

```
<!ELEMENT vrsta_goriva (bencin|diesel|hibrid)>
```

Kot smo omenili že v poglavju 2.2.3 lahko vsak element vsebuje tudi atribut. To v DTD-ju deklariramo po naslednji formuli:

```
<!ATTLIST ime_elementa ime_atributa
tip_atributa privzeta_vrednost>
```

Imeni elementa, ter atributa sta lahko poljubni. Pod tip atributa moramo nujno zapisati enega izmed vrednosti iz leve strani tabele 2.1. Atributu lahko še določimo njegovo privzeto vrednost, katero lahko kasneje spremenimo, (ne)obveznost in pa točno določeno vrednost, ki jo mora atribut zavzeti (drugih vrednosti ne dopuščamo). Te vrednosti so prikazane na levi, opis pa na desni strani tabele 2.2.

2.4.3 Večkrat uporabljene besede

Besede oz. besedne zveze, ki jih znotraj dokumenta XML večkrat uporabimo, lahko deklariramo v DTD-ju. Za to imamo, tako kot pri sami deklaraciji DTD-ja, dve opciji - znotraj dokumenta XML ali v ločeni datoteki [4].

```
<!-- Primer deklaracije znotraj dokumenta XML -->
<!ENTITY ZMK "Zalozba Mladinska knjiga">
<!-- Primer deklaracije v loceni datoteki -->
<!ENTITY ZMK
SYSTEM "http://wwww.mladinska.si/entities.dtd">
```

Deklaracija večkrat uporabljenih besed v DTD

Tip Atributa	Opis
CDATA	Poljubno besedilo
ID	Edinstven ID ali ime
IDREF	Vrednost ID drugega elementa
IDREFS	Vrednost večih ID elementov, ločenih s presledkom
NMTOKEN	Veljavno XML ime
NMTOKENS	Veljavna XML imena, ločena s presledkom
ENTITY	Ime entitete (mora biti deklarirana v DTD)
ENTITIES	Imena entitet, ločenih s presledkom
NOTATION	Ime zapisa (mora biti deklariran v DTD)
(vrednost1 |vrednost2 |...)	Ena izmed vrednosti iz seznama
xml	Vnaprej določena XML vrednost

Tabela 2.1: Tipi atributov

Privzeta vrednost	Opis
value	Poljubna privzeta vrednost (številka ali beseda)
#REQUIRED	Atribut je obvezen
#IMPLIED	Atribut ni obvezen
#FIXED	Vrednost atributa je določena in jo ni mogoče spreminjati

Tabela 2.2: Privzete vrednosti

Ko to besedo uporabimo znotraj dokumenta XML, ne glede na način deklaracije, moramo pred njo napisati znak `&` za njo pa `;`. To je prikazano v naslednjem zgledu.

```
<knjiga>
   <ime>Tom na zadnjem pohodu</ime>
   <avtor>Jerry Veliki</avtor>
   <zalozba>&ZMK;</zalozba>
</knjiga>
```

Primer uporabe v XML

2.5 Splošno o shemi XML

Shema XML ima zelo, zelo podobno nalogo kot DTD (opisovanje strukture dokumenta XML), vendar je veliko bolj izpopolnjena in temelji na XML-ju. S pomočjo

sheme XML lahko določimo točno število znakov znotraj elementa, da sme element vsebovati samo velike ali male črke, da so v elementu lahko zapisane samo številke itd. Vseh teh možnosti DTD nima. Za bolj nazoren prikaz, si poglejmo spodnja primera [1].

```
1 <quantity>5</quantity>
2 <quantity>hello</quantity>
```

Za primer v prvi vrstici bi v DTD-ju zapisali tako - element `<quantity>`, ki vsebuje besedilo (črke ali številke).

```
<!ELEMENT quantity (#PCDATA)>
```

Pri primeru v drugi vrstici bi s pomočjo DTD-ja opisali element XML enako kot za prvi primer. Želeli bi, da bi nam za drugi primer javilo napako, saj moramo v element količina zapisati številsko vrednost, kar pa nismo storili. DTD nam ne ponuja bolj natančne opcije kako opisati vsebino elementov. To možnost imamo pri zapisu sheme XML. Pri njej enostavno določimo, da mora biti znotraj elementa `<quantity>` število, drugače nam javi napako. Tako bi s prvim primerom bilo vse vredu, pri drugem primeru pa bi nam javilo napako. Shema XML pa nam ponuja še veliko več možnosti. Pri W3C [9] so prepričani, da je shema XML naslednica DTD-ja in da bo prevzela njegovo vlogo [11]. Za to imajo upravičene razloge. Shema XML:

- je bogatejša in močnejša od DTD-ja,
- je zapisana v XML-ju,
- podpira podatkovne tipe,
- je razširljiva za prihodnje dodatke.

Shema XML je poznana tudi pod imenom XML Schema Definition (XSD), datoteka pa ima končnico `.xsd`.

2.6 Pravila sheme XML

Shema XML je, tako kot smo že omenili, v primerjavi z DTD-jem veliko močnejša, zato ima posledično tudi veliko več pravil. Pravila so sila enostavna in logična, ter se jih hitro naučimo.

2.6.1 Deklaracija sheme XML

Tako, kot smo že navajeni iz DTD-ja, moramo tudi shemo XML deklarirati. Tako dokument XML ve na katero shemo se sklicujemo, ter pravila katere sheme mora upoštevati. Deklaracija sheme XML sledi takoj za deklaracijo dokumenta XML [11].

```
<xs:schema
xmlns="http://www.spletnastran.si"
xmlns:xs="http://www.spletnastran.si/XMLSchema"
(ostala koda je tukaj)
</xs:schema>
```

Primer deklaracije sheme XML

Prva vrstica v shemi se vedno začne s korenski elementom `<shema>`, konča pa z `</shema>`. Druga vrstica v kodi nam pove, da je privzet imenski prostor "http://www.spletnastran.si/". Tretja vrstica nam pravi, da elementi in podatki, prihajajo iz "http://www.spletnastran.si/XMLSchema"imenskega prostora. Prav tako nam določa, da morajo biti elementi in podatkovni tipi, ki prihajajo iz omenjenega imenskega prostora predpono `xs:`.

2.6.2 Elementi in atributi

Prava moč sheme XML je ravno na področju elementov in atributov. Razlog za to bomo spoznali v nadaljevanju poglavja. Shema deli elemente na enostavne (angl. simple) in zapletene (angl. complex) [11].

Enostavni elementi

Elementi, ki so enostavni, vsebujejo samo besedilo. Bolje rečeno, vsebujejo enega izmed vnaprej definiranih podatkovnih tipov ali pa podatkovni tip, ki ga definiramo

sami. Nekaj najpogostejše uporabljenih podatkovnih tipov je zapisanih v tabeli 2.3.

Podatkovni tip	**Opis**
xs:string	Poljubno besedilo (brez števil)
xs:decimal	Pozitivno ali negativno decimalno ali pa celo število
xs:integer	Pozitivno ali negativno celo število
xs:boolean	Ima dve vrednosti: true (1) ali pa false (0)
xs:date	Datum v formatu YYYY-MM-DD (leto-mesec-dan)
xs:time	Čas v formatu hh:mm:ss (ure:minute:sekunde)

Tabela 2.3: Podatkovni tipi

V spodnjem zgledu si poglejmo primer dokumenta XML, ki je veljaven (glede na shemo).

```
<!-- Shema XML -->
<xs:element name="ime" type="xs:string"/>
<xs:element name="registracija" type="xs:date"/>
<xs:element name="moc" type="xs:integer"/>
<!-- Dokument XML -->
<ime>Audi A5</ime>
<registracija>2012-06-10</registracija>
<moc>205</moc>
```

Deklaracija enostavnih elementa

Poleg naštetih podatkovnih tipov lahko s pomočjo sheme XML določimo tudi svoje podatkovne tipe (angl. facets). Tem lahko sami določimo razne omejitve in jih popolnoma prilagodimo našim potrebam. Na izbiro imamo dva načina, kako definirati tak podatkovni tip. Prvi način je, da znotraj elementa ali atributa definiramo podatkovni tip. Pomankljivost te definicije je, da omenjeni tip ne moramo uporabiti tudi pri drugih elementih/atributih, vendar ga moramo ponovno definirati. Drugi način pa nam omogoča uporabo definiranega tipa na večih elementih/atributih. V naslednjem zgledu, imamo podatkovni tip `avtoTip` (definiran na oba načina), s pomočjo katerega definiramo element in atribut, da zavzame eno izmed naštetih

vrednosti - Audi, BMW ali VW. V primeru vnosa katerekoli druge vrednosti, nam shema XML javi napako.

```
<!-- prvi nacin -->
<xs:element name="avto">
  <xs:simpleType>
    <xs:restriction base="xs:string">
          <xs:enumeration value="VW"/>
          <xs:enumeration value="Audi"/>
          <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:attribute name="avto">
  <xs:simpleType>
    <xs:restriction base="xs:string">
          <xs:enumeration value="VW"/>
          <xs:enumeration value="Audi"/>
          <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<!-- drugi nacin -->
<xs:element name="avto" type="avtoTip"/>
<xs:attribute name="avto" type="avtoTip"/>

<xs:simpleType name="avtoTip">
  <xs:restriction base="xs:string">
    <xs:enumeration value="VW"/>
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
```

```
</xs:simpleType>
```

Deklaracija podatkovnega tipa `avtoTip`

Podatkovni tip `avtoTip` smo omejili s pomočjo enumeration. Izbiramo lahko še med ostalimi omejitvami, naštetimi v tabeli 2.4.

Omejitev	**Opis**
enumeration	Definira seznam sprejemlivih vrednosti
fractionDigits	Definira največje dovoljeno število decimalnih mest (enako ali večje od 0)
length	Določa natančno št. znakov ali seznama sprejemlivih vrednosti (enako ali večje od 0)
maxLength	Določa največje št. znakov ali seznama sprejemlivih vred. (enako ali večje od 0)
minLength	Definira najmanjše št. znakov ali seznama sprejemlivih vred. (enako ali večje od 0)
maxExclusive	Določa zgornjo mejo za številčne vrednosti (mora biti manjša od te vrednosti)
maxInclusive	Določa zgornjo mejo za številčne vrednost (mora biti enaka ali manjša od te vrednosti)
minExclusive	Določa spodnjo mejo za številčne vrednosti (mora biti večja od te vrednosti)
minInclusive	Določa spodnjo mejo za številčne vrednosti (mora biti enaka ali manjša od te vrednosti)
pattern	Definira natančen vrstni red znakov, ki so sprejemlivi
totalDigits	Določa točno število dovoljenih decimalnih mest (večje od 0)
whiteSpace	Določa, kako se presledki (in podobni znaki) obravnavajo

Tabela 2.4: Omejitve pri podatkovnih tipih

Za pridobitev pravega občutka moči sheme XML si poglejmo naslednjih nekaj zgledov.

```
<!-- Element geslo, ki mora vsebovati
najmanj 4 znake in najvec 10 znakov -->
<xs:element name="geslo">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="4"/>
      <xs:maxLength value="10"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<!-- Pravilen zapis v XMLju -->
<geslo>Testno</geslo>
<!-- Nepravilen zapis v XMLju -->
<geslo>akd</geslo>
```

minLength in maxLength

```
<!-- Element id, ki mora vsebovati
natanko 4 stevilke -->
<xs:element name="id">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<!-- Pravilen zapis v XMLju -->
<id>5413</id>
<!-- Nepravilen zapis v XMLju-->
<id>587593</id>
```

Integer pattern

```
<!-- Element odgovor, ki vsebuje samo eno
izmed vrednosti (DA ali NE) -->
<xs:element name="odgovor">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="DA|NE"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<!-- Pravilen zapis v XMLju -->
<odgovor>DA</odgovor>
<!-- Nepravilen zapis v XMLju-->
<odgovor>NEVEM</odgovor>
```

String pattern

```
<!-- Atribut starost, ki se mora nahajati
na intervalu od vkljucno 18, pa do vkljucno 110 -->
```

```
<xs:attribute name="starost">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="18"/>
      <xs:maxInclusive value="110"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<!-- Pravilen zapis v XMLju -->
<ime starost="24">Miha</ime>
<!-- Nepravilen zapis v XMLju -->
<ime starost="15">Miha</ime>
```

minInclusive in maxInclusive

S pomočjo sheme XML lahko enostavnim elementom določimo privzeto ali fiksno vrednost. Privzeta vrednost (angl. default value) je vrednost, ki je elementu samodejno dodeljene, če ni določena nobena druga vrednost. Fiksna vrednost (angl. fixed value) je vnaprej določena in je ni moč na noben način spremeniti.

```
<!-- Shema XML -->
<xs:element name="podjetje"
type="xs:string" default="Microsoft"/>
<!-- Dokument XML -->
<!-- Pravilni zapisi -->
<podjetje>Microsoft</podjetje>
<podjetje>Audi</podjetje>
<podjetje>Heineken</podjetje>
<podjetje>Krka</podjetje>
```

Privzeta vrednost elementa

Element `<podjetje>` lahko zavzame poljubno vrednost (ni nepravilnega vnosa). V primeru, da pustimo element prazen pa zavzame vrednost Microsoft. V naslednjem primeru je vrednost elementa `<podjetje>` fiksno določena, tako da moramo znotraj omenjenega elementa vpisati vrednost Apple, drugače nam javi napako (imamo samo en pravilen vnos, ostali so nepravilni glede na shemo).

```
<!-- Shema XML -->
<xs:element name="podjetje"
type="xs:string" fixed="Apple"/>
<!-- Dokument XML -->
<!-- Pravilnen zapis -->
<podjetje>Apple</podjetje>
<!-- Neravilni zapisi -->
<podjetje>Audi</podjetje>
<podjetje>Heineken</podjetje>
<podjetje>Krka</podjetje>
<podjetje>Gorenje</podjetje>
```

Fiksna vrednost elementa

Atributi

Enostavni elementi ne morajo imeti atributov. Če želi element imeti atribut, mora biti kompleksen. Atributi pa so sami po sebi deklarirani kot enostaven tip (angl. simple type). Atributom je potrebno, tako kot elementom, določiti podatkovni tip. Podatkovni tipi so enaki kot pri elementih. Lahko uporabljamo vnaprej določene podatkovne tipe (zapisane v tabeli 2.3) ali pa določimo svoje podatkovne tipe. Omejitve so zapisane v tabeli 2.4.
Deklaracija atributa je skoraj identična deklaraciji elementa, samo da namesto `<element name="ime">` zapišemo `<attribute name="ime">`.

```
<!-- Shema XML -->
<xs:attribute name="moc" type="xs:integer"/>
<!-- Dokument XML -->
<ime moc="350">Audi A5</ime>
```

Deklaracija atributa

Atributom lahko, tako kot pri enostavnih elementih, določimo privzeto ali fiksno vrednost.

```
<xs:attribute name="podjetje"
type="xs:string" default="Microsoft"/>
```

```
<xs:attribute name="podjetje"
type="xs:string" fixed="Apple"/>
```

Privzeta in fiksna vrednost atributa

Poleg privzete in fiksne vrednosti, lahko atributom določimo, da so obvezni. To pomeni, da se mora atribut nujno nahajati znotraj določenega elementa. V primeru, da ne določimo atributu obveznost, je ta neobvezen.

```
<!-- Shema XML -->
<xs:attribute name="starost"
type="xs:integer" use="required"/>
<!-- Dokument XML -->
<!-- Pravilen zapis -->
<ime starost="12">Klemen</ime>
<!-- Nepravilna zapisa -->
<ime>Klemen</ime>
<ime starost="Petnajst">Klemen</ime>
```

Obvezen atribut starost

Kompleksni elementi

Kompleksni elementi so elementi, ki vsebujejo druge elemente in/ali atribute. Poznamo štiri vrste kompleksnih elementov [11]:

- prazni elementi,
- elementi, ki vsebujejo samo druge elemente,
- elementi, ki vsebujejo samo besedilo,
- elementi, ki vsebujejo druge elemente in besedilo.

Vsi zgoraj našteti elementi lahko vsebujejo atribute.

V prejšnjem podpoglavju o atributih smo že govorili o deklaraciji samih atributov. Sedaj bomo to znanje dopolnili še z deklaracijo praznega elementa, ki vsebuje atribut. Imamo dva načina deklaracije praznega elementa z atributom.

```
<!-- Shema XML -->
<!-- prvi nacin -->
<xs:element name="ime">
  <xs:complexType>
    <xs:attribute name="starost" type="xs:integer"/>
  </xs:complexType>
</xs:element>
<!-- drugi nacin -->
<xs:element name="ime" type="starostTip"/>

<xs:complexType name="starostTip">
  <xs:attribute name="starost" type="xs:integer"/>
</xs:complexType>
<!-- Dokument XML -->
<ime starost="13">Klemen</ime>
```

Deklaracija praznega elementa z atributom

Pri prvem načinu deklariramo element `ime`, znotraj tega elementa pa še atribut `starost`. Pri drugem načinu, pa deklariramo kompleksni tip z imenom `starostTip`, kateri vsebuje atribut z imenom `starost`. Elementu `ime` potem določimo podatkovni tip `starostTip`. Pri drugem načinu, lahko omenjeni podatkovni tip uporabimo še pri drugih elementih, brez potrebe po ponovni deklaraciji.
V enem izmed prejšnih podpoglavji (Enostavni elementi) smo že napisali kako se definira elemente, kateri vsebujejo samo besedilo. Sedaj pa si bomo podrobneje ogledali deklaracijo elementov, ki vsebujejo elemente, te pa besedilo. Na razpolago imamo dva načina deklaracije, za katerima stoji enak način razmišljanja, kot pri praznih elementih.

```
<!-- Shema XML -->
<!-- prvi nacin -->
<xs:element name="oseba">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ime" type="xs:string"/>
```

```
      <xs:element name="priimek" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!-- drugi nacin -->
<xs:element name="oseba" type="osebaTip"/>

<xs:complexType name="osebaTip">
  <xs:sequence>
    <xs:element name="ime" type="xs:string"/>
    <xs:element name="priimek" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<!-- Dokument XML -->
<oseba>
        <ime>Hinko</ime>
        <priimek>Smrekar</priimek>
</oseba>
```

Deklaracija elementa, ki vsebuje elemente

Pokažimo še, kako lahko podatkovni tip `starostTip` uporabimo pri drugih elementih.

```
<!-- Shema XML -->
<xs:element name="oseba" type="osebaTip"/>
<xs:element name="student" type="osebaTip"/>

<xs:complexType name="osebaTip">
  <xs:sequence>
    <xs:element name="ime" type="xs:string"/>
    <xs:element name="priimek" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

```
<!-- Dokument XML -->
<oseba>
        <ime>Hinko</ime>
        <priimek>Smrekar</priimek>
</oseba>

<student>
        <ime>Jan</ime>
        <priimek>Mizar</priimek>
</student>
```

Deklaracija elementa, ki vsebuje elemente

Podatkovnemu tipu `osebaTip` lahko brez večjih težav dodamo poljubne elemente (ga razširimo). V naslednjem primeru bomo elementu `student`, poleg že definiranega `imena` in `priimka`, dodali še `vpisno številko` in `kraj`.

```
<!-- Shema XML -->
<xs:element name="student" type="studentTip"/>

<xs:complexType name="osebaTip">
  <xs:sequence>
    <xs:element name="ime" type="xs:string"/>
    <xs:element name="priimek" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="studentTip">
  <xs:complexContent>
    <xs:extension base="osebaTip">
      <xs:sequence>
        <xs:element name="vp_st" type="xs:integer"/>
        <xs:element name="kraj" type="xs:string"/>
      </xs:sequence>
```

```
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Dokument XML -->
<student>
        <ime>Jan</ime>
        <priimek>Mizar</priimek>
        <vp_st>123123</vp_st>
        <kraj>Ljubljana</kraj>
</student>
```

Razširitev podatkovnega tipa osebaTip v studentTip

Indikatorji

S pomočjo indikatorjev (angl. indicators), lahko nadziramo kako so elementi uporabljeni. Določimo lahko njihov vrstni red, število minimalnih in maksimalnih pojavitev elementa itd. Poznamo sedem različnih indikatorjev [11]:

- all,
- choice,
- sequence,
- minOccurs,
- maxOccurs,
- Group name,
- attributeGroup name.

Uporabo in pomen indikaterjev bomo pregledali s pomočjo zgledov.

```
<!-- Shema XML -->
<xs:element name="avto">
  <xs:complexType>
```

```
    <xs:all>
      <xs:element name="znamka" type="xs:string"/>
      <xs:element name="ime" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<!-- Dokument XML -->
<!-- pravilno -->
<avto>
        <znamka>Audi</znamka>
        <ime>A5</ime>
</avto>

<avto>
        <ime>A5</ime>
        <znamka>Audi</znamka>
</avto>
```

Indikator `<all>`

Indikator `<all>` določa, da se podrejeni elementi lahko pojavijo v poljubnem vrstnem redu (samo enkrat).

```
<!-- Shema XML -->
<xs:element name="avto">
  <xs:complexType>
    <xs:choice>
      <xs:element name="bencin" type="xs:string"/>
      <xs:element name="diesel" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<!-- Dokument XML -->
```

```
<!-- pravilno -->
<avto>
        <bencin>Audi</bencin>
</avto>

<avto>
        <diesel>Audi</bencin>
</avto>
<!-- nepravilno -->
<avto>
        <bencin>Audi</bencin>
        <diesel>BMW</diesel>
</avto>
```

Indikator `<choice>`

Indikator `<choice>` nam določa, tako kot že ime pove, da moramo izbrati samo en podrejeni element.

```
<!-- Shema XML -->
<xs:element name="avto">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="znamka" type="xs:string"/>
      <xs:element name="ime" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Dokument XML -->
<!-- pravilno -->
<avto>
        <znamka>Audi</znamka>
        <ime>A5</ime>
</avto>
```

```
<!-- nepravilno -->
<avto>
        <ime>A5</ime>
        <znamka>Audi</znamka>
</avto>
```

Indikator <sequence>

Ko uporabimo indikator <sequence>, se morajo podrejeni elementi pojaviti v enakem vrstnem redu, kot se v shemi.

```
<!-- Shema XML -->
<xs:element name="narocilo">
  <xs:complexType>
    <xs:sequence>
         <xs:element name="ime" type="xs:string"/>
         <xs:element name="izdelek" type="xs:string"
     maxOccurs="unbounded" minOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Dokument XML -->
<!-- pravilno -->
<narocilo>
        <ime>Tim Gorjak</ime>
        <izdelek>Hladilna torba</izdelek>
        <izdelek>Kosilnica</izdelek>
</narocilo>
<!-- nepravilno -->
<narocilo>
        <ime>Tim</ime>
        <priimek>Mito</priimek>
</narocilo>
```

Indikatorja <minOccurs> in <maxOccurs>

Z indikatorjema `<minOccurs>` in `<maxOccurs>` lahko določimo število pojavitev elementov. `MinOccurs` uporabimo, ko želimo predpisati minimalno število pojavitev tega elementa. Če želimo predpisati največje dovoljeno število pojavitev nekega elementa, uporabimo `maxOccurs`. Unbounded pomeni, da zgornja meja ni določena (element se lahko pojavi v neomejenem številu).

```
<!-- Shema XML -->
<xs:group name="oseba">
  <xs:sequence>
    <xs:element name="ime" type="xs:string"/>
    <xs:element name="priimek" type="xs:string"/>
  </xs:sequence>
</xs:group>

<xs:element name="student" type="studentInfo"/>

<xs:complexType name="studentInfo">
  <xs:sequence>
    <xs:group ref="oseba"/>
    <xs:element name="vpisna_st" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>

<!-- Dokument XML -->
<!-- pravilno -->
<student>
        <ime>Marko</ime>
        <priimek>Skace</priimek>
        <vpisna_st>562547</vpisna_st>
</student>
```

Indikator `<group name>`

Razlika med indikatorjema `<group name>` in `<attributeGroup>` je zelo majhna. Prvi nam omogoča, da lahko podrejene elemente združimo v skupino in jih večkrat

uporabimo ali pa celo dopolnimo. Drugi indikator ima enako možnost, samo da namesto elementov, združuje atribute.

```
<!-- Shema XML -->
<xs:attributeGroup name="osebaAtribut">
  <xs:attribute name="ime" type="xs:string"/>
  <xs:attribute name="priimek" type="xs:string"/>
</xs:attributeGroup>

<xs:element name="student">
  <xs:complexType>
    <xs:attributeGroup ref="osebaAtribut"/>
        <xs:element name="vpisna_st"
        type="xs:integer"/>
  </xs:complexType>
</xs:element>

<!-- Dokument XML -->
<!-- pravilno -->
<student ime="Marko" priimek="Skace">
        <vpisna_st>895412</vpisna_st>
</student>
```

Indikator `<attributeGroup>`

2.7 Splošno o XSL

XSL je angleška kratica za e**X**tensible **S**tylesheet **L**anguage. To je slogovni jezik, ki je namenjen oblikovanju dokumentov XML. XSL je sestavljen iz treh delov [12]:

- XSL-FO/XSL
- XPath
- XSLT

2.7.1 XSL-FO/XSL

XSL-FO je kratica za e**X**tensible **S**tylesheet **L**anguage **F**ormatting **O**bjects. Temelji na XML-ju in je namenjen, tako kot že samo ime pove, oblikovanju podatkov iz dokumentov XML. XSL-FO se uradno imenuje tudi XSL [12].

2.7.2 XPath

XPath se uporablja za navigacijo, preko elementov in atributov, znotraj dokumentov XML. Uporablja podobne izraze, ki jih srečamo pri delu s tradicionalnim datotečnim računalnikom. XPath je zelo pomemben člen XSLT standarda. Brez njegove pomoči ne bi mogli ustvariti dokumentov XSLT.

2.7.3 XSLT

XSLT je angleška kratica za e**X**tensible **S**tylesheet **L**anguage **T**ransformations. Temelji na XML-ju in nam omogoča preoblikovanje dokumenta XML v poljuben format, ki je prepoznan s strani brskalnika. Največkrat se uporabi za preoblikovanje v XHTML, HTML ali pa XML. Sestavni del XSLT-ja je XPath, ki se uporablja za navigacijo preko elementov in atributov v dokumentih XML. XSLT je najpomembnejši del XSL-ja, zato se bomo v nadaljevanju osredotočili predvsem nanj.

2.8 Pravila XSLT

DTD in shemo XML lahko definiramo znotraj dokumenta XML ali pa v ločeni datoteki. Temu pri XSLTju ni tako. Nujno je potrebno ustvariti dodatno datoteko, ki jo ustrezno deklariramo in ima končnico `.xsl`. Na voljo imamo dve možnosti kako deklarirati XSLT datoteko [12]. Med eno in drugo deklaracijo ni popolnoma nobene razlike.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- ALI -->
<xsl:transform version="1.0"
```

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Deklaracija XSLT

Ker sta `<xsl:stylesheet>` oz. `<xsl:transform>` korenska elementa dokumenta XSLT, ju na koncu ne smemo pozabiti zaključit `</xsl:stylesheet>` oz. `</xsl:transform>`. Poleg deklaracije dokumenta XSLT, je potrebna tudi deklaracija znotraj dokumenta XML.

```
<?xml-stylesheet type="text/xsl"
href="imeXSLTdatoteke.xsl"?>
```

Deklaracija XSLT v dokumentu XML

Skozi naslednjih nekaj zgledov bomo spoznali kaj nam vse omogoča XSLT in kako to dosežemo. Vzeli bomo naslednji zgled datoteke XML z imenom `primer.xml`, v kateri je opis izdelkov (`znamka`, `ime`, `cena`) in na njej izvajali različne XSLT ukaze z datoteko `zgled.xsl`, ki jo bomo malo dopolnjevali in spreminjali, da bomo dobili različne rezultate.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="zgled.xsl"?>
<trgovina>
  <izdelek kategorija="prehrana">
    <znamka>Ljubljanske mlekarne</znamka>
    <ime>Jogurt 1.6 MM</ime>
    <cena>0,95</cena>
  </izdelek>
  <izdelek kategorija="pijaca">
    <znamka>Pivovarna Lasko</znamka>
    <ime>Pivo Zlatorog Klub</ime>
    <cena>1,22</cena>
  </izdelek>
  <izdelek kategorija="prehrana">
    <znamka>Ljubljanske mlekarne</znamka>
```

```
    <ime>Mleko 3.2 MM</ime>
    <cena>0,82</cena>
  </izdelek>
  <izdelek kategorija="pijaca">
    <znamka>Fructal</znamka>
    <ime>Sirup iz sadja, Malina</ime>
    <cena>2,50</cena>
  </izdelek>
</trgovina>
```

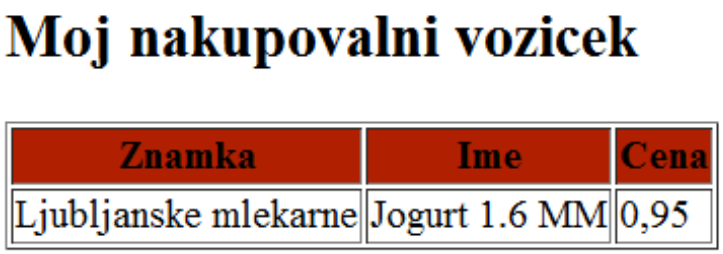
primer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<!-- prvi del dokumenta -->
 <html>
 <body>
  <h2>Moj nakupovalni vozicek</h2>
  <table border="1">
   <tr bgcolor="rgb(0, 20, 0)">
    <th>Znamka</th>
    <th>Ime</th>
    <th>Cena</th>
   </tr>
   <!-- drugi del dokumenta -->
   <tr>
    <td>
<xsl:value-of select=
"trgovina/izdelek/znamka"/></td>
    <td>
<xsl:value-of select=
"trgovina/izdelek/ime"/></td>
    <td>
```

```
<xsl:value-of select=
"trgovina/izdelek/cena"/></td>
   </tr>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Value-of zgled.xsl

Rezultat value-of izraza vidimo na sliki 2.1.



Slika 2.1: Uporaba value-of elementa.

V prvem delu dokumenta `zgled.xsl` smo opisali HTML tabelo, določili barvo ozadja in imena stolpcev (`znamka`, `ime`, `cena`). V drugem delu smo uporabili value-of element. Omenjeni element se uporabi za pridobitev vrednosti elementa XML in ga doda na izhodni tok. Izraz `select`, ki sledi value-of elementu je XPath izraz s pomočjo katerega izberemo pot do našega elementa XML, ki ga želimo izpisati. Element value-of izbere samo prvo pojavitev omenjenega elementa in ga izpiše. Če želimo izpisati vse pojavitve omenjenih elementov moramo, tako kot pri programskih jezikih, uporabiti zanko. V XSLT-ju se omenjena zanka imenuje for-each. Prvi del datoteke `zgled.xsl` se ne bo spremenil, zato bomo napisali samo drugi del, stavek for-each.

```
<!-- drugi del dokumenta -->
<xsl:for-each select="trgovina/izdelek">
```

```
 <tr>
  <td><xsl:value-of select="znamka"/></td>
  <td><xsl:value-of select="ime"/></td>
  <td><xsl:value-of select="cena"/></td>
 </tr>
</xsl:for-each>
```

for-each zgled.xsl

Rezultat, for-each stavka iz datoteke `zgled.xsl` in dokumenta `primer.xml`, lahko vidimo na sliki 2.2.

Moj celotni nakupovalni vozicek

Znamka	Ime	Cena
Ljubljanske mlekarne	Jogurt 1.6 MM	0,95
Pivovarna Lasko	Pivo Zlatorog Klub	1,22
Ljubljanske mlekarne	Mleko 3.2 MM	0,82
Fructal	Sirup iz sadja, Malina	2,50

Slika 2.2: Uporaba for-each elementa.

XSLT pozna tudi element `sort`, s pomočjo katerega uredimo poljuben XML element. Sort vstavimo takoj za elementom for-each, ter mu povemo po katerem XML elementu naj uredi izpis. Sort zna urediti tako črke, kot številke. V našem primeru smo izbrali, da uredi izpise glede na znamko.

```
<!-- drugi del dokumenta -->
<xsl:for-each select="trgovina/izdelek">
<xsl:sort select="znamka"/>
 <tr>
  <td><xsl:value-of select="znamka"/></td>
  <td><xsl:value-of select="ime"/></td>
  <td><xsl:value-of select="cena"/></td>
```

```
 </tr>
</xsl:for-each>
```

sort zgled.xsl

Rezultat, for-each s sort in dokumenta `primer.xml`, lahko vidimo na sliki 2.3.

Moj celotni nakupovalni vozicek

Znamka	Ime	Cena
Fructal	Sirup iz sadja, Malina	2,50
Ljubljanske mlekarne	Jogurt 1.6 MM	0,95
Ljubljanske mlekarne	Mleko 3.2 MM	0,82
Pivovarna Lasko	Pivo Zlatorog Klub	1,22

Slika 2.3: Uporaba for-each elementa, s sort na elementu znamka.

2.9 XLink in XPointer

XLink je okrajšava za XML Linking Language. Njegova naloga je ustvarjanje hiperpovezave v dokumentih XML [6]. Znotraj HTML-ja imamo element `<a>`, za katerega vemo, da definira hiperpovezavo. Ker pa imajo elementi znotraj XML-ja poljubna imena, ni nujno da bo, tako kot pri HTML-ju, imel element `<a>` hiperpovezavo. Znotraj XML-ja ima lahko poljuben element hiperpovezavo, samo definirati jo moramo s pomočjo XLinka.
Torej, kako definiramo hiperpovezavo v XML-ju ? Prvo moramo v začetku dokumenta deklarirati XLink imenski prostor s katerim pridobimo vse funkcionalnosti XLinka. Potem pa lahko uporabimo XLink na poljubnem XML elementu in s tem ustvarimo hiperpovezavo. V našem naslednjem zgledu bomo uporabili enostavno hiperpovezavo na spletno stran http://www.stran.si na elementu z imenom `<link>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<stran xmlns:xlink="http://www.w3.org/1999/xlink">

<link xlink:type="simple"
xlink:href="http://www.stran.si">Stran</link>

</stran>
```

Primer hiperpovezave s pomočjo XLinka

S pomočjo XLinka lahko ustvarjamo hiperpovezave na celoten dokument, ne moremo pa ustvariti povezavo na določen del dokumenta. To nam omogoča XPointer. XPointer je kratica za XML Pointer Language in nam, tako kot že omenjeno, omogoča ustvariti hiperpovezavo na določen del dokumenta s pomočjo ID-ja. Pri naslednjem zgledu nam XPointer ustvari hiperpovezavo na element z ID-jem `omeni`, ki se nahaja na spletni strani http://www.stran.si.

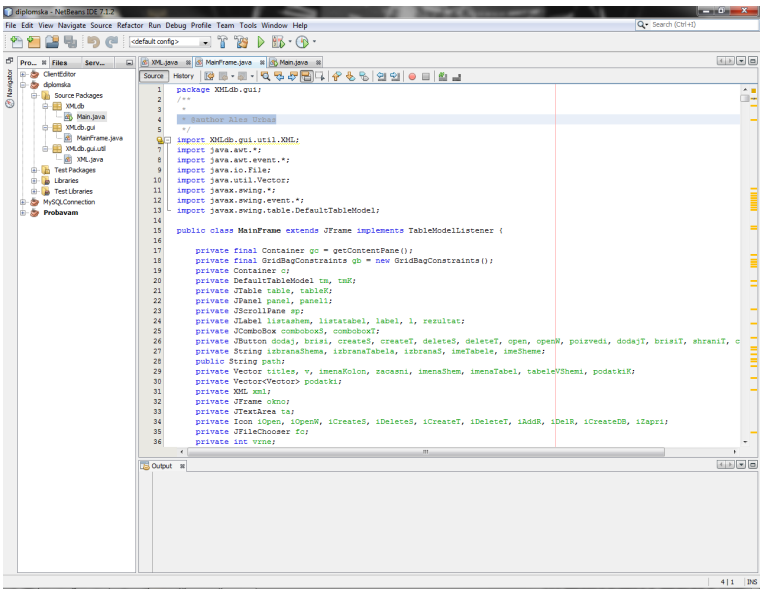
```
<?xml version="1.0" encoding="UTF-8"?>
<Igralci xmlns:xlink="http://www.w3.org/1999/xlink">
<Igralec xlink:type="simple"
 xlink:href=
 "http://www.stran.si/Igralci.xml#Center">
 <opis xlink:type="simple"
 xlink:href=
 "http://www.stran.si/Igralci/Center.gif">
 Centri so zelo robustni igralci.
 </opis>
</Igralec>
<Igralec xlink:type="simple"
 xlink:href="http://www.stran.si/Igralci.xml#Krilo">
 <opis xlink:type="simple"
 xlink:href="http://www.stran.si/Igralci/Krilo.gif">
 Krila so nekoliko nizja od centrov.
 </opis>
</Igralec>
```

```
</Igralci>
```

Primer hiperpovezave s pomočjo XPointerja in XLinka

2.10 NetBeans

NetBeans [5] je brezplačno, večkrat nagrajeno, odprtokodno integrirano razvojno orodje (IDE) za razvijalce programske opreme. NetBeans se je začel kot študentski projekt, ki se je prvotno imenoval Xelfi, na Češkem leta 1996. Sedaj je na voljo v Windows, Mac, Linux in Solaris okolju. Glavni programski jezik v razvojnem okolju NetBeans je Java, sledijo ji PHP, Groovy, C in še nekateri drugi. Z njegovo pomočjo je mogoče razviti vse od namiznih do spletnih in mobilnih aplikacij, apletov itd.



Slika 2.4: Izgled projekta v NetBeans razvojnem okolju.

2.11 Java

Java [7] je programski jezik in računalniška platforma. Razvilo jo je podjetje Sun Microsystems leta 1995. Velja za visoko nivojski programski jezik, saj jo človek

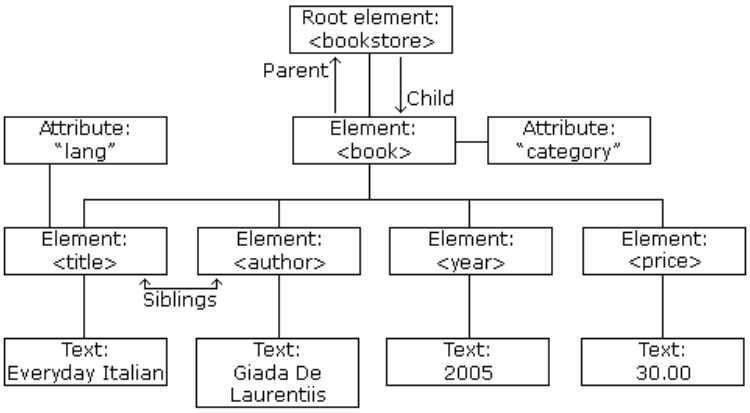
lahko preprosto bere in piše. Tako kot vsi programski jeziki ima tudi java nekaj pravil, ki se imenujejo sintaksa. Enkrat ko je program napisan, se s pomočjo Java Virtual Machine (JVM) programska koda prevede v računalniku razumljiv jezik. Poleg vsega naštetega, pa je Java še:

- arhitekturno neodvisna in prenosna,
- preprosta in objektno orientirana,
- robustna in varna,
- zanesljiva,
- visoko zmogljiva.

Dandanes Java teče na več kot 850 milijonih osebnih računalnikih in na več milijardah naprav po vsem svetu [7].

2.12 Dom

DOM je W3C standard, ki opredeljuje dostop do dokumentov, kot sta HTML in XML [3]. Je vmesnik uporabniškega programa (API) in je namenjen za uporabo z vsemi programskimi jeziki. DOM programerjem omogoča ustvarjanje, spreminjanje in brisanje tako dokumentov, kot elementov in njihove vsebine znotraj samega dokumenta. DOM vidi vsebino dokumentov, kot drevesno strukturo imenovano DOM tree. Primer drevesne strukture [3] je prikazan na sliki 2.5. S pomočjo DOM-a se v našem programu sprehajamo po vsebini XML dokumentov.



Slika 2.5: DOM drevesna struktura.

Poglavje 3

Opis razvitega programa

Za zapis ali branje velike večine podatkovnih baz potrebujemo nameščeno ustrezno programsko opremo, povezavo do strežnika itd. Da bi se izognili tem nevšečnostim in raznim možnim zapletom (povezava ne deluje, strežnik ni dosegljiv, nimamo pravic za namestitev ustrezne programske opreme itd.), smo razvili program, ki ne potrebuje nič od naštetega. Vse kar potrebujemo za branje in zapisovanje podatkovne baze je ustrezna XML datoteka. Tako, kot smo že ugotovili v poglavju 2.1, je format XML zares idealen za zapis podatkov in zato posledično podatkovne baze. Z našim razvitim programom enostavno odpremo datoteko formata `.xml` in že lahko pregledujemo sheme, tabele in zapise znotraj tabel. Program nam omogoča urejanje zapisov znotraj tabel, dodajanje novih vrstic, brisanje obstoječih, ter ustvarjanje shem, tabel in brisanje le-teh. Vsi urejeni podatki se takoj zapišejo v našo XML podatkovno bazo, tako da tudi v primeru izpada električne energije ne izgubimo na novo spremenjenih podatkov.
V nadaljevanju si bomo bolj natančno pogledali, kako smo razvili program, kakšne datoteke XML prebira in zapisuje, algoritme programa, podatkovne strukture, ter kaj vse smo morali storiti, da je program dobil končno funkcionalnost.

3.1 Zgradba XML datoteke

Za kasnejše razumevanje algoritmov programa si moramo prvo pogledati samo zgradbo XML datotek, ki služijo kot podatkovna baza. Datoteke XML morajo biti pravilno oblikovane, tako kot je to zapisano v poglavju 2.2.6. Če datoteka ni

pravilno oblikovana, naš program ni zmožen obdelave omenjene datoteke in zato javi napako.

Sestavo baze si bomo pogledali na zgledu `Baza.xml`. Iz poglavja 2.2.3 vemo, da mora vsak XML dokument vsebovati korenski element. V našem primeru je korenski element `<bazaXML>`. Element, ki sledi korenskemu elementu je ime prve sheme. V našem zgledu je to element `<NBA>`, kar pomeni da je ime prve in edine sheme v naši bazi `NBA`. Elementa, ki sta neposredno podrejena (angl. child elements) elementu `<NBA>`, sta `<TRENERJI>` in `<IGRALCI>`. Ta dva elementa sta imeni obeh tabel, ki se nahajata znotraj sheme `NBA`. Poglejmo si tabelo `IGRALCI`, ter kako so zapisi shranjeni v dokumentu XML. Tabela ima dva neposredno podrejena elementa, oba z imenom `IGRALCI1`. To pomeni, da imamo dva zapisa (dve vrstici) v tabeli `IGRALCI`. Če pogledamo podrejene elemente elementa `<IGRALCI1>`, opazimo da so popolnoma enaki elementi v obeh zapisih: `<IME>`, `<PRIIMEK>`, `<VISINA>`, `<POZICIJA>`. Te elementi so stolpci tabele. Besedilo, ki se nahajajo znotraj omenjenih elementov, pa so zapisi znotraj tabele prikazani na sliki 3.1.

```
<BazaXML>
<NBA>
<TRENERJI>
<TRENERJI1>
<IME>Zmago</IME>
<PRIIMEK>Sagadin</PRIIMEK>
<STAROST>60</STAROST>
</TRENERJI1>
<TRENERJI1>
<IME>Erik</IME>
<PRIIMEK>Spoelstra</PRIIMEK>
<STAROST>32</STAROST>
</TRENERJI1>
</TRENERJI>
<IGRALCI>
<IGRALCI1>
<IME>LeBron</IME>
<PRIIMEK>James</PRIIMEK>
```

```
<VISINA>203</VISINA>
<POZICIJA>SF</POZICIJA>
</IGRALCI1>
<IGRALCI1>
<IME>Beno</IME>
<PRIIMEK>Udrih</PRIIMEK>
<VISINA>186</VISINA>
<POZICIJA>PG</POZICIJA>
</IGRALCI1>
</IGRALCI>
</NBA>
</BazaXML>
```

Baza.xml

IME	PRIIMEK	VISINA	POZICIJA
LeBron	James	203	SF
Beno	Udrih	186	PG

Slika 3.1: Zapisi v tabeli `IGRALCI`.

Sedaj, ko razumemo zgradbo dokumentov XML si lahko bolj podrobno pogledamo samo delovanje programa in njegove algoritme.

3.2 Algoritmi in podatkovne strukture

V naslednjih podpoglavjih si bomo pogledali "ozadje"programa, opisali algoritme in podatkovne strukture, kaj naredi program ko odpremo datoteko XML, kako vstavimo vrstico v tabelo itd. Prvo pa si poglejmo vse tri razrede (angl. class) razvitega programa:

- `MainFrame.java`
  Glavno okno programa, ki skrbi za izris vseh vidnih komponent. V razredu `MainFrame` so prav tako definirani vsi poslušalci gumbov, tabel, spustnih seznamov itd.

- `XML.java`
  Glavni razred za upravljanje z datotekami tipa `.xml`.

- `Main.java`
  Razred, ki je zadolžen, da se ob odprtju programa ustvari nov objekt razreda `MainFrame`.

3.2.1 Odpiranje in ustvarjanje podatkovne baze XML

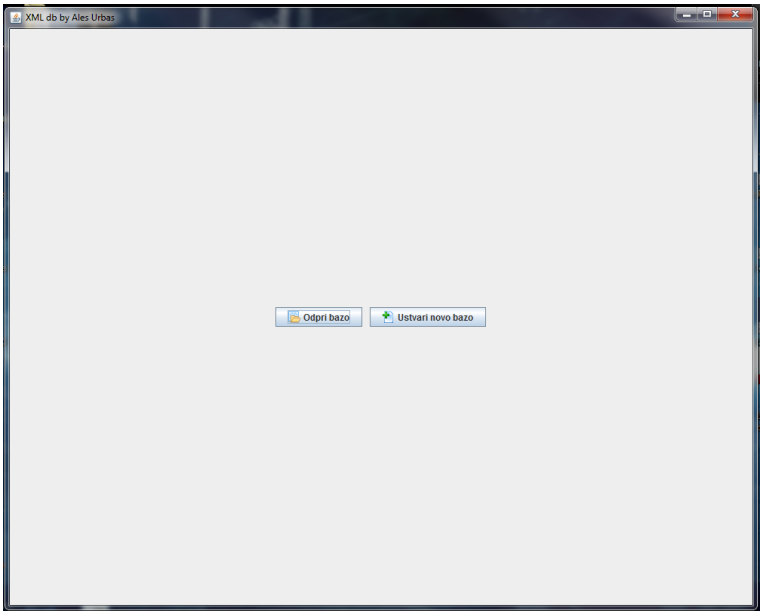
Takoj, ko zaženemo naš program, se zažene main metoda v razredu `Main.java`, ki ustvari nov objekt razreda `MainFrame.java`. Ta naloži okno, ter vse njegove komponente in poslušalce za te komponente. Takoj za tem, skrijemo vse komponente, ter prikaže samo dva gumba: `Odpri bazo` in `Ustvari novo bazo`, prikazano na sliki 3.2. Klik na en ali pa drug gumb naredi akciji, ki se razlikujeta samo v prvem delu. Če kliknemo na gumb `Odpri bazo`, se nam odpre okno prikazano na sliki 3.3. Koda s katero smo dosegli to:

```
JFileChooser fc = new JFileChooser();
int vrne = fc.showOpenDialog(this);
```

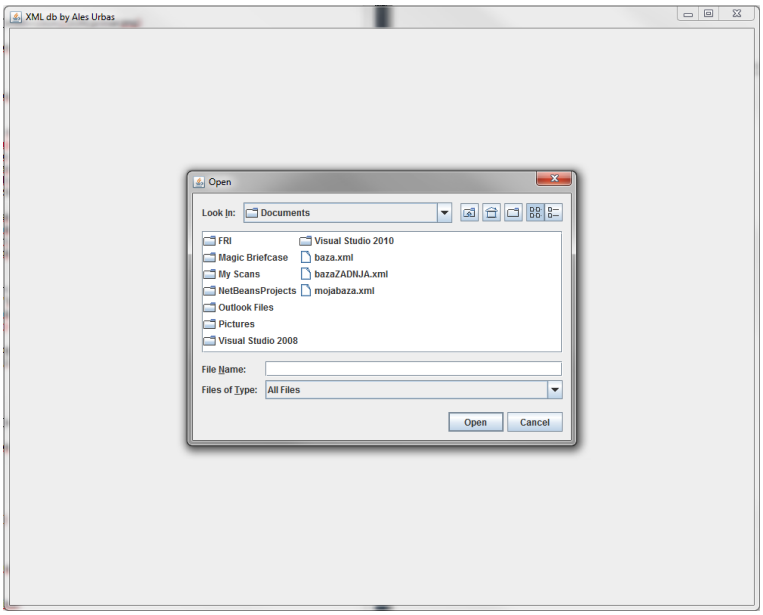
Če pa kliknemo na gumb `Ustvari novo bazo` se nam odpre enako okno, samo da bo sedaj to okno ustvarilo novo datoteko, ne pa odprlo obstoječe. Programska koda izgleda tako:

```
JFileChooser fc = new JFileChooser();
int vrne = fc.showSaveDialog(this);
```

Ko ustvarimo novo bazo, moramo še preveriti če datoteka s tem imenom že obstaja. Prav tako moramo avtomatsko dodati končnico `.xml`, če uporabnik tega ni storil. To storimo z naslednjo programsko kodo:



Slika 3.2: Prikaz uvodnega okna programa.



Slika 3.3: Okno programa po kliku na gumb `Odpri bazo`.

```
File file = fc.getSelectedFile();
path = file.getAbsoluteFile().toString();
String ext = ".xml";
if (!path.endsWith(ext)) {
        file = new File(path + ext);
        path += ext;
}
if (file.exists()) {
        Object[] options = {"Da!", "Ne!"};
        izbira = JOptionPane.showOptionDialog(okno,
"Datoteka ze obstaja. Ali jo zelite prepisati?",
"Opozorilo!",
JOptionPane.DEFAULT_OPTION,
JOptionPane.WARNING_MESSAGE, null, options,
options[0]);
        System.out.println("Izbira: "+izbira);
}
```

Del programske kode pri ustvarjanju nove baze

Prišli smo do koraka, ko smo ustvarili novo bazo ali pa odpri obstoječo. Programska koda je od tukaj naprej za oba koraka enaka, poimenovali jo bomo kar skupna koda. Najboljše bo, da si pogledamo programsko kodo (malenkost je drugačna od originalne, zaradi lažjega razumevanja), katero bomo razložili po korakih.

```
1  XML xml=new XML(file, this);
2  Vector iShem=xml.getShemaNames();
3  String iSheme=iShem.firstElement().toString();
4  nastaviCBS(iShem);
5  Vector iTabel=xml.getTableNames(iSheme);
6  String iTabele=iTabel.firstElement().toString();
7  nastaviCBT(imenaTabel);
8  int stStolpcev=xml.stStolpcev(iSheme,iTabele);
9  iStolpcev=xml.iStolpcev(iSheme,iTabele,stStolpcev);
10 podatki=new Vector<Vector>(stStolpcev);
```

```
11 podatki=xml.iElementov(iSheme,iTabele,stStolpcev);
12 nastaviTabelo(podatki,iStolpcev);
```

Skupna koda

Na začetku kode ustvarimo nov objekt razreda `XML.java`, kateremu v parametrih podamo odprt dokument tipa `File`, ter trenutno odprto okno tipa `MainFrame`. Konstruktor razreda `XML.java` izgleda tako:

```
1  public XML(File xml1, MainFrame okno) {
2   xml = xml1;
3   okn = okno;
4   try {
5    dbf = DocumentBuilderFactory.newInstance();
6    db = dbf.newDocumentBuilder();
7    doc = db.parse(xml);
8    doc.getDocumentElement().normalize();
9   } catch (Exception e) {
10   JOptionPane.showMessageDialog(okn,
11   "Napaka: "+e,
12   "Napaka!",
13   JOptionPane.ERROR_MESSAGE);
14  }
15 }
```

Konstruktor razreda `XML.java`

V konstruktorju so najpomembnejše vrstice 5, 6 in 7. V teh vrsticah razčlenimo podano datoteko na DOM drevesno strukturo, katero bomo kasneje uporabljali za urejanje našega dokumenta XML.

Poglejmo si 2. vrstico v `Skupni kodi`:

```
Vector iShem=xml.getShemaNames();
```

V tej vrstici kličemo metodo `getShemaNames()` razreda `XML`, ki v obliki vectorja vrne imena vseh shem, ki se nahajajo v izbrani datoteki. Zapis metode izgleda tako:

```
1  public Vector getShemaNames() {
2   root = doc.getDocumentElement();
3   nl = root.getChildNodes();
4   imenaShem = new Vector(3, 1);
5   for (int i = 0; i < nl.getLength(); i++) {
6    node = nl.item(i);
7    if (node.getNodeType() == 1)
8     imenaShem.add(node.getNodeName());
9   }
10  return imenaShem;
11 }
```

Metoda `getShemaNames()` razreda `XML`

Algoritem je preprost:

- Pridobimo ime korenskega elementa (2. vrstica).
- Pridobimo vse njegove podrejene elemente (3. vrstica).
- Ustvarimo vektor `imenaShem` (4. vrstica).
- V `for` zanki se sprehodimo čez vse podrejene elemente (5. vrstica) in jih zapisujemo v vektor `imenaShem` (8. vrstica).
- Vrnemo vektor `imenaShem` (10. vrstica).

Sedaj smo v vektor `iShem` zapisali imena vseh shem, ki se nahajajo v našem odprtem dokumentu. Vrednost prvega elementa, ki se nahaja v vektorju `iShem` zapišemo v spremenljivko tipa `String` z imenom `iSheme`. To je korak v 3. vrstici `Skupne kode`:

```
String iSheme=iShem.firstElement().toString();
```

Zakaj smo to naredili, bomo videli čez nekaj korakov.

Poglejmo si naslednjo vrstico kode:

```
nastaviCBS(iShem);
```

Klic metode `nastaviCBS`, ki okrajšano pomeni nastavi combobox (slo. spustni seznam) shem, naredi točno to kar ime samo pove: Nastavi omenjeni spustni seznam z vektorji, ki so podani v parametru klica (`iShem`). Na sliki 3.4 je to spustni seznam, z napisom "`APP`".

Poglejmo si 5. vrstico iz `Skupne kode`:

```
Vector iTabel=xml.getTableNames(iSheme);
```

Kličemo metodo `getTableNames(iSheme)` iz razreda `XML`. Spomnimo se, da je v spremenljivki iSheme zapisana prva shema v naši podatkovni bazi. Metoda izgleda tako:

```
1  public Vector getTableNames(String shema) {
2   shema1 = shema.toUpperCase();
3   nlTable = doc.getElementsByTagName(shema1);
4   root = doc.getDocumentElement();
5    for(int i = 0; i < nlTable.getLength(); i++) {
6     node = nlTable.item(i);
7     if(node.getParentNode().getNodeName().equals(
8     root.getNodeName()))
9     nodePravi = node;
10   }
```

```
11  nlTable = nodePravi.getChildNodes();
12  imenaTabel = new Vector(10,2);
13  for (int i = 0; i < nlTable.getLength(); i++) {
14   child = nlTable.item(i);
15   if (child.getNodeType() == 1)
16   imenaTabel.add(child.getNodeName().toUpperCase());
17  }
18  return imenaTabel;
19 }
```

Metoda `getTableNames(String shema)` razreda `XML`

Metoda najprej pridobi vse element z imenom sheme (3. vrstica). V 4. vrstici določimo korenski element dokumenta XML. Ker je lahko ime sheme enako imenu tabele, moramo to preveriti. To naredimo s pomočjo `for` zanke, kjer se prepričamo, da je izbrani element zares ime sheme in ne ime tabele (vrstice 5-10). Potem pridobimo vse podrejene elemente izbrane sheme (11. vrstica), ter se sprehodimo čez njih v `for` zanki (13. vrstica) in uspešno zapisujemo imena tabel v vektor `imenaTabel` (16. vrstica). Sproti še v `if` stavku (15. vrstica) preverjamo, če je omenjeni element zares element, ne pa prazen element z napisom `#TEXT`. Na koncu metode vrnemo omenjeni vektor (18. vrstica).

Sedaj smo v vektor `iTabel` zapisali imena vseh tabel, ki se nahajajo v naši prvi shemi dokumenta XML. Vrednost prvega elementa, ki se nahaja v vektorju `iTabel`, zapišemo v spremenljivko tipa `String` z imenom `iTabele`. To je korak v 6. vrstici `Skupne kode`:

```
String iTabele=iTabel.firstElement().toString();
```

7. vrstica `Skupne kode` naredi popolnoma enako kot 4. vrstica, samo da tokrat napolnimo spustni seznam (angl. combobox) tabel z vektorji vseh tabel, ki se nahajajo v naši prvi shemi.

```
nastaviCBT(imenaTabel);
```

Vrstica, ki se nahaja pod zaporedno številko 8 v `Skupni kodi` izgleda tako:

```
int stStolpcev=xml.stStolpcev(iSheme,iTabele);
```

Ponovno se kliče metoda razreda `XML`, tokrat `stStolpcev(iSheme,iTabele)`. Poglejmo si pobližje delovanje omenjene metode:

```
1  public int stStolpcev(String iSheme,String iTabele){
2   imeSheme1 = iSheme.toUpperCase();
3   imeTabele1 = iTabele.toUpperCase()
4   nlTable = doc.getElementsByTagName(imeTabele1);
5   for(int i = 0; i < nlTable.getLength(); i++) {
6    node = nlTable.item(i);
7    parent = node.getParentNode();
8    if(parent.getNodeName().equals(imeSheme1))
9     nodePravi = node;
10  }
11  nlTable = nodePravi.getChildNodes();
12  elementNumber = nlTable.getLength();
13  stZapisov = 0;
14  stVrstic = 0;
15  for (i = 0; i < elementNumber; i++) {
16   child = nlTable.item(i);
17   if (child.getNodeType() == 1) {
18    nlTemp = child.getChildNodes();
19    childNumber = nlTemp.getLength();
20    stVrstic++;
21    for (j = 0; j < childNumber; j++) {
22     childchild = nlTemp.item(j);
23     if (childchild.getNodeType() == 1)
```

```
24          stZapisov++;
25        }
26       }
27     }
28 stStolpcev=stZapisov/stVrstic;
29 return stStolpcev;
30 }
```

Metoda `stStolpcev(String iSheme,String iTabele)` razreda `XML`

Najprej v metodi `stStolpcev` pogledamo vse elemente, kateri ime imajo enak `imeTabele1` (4. vrstica). Potem se sprehodimo čez vse te elemente in izberemo tistega, čigar nadrejeno vozlišče (angl. parent node) je enak imenu sheme (vrstice od 5-10). Ta korak moramo storiti, saj bi lahko v primeru tabel z enakim imenom v dveh različnih shemah prišlo do težav, ker bi označili napačno vozlišče. V korakih ki sledijo, se sprehodimo čez vse zapise v tabeli in povečujemo števec `stZapisov` oz. `stVrstic`. Koda je na las podobna kodi `getTableNames`. Za obnovitev spomina, se spomnimo datoteke `Baza.xml` v poglavju 3.1. Ko imamo enkrat `stZapisov` in `stVrstic` moramo ti dve števili samo še deliti med seboj. Npr. imamo 6 zapisov, ter 2 vrstici, kar pomeni, da imamo število stolpcev 3. Rezultat (`stStolpcev`) vrnemo v predzadnji vrstici kode.

Poglejmo si 9. vrstico `Skupne kode`. Tako izgleda koda:

```
iStolpcev=xml.iStolpcev(iSheme,iTabele,stStolpcev);
```

V tej vrstici kličemo metodo `iStolpcev` iz razreda `XML`. Koda metode izgleda tako:

```
1 public Vector iStolpcev(String iSheme,
2 String iTabele,int stStolpcev){
3 imeSheme1 = imeSheme.toUpperCase();
4 imeTabele1 = imeTabele.toUpperCase();
5 imeStolpcev = new Vector(stStolpcev);
```

```
6  nlTable = doc.getElementsByTagName(imeTabele1);
7  for (int i = 0; i < nlTable.getLength(); i++) {
8  node = nlTable.item(i);
9  parent = node.getParentNode();
10 if (parent.getNodeName().equals(imeSheme1))
11 nodePravi = node;
12 }
13 nlTable = nodePravi.getChildNodes();
14 stZapisov = 0;
15 for (i = 0; i < nlTable.getLength(); i++) {
16 child = nlTable.item(i);
17 if (child.getNodeType() == 1) {
18 nlTemp = child.getChildNodes();
19 for (j = 0; j < nlTemp.getLength(); j++) {
20 childchild = nlTemp.item(j);
21 if (childchild.getNodeType() == 1) {
22 if (stZapisov < stStolpcev){
23 imeStolpcev.add(childchild.getNodeName());
24 stZapisov++;
25 }
26 }
27 }
28 }
29 }
30 return imeStolpcev;
31 }
```

Metoda `iStolpcev(String iSheme,String iTabele,int stStolpcev)` razreda `XML`

V začetku kode preverimo, če se nahajamo v tapravi shemi in tapravi tabeli (vrstice 6-12). Kasneje se sprehodimo po DOM drevesni strukturi, vse do imen stolpcev. Dokler je število zapisov v vektorju `imeStolpcev` manjše od dejanskega števila stolpcev, zapisujemo imena stolpcev v vektor, ter povečamo števec `stZapisov` za ena (22-25. vrstica). Vrnemo (30. vrstica) vektor, ki vsebuje zapise vseh imen

stolpcev.

Vrnimo se na našo `Skupno kodo` in si poglejmo naslednjo vrstico (10):

```
podatki=new Vector<Vector>(stStolpcev);
```

V 10. vrstici ustvarimo vektor vektorjev velikosti enaki kot je stStolpcev.

Predzadnja vrstica v `Skupni kodi` je namenjena za pridobivanje podatkov, ki jih bomo kasneje uporabili za prikaz v naši tabeli.

```
podatki=xml.iElementov(iSheme,iTabele,stStolpcev);
```

Metoda `iElementov` izgleda tako:

```
1  public Vector<Vector> iElementov(String iSheme,
2  String iTabele, int stStolpcev) {
3   imeSheme1 = imeSheme.toUpperCase();
4   nlTable = doc.getElementsByTagName(imeTabele);
5   for (int i = 0; i < nlTable.getLength(); i++) {
6    node = nlTable.item(i);
7    parent = node.getParentNode();
8    if (parent.getNodeName().equals(imeSheme1))
9    nodePravi = node;
10  }
11  nlTable = nodePravi.getChildNodes();
12  stZapisov = 0;
13  podatki = new Vector<Vector>();
14  for (i = 0; i < nlTable.getLength(); i++) {
15   child = nlTable.item(i);
16   if (child.getNodeType() == 1) {
```

```
17    nlTemp = child.getChildNodes();
18    zacasni = new Vector();
19    for (j = 0; j < nlTemp.getLength(); j++) {
20     childchild = nlTemp.item(j);
21     if (childchild.getNodeType() == 1) {
22      if (stZapisov < stStolpcev) {
23       zacasni.add(childchild.getTextContent());
24       stZapisov++;
25       if (stZapisov == stStolpcev) {
26        podatki.add(zacasni);
27        stZapisov = 0;
28       }
29      }
30     }
31    }
32   }
33  }
34 return podatki;
35 }
```

Metoda `iElementov(String iSheme, String iTabele, int stStolpcev)` razreda `XML`

V zgoraj napisani metodi se, tako kot pri prejšnjih, najprej sprehodimo v `for` zanki čez elemente, ter pogledamo če se "nahajamo"v pravilni tabeli, znotraj pravilne sheme (vrstice 4-10). Metoda ima potem enake korake kot metoda `iStolpcev`. Edina razlika je, da v vrsticah od 23-27 namesto imena elementa, dobimo besedilo, ki se nahaja znotraj elementa (vrstica 23), ter preverjanja (vrstica 25) če je število zapisov enako številu stolpcev in v primeru ko to je, zapišemo vektor `zacasni` v vektor vektorjev z imenom `podatki`. V 34. vrstici metoda vrača vektor vektorjev `podatki`.

Sledi še zadnji korak v naši `Skupni kodi`, klicanje metode `nastaviTabelo` znotraj razreda `MainFrame`.

```
nastaviTabelo(podatki,iStolpcev);
```

Ko pokličemo metodo `nastaviTabelo`, želimo prikazati podatke prve tabele iz prve sheme baze. Koda metode `nastaviTabelo` izgleda tako:

```
1  public void nastaviTabelo(Vector<Vector> podatki,
2  Vector iStolpcev) {
3   tm = new DefaultTableModel(podatki, imenaStolpcev);
4   table.setModel(tm);
5   table.setVisible(true);
6   tm.addTableModelListener(new TableModelListener(){
7    public void tableChanged(TableModelEvent e) {
8     save();
9    }
10  });
11 }
```

Metoda `nastaviTabelo(Vector<Vector> podatki, Vector iStolpcev)` razreda `MainFrame.java`

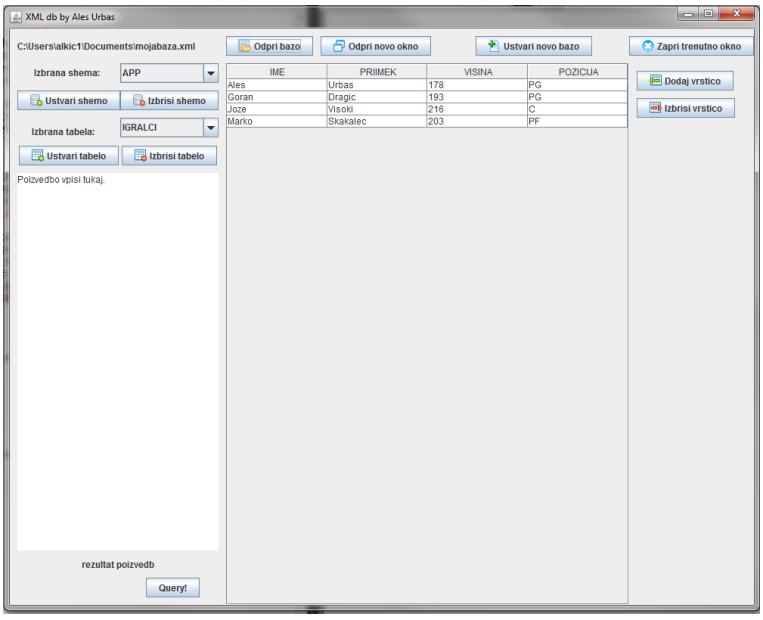
Metoda sprejme dva argumenta. Prvi je vektor vektorjev, ki vsebuje podatke. Drugi pa je vektor z imeni stolpcev. S pomočjo teh dveh argumentov ustvarimo objekt `DefaultTableModel` (3. vrstica), katerega kasneje dodamo tabeli tipa `JTable` (4. vrstica). Tabeli prav tako dodamo poslušalca (6. vrstica), kateri posluša in v primeru, da se podatki v tabeli spremenijo, to spremembo ustrezno shrani v dokument XML. Če smo bolj natančni kličemo metodo `save()`, ki najprej zbriše vse zapise znotraj dokumenta XML za tabelo v kateri se trenutno nahajamo. Kasneje pa prepišemo podatke iz tabele, ki smo jo urejali, nazaj v dokument XML. Za konec moramo še prikazati vse gradnike programa:

```
ta.setVisible(true);
poizvedi.setVisible(true);
openW.setVisible(true);
zapri.setVisible(true);
```

```
brisi.setVisible(true);
rezultat.setVisible(true);
listashem.setVisible(true);
listatabel.setVisible(true);
comboboxS.setVisible(true);
comboboxT.setVisible(true);
createS.setVisible(true);
deleteS.setVisible(true);
createT.setVisible(true);
deleteT.setVisible(true);
dodaj.setVisible(true);
sp.setVisible(true);
label.setVisible(true);
```

Prikaz vseh gradnikov v glavnem oknu

Na zadnje smo le prispeli do točke, ko uspešno odpremo datoteko XML, ter nastavimo vse potrebno za uporabo našega programa, ki je prikazan na sliki 3.4.



Slika 3.4: Glavno okno razvitega programa.

Na sliki 3.4 imamo prvo shemo z imenom `APP`, ter prvo tabelo v omenjeni shemi z imenom `IGRALCI`, katere vsebina je na sredini okna prikazana v tabeli.

3.2.2 Delovanje programa

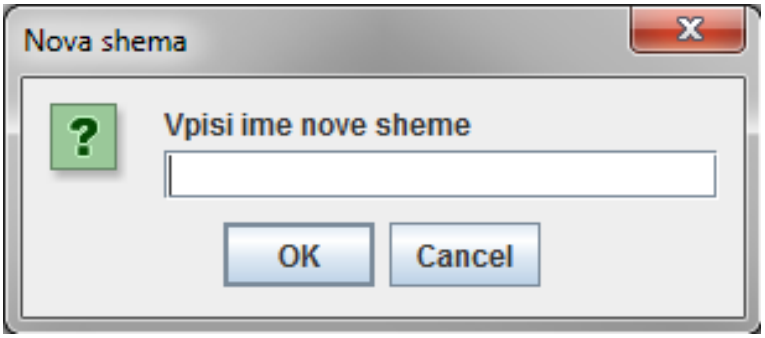
Ko enkrat uspešno odpremo podatkovno bazo ali ustvarimo novo, se nam odpre glavno okno programa (slika 3.4). Znotraj glavnega okna lahko:

- ustvarjamo sheme in tabele,
- brišemo sheme in tabele,
- pišemo poizvedbe SQL,
- dodajamo in brišemo vrstice v tabeli,
- odpremo in zapremo glavno okno,
- odpremo in ustvarimo novo podatkovno bazo.

V nadaljevanju si bomo pogledali ozadje vsake izmed naštetih možnosti, razen odpiranja in ustvarjanja nove podatkovne baze, katero smo že obdelali v poglavju 3.2.1.

Ustvarjanje shem in tabel

Najprej si poglejmo, kaj se zgodi ko pritisnemo na gumb ustvari shemo. Po kliku na gumb, se nam odpre novo okno (`JOptionPane.showInputDialog`) prikazano na sliki 3.5. V besedilno polje vpišemo željeno ime nove sheme.
S pomočjo že opisane metode `getShemaNames()` razreda `XML` pridobimo vektor vseh shem. V `for` zanki se sprehodimo čez vektor in pogledamo, če morda vpisano ime nove sheme že obstaja. Če obstaja javimo napako, saj ne moramo imeti dveh enakih imen. V primeru, da ime sheme še ne obstaja, jo ustvarimo. To naredimo tako, da kličemo metodo `ustvariNovoShemo(imeSheme)`, katera doda nov podrejeni element korenskemu elementu. Koda, kako pripnemo podrejeni element korenskemu znotraj metode, `ustvariNovoShemo(imeSheme)` izgleda tako:



Slika 3.5: Okno za vpis imena nova sheme.

```
root = doc.getDocumentElement();
el = doc.createElement(imeSheme.toUpperCase());
root.appendChild(el);
```

Ko smo uspešno ustvarili novo shemo, moramo ustvariti še našo prvo tabelo znotraj nove sheme. Ime prve tabele v novi shemi je: `DEFAULT`. Omenjena tabela vsebuje dva stolpca z imeni `PRVI` in `DRUGI`. Ko imamo ime tabele (tipa `String`) in imeni stolpcev (tipa `Vector`), pokličemo metodo `ustvariNovoTabelo` iz razreda `XML`. Vse skupaj v programski kodi izgleda tako:

```
iTabele = "DEFAULT";
iStolpcev.add("PRVI");
iStolpcev.add("DRUGI");
xml.ustvariNovoTabelo(iSheme,iTabele,iStolpcev);
```

Metoda `ustvariNovoTabelo(iSheme,iTabele,iStolpcev)` nam v dokumentu XML ustrezno naredi nov element, ki predstavlja ime tabele, ter njegove podrejene elemente, ki predstavljajo imena stolpcev. Znotraj pa vstavi besedilo ”Prazno”. Glavni del kode `ustvariNovoTabelo` izgleda tako:

```
1  StringBuffer sb = new StringBuffer(iTabele);
2  String player1 = sb.append("1").toString();
3  nlTable = doc.getElementsByTagName(iSheme);
4  shema = nlTable.item(0);
5  nodePravi = doc.createElement(iTabele);
6  shema.appendChild(nodePravi);
7  el = doc.createElement(player1);
8  nodePravi.appendChild(el);
9  for (int i = 0; i < titles.size(); i++) {
10  Element element;
11 element=doc.createElement(titles.get(i).toString());
12  el.appendChild(element);
13  element.appendChild(doc.createTextNode("Prazno"));
14 }
```

Glavni del kode metode `ustvariNovoTabelo`

Ko imamo uspešno ustvarjeno našo novo shemo, ter našo prvo tabelo, moramo v programu napraviti še naslednje korake:

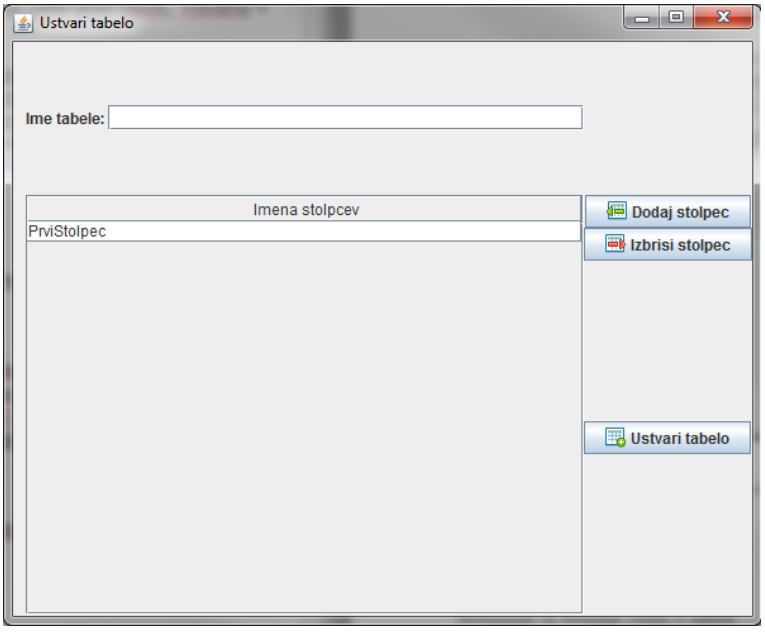
- Obnoviti spustni seznam shem (ker smo dodali novo shemo v dokument XML).
- Spustni seznam shem nastaviti na vrednost naše nove sheme, to pa avtomatsko nastavi spustni seznam tabel na našo prvo (in edino) tabelo.
- Nastaviti našo tabelo (JTable), da ustrezno prikaže podatke - stolpca `PRVI` in `DRUGI`, z zapisoma "Prazno".

Napisani koraki v programski kodi:

```
1  setShemaNames();
2  stStolpcev = xml.stStolpcev(iSheme, iTabele);
3  comboboxS.setSelectedItem(iSheme);
4  podatki = new Vector<Vector>(stStolpcev);
5  podatki=xml.imenaElementov(iSh,iTab,iSt,stSt);
6  nastaviTabelo(podatki, iStolpcev);
```

Del kode za ustvarjanje nove sheme

Sedaj, ko smo uspešno ustvarili novo shemo, si lahko pogledamo še kako ustvariti novo tabelo. Ko kliknemo na gumb `Ustvari tabelo`, se nam odpre novo okno tipa `JFrame` prikazano na sliki 3.6. V zgornje besedilno polje vpišemo ime nove tabele v spodnjo tabelo pa imena stolpcev. Z gumboma na desni strani lahko poljubno dodajamo oz. brišemo stolpce. Nov stolpec lahko dodamo tudi s pritiskom tipke DOL na tipkovnici, ko se nahajamo na zadnji vrstici tabele. Ko končamo z urejanjem pritisnemo gumb `Ustvari tabelo`. Ta sproži naslednje korake. Najprej s pomočjo `getTableNames(imeSheme)` metode pridobimo imena vseh tabel (zapisane v vektorju), čez katere se sprehodimo v `for` zanki in pregledamo če morebitno izbrano ime tabele že obstaja. Če ime obstaja, javimo napako, ter vrnemo uporabnika nazaj na okno "`Ustvari tabelo`". Če vpisano ime tabele še ne obstaja, s klicem metode `ustvariNovoTabelo(iSheme,iTabele,iStolpcev)` iz razreda `XML` ustvarimo novo tabelo. Omenjeno metodo smo že opisali v prvem delu trenutnega podpoglavja.



Slika 3.6: Prikazno okno za ustvarjanje nove tabele.

Brisanje shem in tabel

Naj na začetku podpoglavja omenimo, da pritisk na gumb `Izbrisi tabelo` oz. `Izbrisi shemo` izbrise tabelo/shemo, ki je trenutno izbrana v spustnem seznamu tabel/shem.
Brisanje shem oz. tabel ni zapletena operacija. V dokumentu XML najdemo ustrezni element (po enakem postopku kot npr. v metodi `stStolpcev`), dobimo njegov nadrejeni element, ter s klicem iz nadrejenega elementa izbrišemo njegov podrejeni (naš najden) element. V programski kodi Java naredimo to tako:

```
parent = nodePravi.getParentNode();
parent.removeChild(nodePravi);
```

Ko je tabela zadnja v izbrani shemi jo ne moramo izbrisati. Prav tako ne moramo izbrisati zadnje sheme v podatkovni bazi. To enostavno preverimo tako, da pred brisanjem sheme/tabele pogledamo število shem/tabel. Če je to enako 1, sheme/tabele ne zbrišemo.

V primeru, da brišemo tabelo moramo storiti še naslednja koraka:

- Obnoviti spustni seznam tabel (saj smo trenutno označeno izbrisali).
- V tabeli prikazati podatke iz prve tabele iz `''novega''` seznama.

Koda za omenjene korake izgleda tako (metoda `nastaviTabelo` je že razložena v prejšnjem poglavju) :

```
1 imenaTabel = xml.getTableNames(izbranaShema);
2 comboboxT.setModel(
3                 new DefaultComboBoxModel(tabele));
4 comboboxT.setVisible(true);
5 izbranaTabela=comboboxT.
6 getSelectedItem().toString();
7 int stStolpcev = xml.stStolpcev(iSheme, iTabele);
```

```
8  iStolpcev=xml.imenaStolpcev(iSheme,
9                                  iTabele,stStolpcev);
10 podatki = new Vector<Vector>(stStolpcev);
11 podatki=xml.imenaElementov(iSheme,
12                                iTabele,stStolpcev);
13 nastaviTabelo(podatki,iStolpcev);
```

Del kode za brisanje tabele

V primeru, da želimo izbrisati shemo, moramo poleg dejanskega brisanja sheme znotraj dokumenta XML, narediti še naslednje:

- Obnoviti spustni seznam shem (saj smo trenutno označeno izbrisali).
- V tabeli prikazati podatke iz prve tabele iz prve sheme `''novega''` seznama.

Poizvedbe SQL

Na levi strani glavnega okna programa (slika 3.4) se nahaja gradnik imenovan besedilno območje (angl. text area). Ta gradnik služi za poizvedovanje SQL stavkov. Najprej si bomo pogledali poizvedbe SQL, njihove konkretne primere skupaj z opisom, kasneje pa spoznali "ozadje"poizvedb in algoritme, ki omogočajo poizvedbe.

1. `SELECT * FROM imeTabele`
   Primer: `SELECT * FROM Igralci`
   Opis: Prikaže vse podatke za tabelo `Igralci`.

2. `SELECT * FROM imeTabele WHERE stolpec=vrednost`
   Primer: `SELECT * FROM Igralci WHERE ime=Ales`
   Opis: Prikaže vse zapise v tabeli `Igralci`, katerih stolpec `ime` je enak `Ales`.

3. `SELECT stolpec1, stolpec2,... FROM imeTabele`
   Primer: `SELECT ime, priimek FROM Igralci`
   Opis: Prikaže samo stolpca `ime` in `priimek` tabele `Igralci`.

4. `CREATE TABLE imeTabele stolpec1, stolpec2, ...`
   Primer: `CREATE TABLE Slikar Ime, Priimek`
   Opis: Ustvari tabelo z imenom `Slikar`, ter stolpcema `Ime, Priimek`.

5. `CREATE SHEMA imeSheme`
   Primer: `CREATE SHEMA Hrana`
   Opis: Ustvari shemo z imenom `Hrana`.

6. `DROP TABLE imeTabele`
   Primer: `DROP TABLE Slikar`
   Opis: Izbriše tabelo z imenom `Slikar`.

7. `DROP SHEMA imeSheme`
   Primer: `DROP SHEMA Hrana`
   Opis: Izbriše shemo z imenom `Hrana`.

8. `INSERT INTO imeTabele VALUES vrednost1, vrednost2, ...`
   Primer: `INSERT INTO Igralci VALUES Rok, Hlod`
   Opis: V tabelo `Igralci` shrani v prve dva stolpca vrednosti `Rok`, `Hlod`.

Sedaj, ko poznamo vse poizvedbe SQL, si lahko bolj podrobno pogledamo ozadje vsake izmed poizvedb.

1. `SELECT * FROM imeTabele`

Prva poizvedba nam prikaže vse podatke za podano tabelo. Vse kar moramo storiti je, da preverimo če omenjena tabela obstaja v naši trenutno izbrani shemi. To enostavno storimo z že opisano metodo `getTableNames(imeSheme)`, ki vrne v vektorju zapisana imena vseh tabel za podano shemo. S `for` zanko se sprehodimo čez vektor vseh tabel in v primeru, da najdemo ime tabele za katero poizvedujemo, vemo da se naša tabela nahaja v izbrani shemi. Ko enkrat vemo, da omenjena tabela obstaja v naši trenutno izbrani shemi, to tabelo nastavimo v spustni seznam tabel, ter jo prikažemo v naši tabeli. Pravzaprav je rezultat enak rezultatu, kot bi iz spustnega seznama tabel izbrali tabelo. Vse spodaj omenjene metode so že razložene v podpoglavju 3.2.1.

```
1 stStolpcev = xml.stStolpcev(iSheme, iTabele);
2 iStolpcev=xml.imenaStolpcev(iSheme,
3 iTabele,stStolpcev);
4 podatki = new Vector<Vector>(stStolpcev);
```

```
5 podatki=xml.imenaElementov(iSheme,
6 iTabele,stStolpcev);
7 nastaviTabelo(podatki,iStolpcev);
8 comboboxT.setSelectedItem(iTabele);
9 rezultat.setText("Poizvedba uspesna!");
```

Del kode za poizvedbo `SELECT * FROM imeTabele`

2. `SELECT * FROM imeTabele WHERE stolpec=vrednost`

Pri drugi SELECT poizvedbi nam prikaže vse rezultate (in vse stolpce) pri katerih ime stolpca ustreza podani vrednosti. Pri tej poizvedbi je ključen spodnji klic metode `imenaElementov` iz razreda `XML`.

```
podatki = xml.imenaElementov(iSheme,iTabele,
iStolpca,vrednost,stStolpcev);
```

Zgornja metoda je pravzaprav metoda `imenaElementov(iSheme,iTabele, stStolpcev)` z dvema dodatnima parametroma `iStolpca` in `vrednost`. Ta dva parametra nam prideta prav, saj lahko na ustreznem mestu znotraj DOM drevesne strukture preverimo, če ima tabela omenjeni stolpec, ter če je njegova vrednost enaka spremenljivki `vrednost`. To storimo z naslednjima dvema `if` stavkoma:

```
if (iStolpca.equalsIgnoreCase(
childchild.getNodeName().toUpperCase().toString()))

if (childchild.getTextContent().equalsIgnoreCase(
vrednost))
```

Tukaj velja omeniti, da z omenjeno poizvedbo prikažemo novo okno s tabelo rezultatov, katere ni moč urejati.

3. `SELECT stolpec1, stolpec2,... FROM imeTabele`

Tretji in zadnji `SELECT` stavek je na las podoben prejšnjemu stavku, s to razliko, da moramo tokrat samo izpisati podane stolpce iz podane tabele. Ni nam potrebno pregledovati, če stolpec ustreza podani vrednosti. Najprej z metodo `imenaStolpcev` preverimo, če so vsi podani stolpci v naši tabeli. Če je ta pogoj izpolnjen gremo lahko na naslednji pomemben korak. Ta korak vključuje metodo `imenaElementov` iz razreda `XML`, kateri moramo podati naslednje argumente: `iSheme`, `iTabele`, `iStolpcev`, `stStolpcev`. Klic metode izgleda tako:

```
podatki=xml.imenaElementov(iSheme,iTabele,
iStolpcev,stStolpcev);
```

Omenjena metoda je, kot metoda `imenaElementov(iSheme,iTabele,stStolpcev)` s to razliko, da ji podamo še vektor z imeni stolpcev, ki jih želimo prikazati. To storimo zato, da lahko znotraj DOM drevesne strukture na ustreznem koraku z `if` stavkom preverimo, če element ustreza podanemu imenu stolpca.

```
for (int k = 0; k < stStolpcev; k++){
 if (imenaStolpcev.get(k).equals(childchild.
  getNodeName().toUpperCase().toString()))
   ustreza = 1;
}
```

Del kode metode `imenaElementov(iSheme,iTabele,iStolpcev,stStolpcev)`

4. `CREATE TABLE imeTabele stolpec1, stolpec2, ...`

Ko ustvarjamo novo tabelo moramo prvo preverit, če podana tabela morda že obstaja. To enostavno naredimo s `for` zanko v kateri se sprehodimo čez že obstoječe tabele (metoda `getTableNames(imeSheme)` v razredu `XML`). Če tabela ne obstaja jo ustvarimo s klicom metode `ustvariNovoTabelo` iz razreda `XML`. To metodo smo že opisali v podpoglavju **Ustvarjanje shem in tabel** v poglavju 3.2.2.

5. `CREATE SHEMA imeSheme`

Omenjena poizvedba ima popolnoma enako uporabnost in kodo, kot gumb `Ustvari shemo`, samo da imamo v tem primeru že podano ime same sheme, pri gumbu pa jo moramo še vpisati (slika 3.5). Koda in algoritmi so opisani v poglavju 3.2.2, podpoglavje **Ustvarjanje shem in tabel**.

6. `DROP TABLE imeTabele` in 7. `DROP SHEMA imeSheme`

Obe napisani poizvedbi imata enako uporabnost in sledita enakim algoritmom, kot opisana gumba v poglavju 3.2.2, podpoglavje **Brisanje shem in tabel**.

8. `INSERT INTO imeTabele VALUES vrednost1, vrednost2, ...`

Zadnja poizvedba SQL je popolna za zapis večjega števila enakih ali podobnih zapisov v tabelo. Da bomo razumeli kodo, moramo najprej razumeti delovanje poizvedbe in kaj naredi. Napisana poizvedba v podano tabelo vstavi novo vrstico, katero po vrsti zapiše podane vrednosti. Če je podanih vrednosti manj, kot pa je dejanskih stolpcev v tabeli, v ostale vrednosti ne zapiše nič oz. prazno besedilo. Za še boljše razumevanje si poglejmo izvedbo stavka `INSERT INTO trenerji VALUES dober, super, najbolsi` in njegov rezultat na tabeli na sliki 3.7.
Za besedo `VALUES` smo zapisali samo tri vrednosti, tabela `TRENERJI` pa vsebuje štiri stolpce. Zaradi tega je zadnji stolpec v novi vrstici prazen. Omenjeno prazno polje lahko brez težav znotraj tabele popravimo.

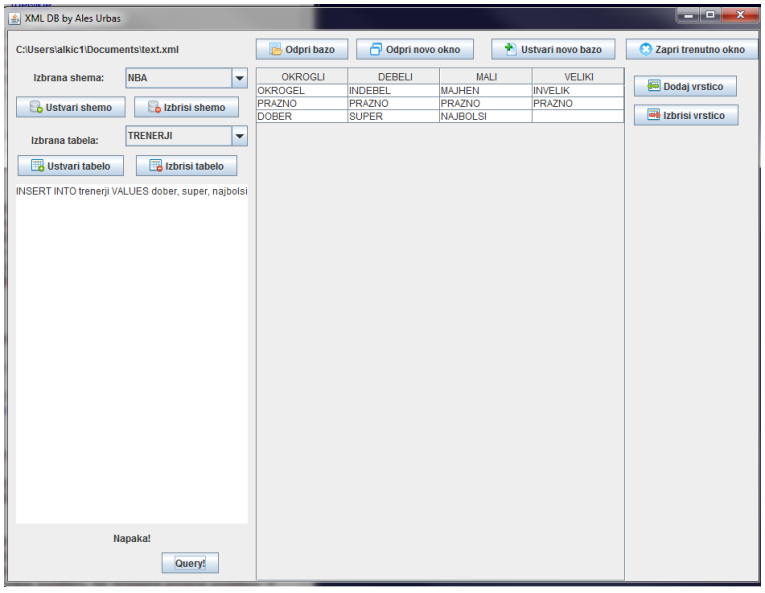
Sedaj ko natančno razumemo delovanje osmega, zadnjega stavka SQL, si bomo pogledali opis delovanje programa in programsko kodo. V vektor `zacasni` shranimo vse vrednosti, ki se pojavijo za besedo `VALUES` (zato `for` stavek začnemo z `i=4`). Koda izgleda tako:

```
st = poizvedba.length;
zacasni = new Vector(1,1);
```

```
for(int i = 4 ; i < st ; i++){
 zacasni.add(poizvedba[i]);
}
```

Nato s pomočjo metode `getTableNames(imeSheme)` pridobimo vsa imena tabel, čez katere se s `for` zanko sprehodimo, da ugotovimo ali tabela, v katero želimo vstaviti zapise sploh obstaja. Spomnimo se kode:

```
tabeleVShemi = xml.getTableNames(imeSheme);
for (int i = 0; i < tabeleVShemi.size(); i++) {
 if (imeTabele.equals(
                 tabeleVShemi.get(i).toString()))
  obstaja = 1;
}
```



Slika 3.7: Primer INSERT stavka in njegov rezultat.

Če tabela obstaja v trenutno izbrani shemi, nastavimo spustni seznam tabel na tabelo, v katero želimo vstaviti novo vrstico. Nato se v `for` zanki sprehodimo in v vektor z imenom "`v`"dodajamo zapise. Bodisi iz `zacasnega` vektorja opisanega zgoraj, bodisi prazno besedilo. Na koncu dodamo novo vrstico "`v`"vektorja v naš `DefaultTableModel` (več o `DefaultTableModel` bomo izvedeli v naslednjem podpoglavju). Programska koda izgleda tako:

```
1  if (obstaja == 1) {
2   comboboxT.setSelectedItem(imeTabele);
3   v = new Vector(tm.getColumnCount());
4   for (int i = 0; i < v.capacity();i++) {
5    if (i < zacasni.size()) {
6    v.add(zacasni.get(i).toString());
7   }
8   else {
9    v.add("");
10  }
11  }
12  tm.addRow(v);
13 }
```

Del kode za `INSERT` stavek SQL

Dodajanje in brisanje vrstic v tabeli

Vrstico v tabeli lahko dodajamo na dva načina:

1. S pritiskom tipke DOL na tipkovnici.
2. S pritiskom na gumb `Dodaj vrstico`.

Najprej si poglejmo kodo za prvi način:

```
1  vstaviVrstico = new AbstractAction() {
2   public void actionPerformed(ActionEvent e) {
3    if(table.getSelectedRow() == tm.getRowCount()-1){
4     v = new Vector(tm.getColumnCount());
```

```
5     for (int i = 0; i < v.capacity();i++) {
6      v.add("Prazno");
7     }
8     tm.addRow(v);
9     int st = tm.getRowCount();
10    table.changeSelection(st-1, 1, false, false);
11   }
12   else {
13    table.changeSelection(table.getSelectedRow()+1,
14    table.getSelectedColumn()+1, false, false);
15   }
16  }
17 };
18 table.getInputMap().put(
19 KeyStroke.getKeyStroke("DOWN"),"vstaviVrstico");
20 table.getActionMap().put("vstaviVrstico",
21 vstaviVrstico);
```

Dodajanje vrstice, ko pritisnemo tipko DOL

Predno razložimo delovanje kode, moramo razložiti spremenljivko `tm`. Spremenljivka `tm` je objekt razreda `DefaultTableModel`. `DefaultTableModel` je zadolžen za shranjevanje podatkov znotraj tabele (JTable). Tabela skrbi samo za prikaz podatkov, ter omogoča urejanje podatkov. V ozadju pa za dejanske podatke (v našem primeru) skrbi `DefaultTableModel`. To storimo z dvema vrsticama kode:

```
tm = new DefaultTableModel(podatki, imenaStolpcev);
table = new JTable(tm);
```

Sedaj si poglejmo kodo za dodajanje vrstice s pomočjo tipke DOL. Ko se sprehajamo po tabeli in na tipkovnici pritisnemo tipko DOL (koda v 18-21. vrstici) se sproži koda, ki najprej preveri, če se nahajamo v zadnji vrstici trenutne tabele (`if`

stavek v 3. vrstici). Če se tam nahajamo, dodamo vektor (v velikosti števila stolpcev), ki vsebuje besedilo "Prazno", v naš objekt `tm` razreda `DefaultTableModel`, ter se pomaknemo na novo ustvarjeno vrstico znotraj tabele. Če pa se ne nahajamo v zadnji vrstici tabele (`else` stavek v 12. vrstici), pa se enostavno premaknemo samo na naslednjo vrstico v tabeli (13 in 14. vrstica).

Poglejmo si kodo, ki se izvede če pritisnemo na gumb `Dodaj vrstico`:

```
1  public void dodajVrstico() {
2   v = new Vector(tm.getColumnCount());
3   for (int i = 0; i < v.capacity();i++){
4    v.add("Prazno");
5   }
6   tm.addRow(v);
7  }
```

Metoda `dodajVrstico()`

Koda je popolnoma enaka kodi, ko se nahajamo v zadnji vrstici tabele in pritisnemo tipko DOL.

Za brisanje vrstice imamo samo en način. Izberemo vrstico, ki jo želimo izbrisati, ter pritisnemo tipko `Izbrisi vrstico`. Tukaj velja omeniti, da v primeru ko ne označimo nobene vrstice metoda `table.getSelectedRow()` vrača vrednost -1. Koda, ki se izvrši ob pritisku gumba `Izbrisi vrstico`:

```
1  public void brisiVrstico() {
2   if (table.getSelectedRow() != -1)
3    tm.removeRow(table.getSelectedRow());
4   else
5    JOptionPane.showMessageDialog(okno,
6     "Izberi vrstico, ki jo zelis izbrisati!",
7     "Napaka!",
8     JOptionPane.ERROR_MESSAGE);
9  }
```

Metoda `brisiVrstico()`

Odpiranje in zapiranje glavnega okna

Ob pritisku na gumb `Odpri novo okno` se naredi nov objekt razreda `MainFrame.java` in odpre se novo okno programa. Koda ki se izvrši:

```
MainFrame novoOkno = new MainFrame();
novoOkno.setVisible(true);
```

Ko imamo enkrat odprtih več oken, moramo omogočit možnost zapiranja samo enega okna naenkrat in ne vseh odprtih oken, tako kot to naredi klik na gumb `X` v desnem zgornjem kotu programa. To naredimo s klikom na gumb `Zapri trenutno okno`, ki izvede kodo:

```
okno.dispose();
```

Poglavje 4

Sklepne ugotovitve

V diplomskem delu smo s teoretičnega vidika pogledali tehnologije XML, DTD, shema XML, XSL, XLink, DOM in XPointer. V nadaljevanju smo razložili delovanje razvitega programa in opisali njegove algoritme in podatkovne strukture.
Uspešno smo uspeli razviti program, ki odpira, ureja in shranjuje podatke celotne podatkovne baze v eni sami datoteki formata XML. Z njegovo pomočjo enostavno pregledujemo vsebino podatkovne baze (sheme in tabele), urejamo zapise znotraj tabel, dodajamo nove itd. Spremembe, ki jih s programom naredimo na podatkovni bazi pa so takoj zapisane v sam dokument XML. Program je bil napisan v programskem jeziku Java in je zato neodvisen od platforme (lahko ga poženemo na poljubnem računalniku).
Program še zdaleč ni idealen in ima še veliko prostora za izboljšave. Pri nadaljnem razvoju programa bi v prvem koraku bilo pametno dodati možnost dodajanja, brisanja in urejanja stolpcev tabel (sedaj imamo samo možnost urejanja vrstic). V naslednjem koraku bi bilo pametno dodati možnost prenosa tabel med različnimi bazami/shemami. Pri ustvarjanju tabele bi bilo dobro, če bi uvedli podatkovne tipe (int, string, date itd.), tako kot je to narejeno pri večji podatkovnih bazah. Prav tako bi bila zelo hvaležna opcija znotraj aplikacije, razveljavi spremenjene podatke. Ta bi stanje podatkovne baze povrnila na začetek, kot da bi bila ta ravnokar ponovno odprta. Možnost, katera bi prav tako bila vredna razmisleka je razvoj JDBC gonilnika. Z njegovo pomočjo bi imeli standarden dostop do datotek tipa XML (naš gonilnik bi lahko uporabili tudi drugi razvijalci), lahko bi izvajali poizvedbe SQL itd.

Slike

Tabele

Literatura

[1] Harvey M. Deitel, Paul J. Deitel, Tem R. Nieto, Ted Lin, and Praveen Sadhu. *XML how to program.* Prentice-Hal, Inc., 2001, pogl. 7.

[2] doc. dr. Damjan Vavpotič. Informacijski sistemi predavanja, 2. del (prosojnice), dostopno na:. http://ucilnica.fri.uni-lj.si/mod/folder/view.php?id=10694, Avgust 2012, str. 1.

[3] World Wide Web Consortium (W3C) DOM. Dostopno na:. http://www.w3schools.com/dom/default.asp, Avgust 2012.

[4] World Wide Web Consortium (W3C) DTD. Dostopno na:. http://www.w3schools.com/dtd/default.asp, Maj 2012.

[5] Oracle (2012). NetBeans IDE. Dostopno na:. http://netbeans.org/, Avgust 2012.

[6] World Wide Web Consortium (W3C) XLink in XPointer. Dostopno na:. http://www.w3schools.com/xlink/default.asp/, Junij 2012.

[7] Oracle (2012). Java. Dostopno na:. http://java.com/en/, Avgust 2012.

[8] Brett McLaughlin. *Java & XML, second edition.* O'reilly & Associates, Inc., 2001, pogl. 1.

[9] World Wide Web Consortium (W3C). Dostopno na:. http://www.w3.org/, Maj 2012.

[10] World Wide Web Consortium (W3C) XML. Dostopno na:. http://www.w3schools.com/xml/, Maj 2012.

[11] World Wide Web Consortium (W3C) Schema XML. Dostopno na:. http://www.w3schools.com/schema/default.asp, Junij 2012.

[12] World Wide Web Consortium (W3C) XSL. Dostopno na:. http://www.w3schools.com/xsl/, Junij 2012.