Proces razvoja generatora klasa sa implementiranim metodama za izvršavanje CRUD operacija

Upotreba generisanih klasa za razvoj JAVA desktop aplikacija

Drago Vidović, Dragoljub Pilipović
studenti drugog ciklusa studija
Slobomir P Unvierzitet
Bijeljina, Bosna i Hercegovina – Republika Srpska
drago.vidovic@spu.ba, dragoljub.pilipovic@gmail.com

Sadržaj – **U radu će biti istaknuta važnost razvoja poslovnih aplikacija, u skladu sa njihovom trenutnom potrebom na tržištu softvera i uvid u potrebu svih aplikacija za realizovanjem osnovnih operacija (CRUD – *Create Read Update Delete*) nad bazom podataka, koje su osnov za druge složenije operacije. Takođe, biće predstavljena mogućnost automatizacije razvoja poslovnih aplikacija, realizovanjem automatskog generatora domenskih klasa sa implementiranim metodama za izvršavanje CRUD operacija nad bazom podataka i detaljnijim opisom upotrebe generisanih klasa za brz razvoj konkretne poslovne aplikacije.**

Ključne riječi - poslovna aplikacija, desktop aplikacija, generator koda;

I. UVOD

Informacione tehnologije, trenutno predstavljaju oblast poslovanja koja se razvija jako velikom brzinom. Na tržištu razvoja softvera javlja se sve veći broj firmi koje se bave razvojem softvera, koje su spremne u svakom trenutku da odgovore na zahteve klijenata i razviju poslovne aplikacije gotovo istog kvaliteta. Presudan faktor za dobijanje posla postaje vreme, kao i cena koja zavisi od vremena koje je potrebno da se projekat realizuje i dobije tražena poslovna aplikacija. U skladu sa navedenim, postavlja se pitanje kako vreme za izradu softvera skratiti, povećati produktivnost i samim tim biti ispred konkurencije.

Na kratko ćemo se osvrnuti u prošlost da bismo videli težnju i pravce razvoja softverskog inženjerstva. Od samog početka softverskog inženjerstva četrdesetih godina prošlog veka do danas, razvoj softvera konstantno napreduje. Širi se spektar namena aplikacija, raste njihova složenost. Stalni ciljevi su poboljšanje produktivnosti softverskih inženjera i povećanje kvaliteta aplikacije za krajnjeg korisnika [1].

Takozvana softverska kriza, od 1965 – 1985, identifikovala je mnoge probleme u razvoju softvera. Projekti su daleko premašivali budžet i rokove, uzrokovali gubitak imovine (usled loše zaštite softvera, hakeri su uspevali da ukradu informacije, novac ...), i čak uzrokovali smrtne slučajeve (zbog greške u proračunavanju doze u terapiji zračenjem umrla su barem 3 pacijenta).

Za softversku krizu je decenijama traženo jednostavno i neposredno rešenje, kao što je na primer razvoj alata (strukturno programiranje, OOP, CASE alati, dokumentacija, standardi, UML...), poboljšanje discipline programera (neki su smatrali da je jedini problem u nedostatku discipline kod programera), razvoj formalne metode (pokušaj da se primenom formalne inženjerske metodologije u softversko inženjerstvo, ono učini jednako predvidivim kao druge grane industrije) i povećanje profesionalizma (rad na etici, licencama i profesionalizmu). Postalo je jasno da magičnog rešenja nema, ali su svi navedeni alati i tehnike doveli do inkrementalnog poboljšanja produktivnosti i kvaliteta.

Softversko inženjerstvo je mlada disciplina i još uvek se razvija. Trenutne tendencije u razvoju softvera su: *aspekti* – (eng. *aspects*) obezbeđuju alate za dodavanje ili uklanjanje generičkog koda, *agilno programiranje* – koje uključuje razvoj funkcionalnih jedinica softvera, iterativnost, manje dokumentacije, timski rad, saradnju sa klijentom i prilagođavanje aplikacije tokom životnog ciklusa projekta, *eksperimentalno softversko inženjerstvo* – grana koja se bavi pravljenjem eksperimenata na softveru, prikupljanjem podataka iz eksperimenata i postavljanjem zakona i teorema na osnovu tih podataka, *razvoj vođen modelom* (eng. *model-driven development*) – razvoj tekstualnih i grafičkih alata koji koriste transformacije modela i generisanje koda da bi generisali dobro organizovane fragmente koda koji služe kao osnova za kreiranje aplikacije, *okviri* – takozvane proizvodne linije softvera – sistematičan način da se proizvede familija softverskih sistema, umesto jednog individualnog proizvoda. Ovaj postupak zahteva dobro planiranu i razvijenu ponovnu upotrebu koda. U suštini, to je pokušaj industrijalizacije razvoja softverskog sistema.

Ako analiziramo trenutnu situaciju na tržištu softvera, uočićemo da većinu aplikacija koje se prave po narudžbi klijenta čine poslovne aplikacije. Upravo u skladu sa navedenim tendencijama softverskog inženjerstva,

pokušaćemo objasniti mogućnost povećanja produktivnosti i smanjenja vremena potrebnog za razvoj poslovnih aplikacija.

II. POSLOVNE APLIKACIJE

Poslovne aplikacije predstavljaju informacioni sistem određene firme, koji sadrži poslovne podatke i operacije za manipulaciju tim podacima [2]. Poslovni podaci su smešteni u nekoj od baza podataka i gotovo uvek postoji potreba za pregled, unos, brisanje i izmenu tih podataka. Ove operacije nad podacima predstavljaju osnovu za druge, složenije operacije. Odatle se već može zaključiti da su poslovne aplikacije donekle standardizovane i pojavljuje se težnja da se njihov razvoj automatizuje na određeni način.

Automatizaciju razvoja poslovnih aplikacija možemo postići razvojem automatskih generatora aplikacija ili nekog njihovog dela (domenskih klasa, tabela u bazi podataka, dokumentacije itd.), koji će u velikoj meri olakšati posao programera, povećati njegovu produktivnost i značajno smanjiti vreme potrebno za izradu poslovnih aplikacija.

III. AUTOMATSKO PROGRAMIRANJE

Automatsko programiranje predstavlja kompjutersko programiranje u kome neki mehanizam generiše program umesto da čovek, programer, piše kod [3].

Evolucijom kompjutera i nauke o njima menjala se i definicija automatskog programiranja. Četrdesetih godina prošlog veka bilo je opisano kao automatizacija ručnog procesa bušenja kartica. Zatim je predstavljalo prelazak na jezike višeg nivoa kao što su *Fortran* i *Algol.* Jedan od prvih prevodioca (kompajlera) nazvan je *Autokod* (eng. *Autocode*) [4]. Parnas je zaključio da je automatsko programiranje uvek predstavljalo eufemizam za programiranje u jeziku višeg nivoa od onog koji je bio dostupan programerima u tom trenutku.

Generativno programiranje (proizvodno) je programiranje koje koristi automatizovano kreiranje programskog koda preko generičkih frejmova, klasa, prototipa, skica, itd [5].

Generisanje programskog koda (eng. *source code*) je bazirano na ontološkom modelu (npr. šemi) koji se preko nekog programerskog alata ili integrisanog razvojnog okruženja (eng. *IDE*) transformiše u programski kod [6].

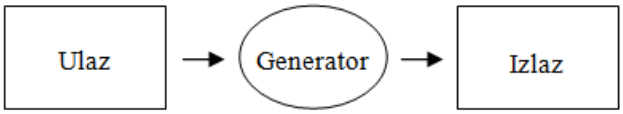
Neki *IDE*-i sadrže razne forme generisanja programskog koda:

- Čarobnjaci (eng. *wizard*) – niz dijaloga vodi korisnika kroz niz koraka kojima odabira željenu akciju. Postoje čarobnjaci za interaktivno generisanje *GUI*-a, koji u pozadini generišu izvorni kod.
- Snipeti (eng. *snippets*) – relativno mali, ponovljivi parčići koda koji se „umeću“ u veće programske module.
- Refaktorizacija – promena unutrašnje strukture programa, bez promene njegove spoljašnje funkcionalnosti u cilju poboljšanja kvaliteta softvera, njegove čitljivosti, performansi, jednostavnosti održavanja i proširivanja, pojednostavljenja strukture, kao i promena koda u svrhu uklapanja u željenu programsku paradigmu.

A. Generator aplikacija

Generator aplikacija predstavlja softversku komponentu koja služi za automatizovano pravljenje drugih softverskih komponenti [2].

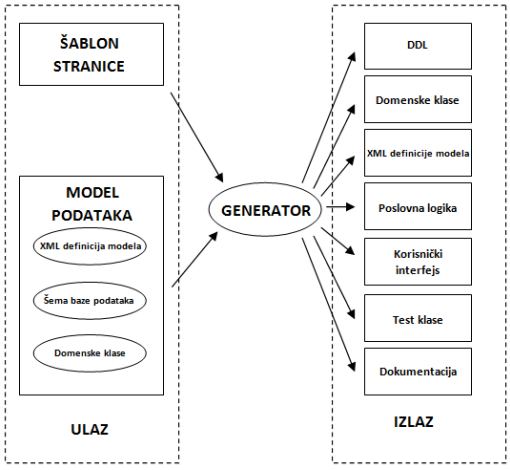
Svaki generator koristi određeni ulaz i na osnovu definisanih pravila generiše izlaz iz generatora.



Slika 1. Opšti model generatora aplikacija

Najčešće korišćeni ulazi u generator su model podataka i šablon stranice:

- **Model podataka** – predstavlja skup podataka, njihovih tipova i međusobnih veza. Može biti dat u vidu xml datoteke u kojoj su definisane tabele i polja baze podataka (xml definicija modela), kao fizički objekti baze podataka sa kojima se ostvaruje konekcija ili proizvoljni domenski objekti (npr. Java Entity klase).
- **Šablon stranice** – predstavljaju tekstualne datoteke koje se sastoje iz statičkih delova – koje čini proizvoljni tekst i dinamičkih delova – koji su predstavljeni oznakama specifičnim za dati šablon i koji se menjaju u zavisnosti od modela podataka. Za šablon stranice se mogu koristiti tehnologije kao što su HTML, JSP, ASP, Freemarker, Velocity i mnoge druge.



Slika 2. Ulazi i izlazi iz generatora aplikacija

Generator, koji se još naziva i procesor šablona, koristi dati model podataka i šablonske stranice, integriše ih na taj način što odgovarajuće podatke modela postavlja u za to označena mesta na šablon stranicama i proizvodi odgovarajuć izlaz. Neki od široko primenjenih procesora šablona su Freemarker template i Apache Velocity.

Izlaz iz generatora može biti HTML stranica, Java izvorni kod, sql skripta, XML definicija ili datoteka sa dokumentacijom.

B. Opravdanost upotrebe generatora

Pre donošenja odluke o izradi nekog generatora aplikacija, potrebno je razmotriti potrebu i opravdanost takvog rešenja kao i prednosti i nedostatke ručnog razvoja aplikacija. Do ideje o razvoju generatora aplikacija, obično se dolazi nakon izrade određenog broja aplikacija i uočavanja delova koji se ponavljaju i koji mogu biti realizovani automatski. Moramo biti svesni odnosa uloženog rada i dobiti kada se bira između izrade generatora i ručnog razvoja aplikacije. Ako gledamo sa strane cene koštanja, obično će izrada generatora biti i veća nego cena jedne aplikacije, a takođe i vreme izrade, ali sa druge strane ukoliko nakon razvijenog generatora generišemo više aplikacija, samim tim povećavamo i isplativost i opravdanost na preduzimanje ovog koraka.

Ovde ćemo navesti neke od koristi upotrebe generatora, naspram ručne izrade aplikacije:

- Upotreba generatora značajno skraćuje vreme potrebno za razvoj aplikacije i povećava produktivnost.
- Povećava kvalitet programskog koda – generiše se standardizovan kod i doprinosi se konzistentnosti imenovanja datoteka, metoda i promenjivih.
- Eliminiše monotonija u radu – čime direktno utičemo na produktivnost zaposlenih, jer upotrebom generatora izbegavamo pisanje velike količine sličnog i ponavljajućeg koda.
- Mogućnost ponovne upotrebe – kada jednom dobro napravimo generator, sa malim izmenama može biti korišćen i u novim projektima iste strukture.
- Mogućnost višestruke upotrebe – jedan generator snabdeven sa odgovarajućim modelom podataka može se koristiti za generisnje programskog koda, baze podataka, xml datoteka sa definicijama modela, dokumentacije i drugih proizvoda.
- Omogućava jedinstvenu ulaznu tačku za promene – masovne promene na brojnim mestima u aplikaciji zahtevaju samo izmenu u odgovarajućim šablonima i ponovno startovanje generatora.
- Smanjuje mogućnost greške – ručnim kodiranjem često nastaju sintaksne greške koje kompajler ne može, ili bar ne dovoljno precizno da definiše i koje zahtevaju dosta vremena da se otkriju.
- Omogućava upoređivanje različitih pristupa – pruža mogućnost da se na brz i lak način pre realizacije projekta probaju različite implementacije i tako testiraju i uporede performanse i to samo promenom napravljenog šablona.
- Razdvaja poslovnu logiku od prezentacione tehnologije – konkretna prezentaciona tehnologija predstavljena šablon stranicama je potpuno nezavisna od tehnologije kojom je predstavljena poslovna logika.

Međutim, generatori mogu predstavljati i mač sa dve oštrice. Potrebno je obratiti pažnju na slučajeve kada mogu proizvesti više problema nego koristi, kao na primer: kada obim posla nije dovoljno veliki da bi pravljenje generatora bilo isplativo u smislu vremena i resursa, kada deo aplikacije koju trebamo generisati nema dobro poznatu strukturu, kada skup osobina ili dizajn aplikacije nisu dovoljno precizno definisani unapred ili se često menjaju, za delove koji nisu šablonski, tj. koji su jedinstveni.

Nakon razmatranja svih navedenih razloga za i protiv upotrebe generatora, možemo zaključiti da generatori mogu predstavljati veliku vrednost za jedan razvojni tim, ali ih i ne prihvatati uvek olako, bez prethodne analize.

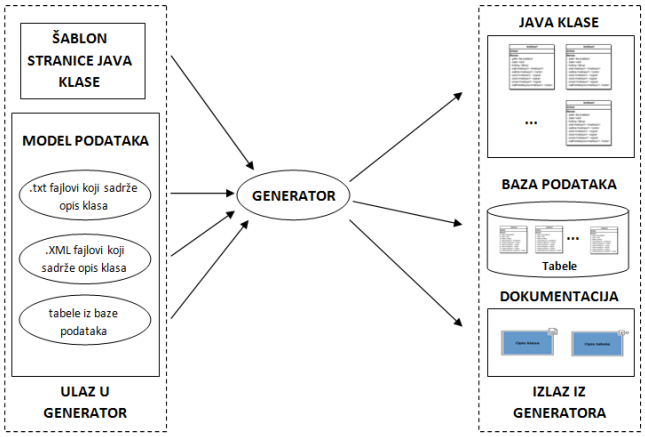
IV. RAZVOJ SOPSTVENOG GENERATORA

Bilo da radite kao samostalni programer ili kao programer u timu, svakom novom projektu pristupaćete sa znanjem i iskustvom stečenim na prethodnim. Kvalitetno osmišljena rešenja, do kojih se dolazi neprestanim usavršavanjem koda, usvajaćemo i primenjivati u daljem radu. Posebno, kao što je napomenuto u prethodnom delu rada, razvojem poslovnih aplikacija, može se uočiti da su one standardizovane, rade sa podacima koji se čuvaju u bazi podataka i obično imaju potrebu za izvršavanjem osnovnih CRUD operacija nad bazom za dobijanje traženih informacija. Ako ovo uzmemo u obzir, razvoj sopstvenog generatora za automatizaciju poslova prilikom razvoja poslovnih aplikacija, bio bi u potpunosti opravdan.

Teško bismo mogli odmah uočiti delove pogodne za automatizaciju i način na koji će oni biti automatizovani, bez prethodnog iskustva, razvijanja nekoliko poslovnih aplikacija i pronalaženja najboljeg načina za rad sa podacima u poslovnim aplikacijama. Praksa obično pokazuje, da razvijanjem svake sledeće poslovne aplikacije, programski kod se neprestano usavršava i dolazi do trenutka kada je pogodan da se prihvati kao standard za razvoj u jednom programerskom timu, te kao takav postaje pogodan za automatizaciju.

Dakle, pre razvoja sopstvenog generatora, treba izdvojiti delove pogodne za automatizaciju, odnosno one delove za koje su pronađena najbolja rešenja. Ukoliko se radi o radu u timu, onda to ne bi smela biti odluka jednog člana, bez dobrog razumevanja i upoznavanja sa predloženim rešenjem od strane ostalih članova tima. Budući da ovde razmatramo razvoj poslovnih aplikacija, deo koji je od izuzetne važnosti za svaki projekat, jesu podaci, odnosno način na koji te podatke administrirati u bazi podataka.

Naša ideja bila je da napravimo generator, koji će nam pomoći oko generisanja jednog većeg dela poslovne aplikacije uz minimalan napor programera za pisanje programskog koda. Odluka za generisanje dela aplikacije, doneta je zbog činjenice da smo pažnju usmerili na podatke i operacije nad bazom, a nikako ne smemo zaboraviti važan deo svake aplikacije, a to je korisnički interfejs, čija automatizacija nije predmet ovog rada, ali je data kao predlog za dalje usavršavanje generatora. Ovim ostavljamo prostora programerima da u pogledu dizajna unose dodatna usavršavanja, koristeći automatski generisane metode koje će puniti interfejs podacima, do trenutka kada će se smatrati pogodnim za automatizaciju.



Slika 3. Konceptualni model generatora

Na prethodnoj slici dat je prikaz komponenti koje čine ulaz i izlaz iz generatora, o čemu će više biti reč u nastavku.

A. *Ulaz*

Posebnu pažnju prilikom razvoja generatora, treba obratiti na ulazne podatke, jer dobro struktuirani ulazni podaci će rezultirati kvalitetnim i upotrebljivim rezultatima na izlazu. Vidimo da ulaz u generator čine: šablon stranice JAVA klase i model podataka.

1) Šablon stranice JAVA klase

Šablon stranice JAVA klase je tekstualni fajl koji sadrži fiksne delove, sa dinamičkim delovima (jedinstvenim) koji zavise od modela podataka. Ovim šablonom data je osnovna struktura jedne JAVA klase. Ovde navodimo samo jedan deo šablona:

```
public class <ImeKlase> {
        // atributi
        atribut 1 (tip ime)
        atribut 2 (tip ime)
        ...
        atribut n (tip ime)
        // konstruktor
        public <ImeKlase>(atribut 1, atribut 2, ..., atribut n){
                this.atribut1 = atribut1;
                this.atribut2 = atribut2;
                ...
                this.atributn = atributn;
        }
        // geteri
        public <tip> get<atribut>(){
                return <atribut>;
        }
        ...
        // seteri
        public void set<atribut>(){
                this.<atribut> = <atribut>;
        }
        ...
        // toString
        public String toString(){
                return <atribut>;
        }

        ...
}
```

Osnovna ideja jeste da se pročita sadržaj šablona i dinamički delovi zamene odgovarajućim, u zavisnosti od modela podataka. Najveći deo dinamičkih delova čine atributi klase, koji su realizovani kao posebna java klasa „Atribut“, sa metodama: *vratiIme(), vratiTip(), vratiGet(), vratiSet(), vratiToString(), vratiKonstruktor() ...,* čime je problem popunjavanja ovih delova rešen.

2) Model podataka

Što se tiče modela podataka, ostavljena je mogućnost da se podaci opišu na više načina, što čini aplikaciju fleksibilnijom za upotrebu. Podaci se mogu opisati upotrebom tekstualnih fajlova sa standardnim formatom ili XML fajlova odgovarajuće strukture.

Format tekstualne datoteke:

```
// opis klase
Naziv klase: | <nazivKlase>
Atribut: | <nazivAtributa> | <tipAtributa> | <primarniKljuc> | <status> | <ogranicenja>
Metode: | geteri | seteri | vrati<NazivKlase> | vratiSve<NazivKlase> | ...
```

Format XML datoteke:

```
<klasa>
        <imeKlase>
                <atribut>
                        <tip></tip>
                        <naziv></naziv>
                        <primarniKljuc></primarniKljuc>
                        <status></status>
                        <ogranicenja></ogranicenja>
                </atribut>
                <metoda>
                        <nazivMetoda></nazivMetoda>
                </metoda>
        </imeKlase>
</klasa>
```

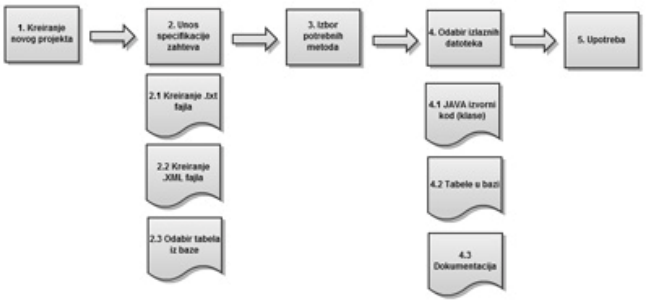
Takođe, kao ulaz u generator, moguće je koristiti i konkretnu tabelu iz baze podataka, čime se ostavlja mogućnost programerima da automatski generišu željene klase u slučaju kada je baza podataka već kreirana.

B. *Generator*

Rad generatora se može podeliti na nekoliko faza, čiji krajnji rezultat je automatski generisan kod, prilagođen za razvoj poslovnih aplikacija.

Sledeća slika prikazuje dijagram toka, generisanja i upotrebe generisanih komponenti. Još jednom ćemo

napomenuti da proces razvoja nove poslovne aplikacije se ne završava automatskim generisanjem željenih komponenti, što se može videti na slici, gde je kao faza navedena i „Upotreba“, o čemu će biti više reči u nastavku.



Slika 4. Blok dijagram generisanja i upotrebe komponenti

Upotreba generatora ne bi trebala biti ograničena samo na programera, već i na ostale korisnike (članove tima bez velikog programerskog znanja), koji imaju znanje da modeliraju sistem, saopšte to generatoru i dobiju potrebne komponente koje će biti osnov programeru za dalji razvoj poslovne aplikacije. U skladu sa tim, sam interfejs za prihvat podataka od korisnika postavlja zahteve u pogledu jednostavnosti i praktičnosti za upotrebu, gde će kroz niz koraka od korisnika tražiti unos podataka.

Analizirajući projekte slične namene, upravo jedan od čestih problema navodi se poteškoća prilikom privikavanja na okruženje generatora, odnosno učenje primene istog, kao posledica složenog interfejsa, što opravdava našu težnju za jednostavnim interfejsom.

Na ovaj način proces razvoja poslovnih aplikacija približavamo i korisnicima sa manjim iskustvom u programiranju i dajemo im mogućnost da se uključe u proces razvoja poslovne aplikacije, za šta im je dovoljna sposobnost modeliranja nekog konkretnog sistema i osnovno poznavanje tipova podataka.

C. Izlaz

Ono što očekujemo na izlazu jeste automatski generisan JAVA izvorni kod (domenske klase), kreirana baza sa tabelama čija struktura odgovara kreiranim klasama i dokumentacija koja će olakšati dalju upotrebu i održavanje sistema. Ovde ćemo obratiti pažnju na metode koje su implementirane unutar svake klase, a koje omogućavaju izvršavanje osnovnih CRUD operacija nad bazom.

- *<ImeKlase> vrati<ImeKlase>(<primarniKljuc>)* – metoda koja u zavisnosti od prosleđenog primarnog ključa tabele, vraća objekat popunjen podacima iz baze.
- *Vector vratiSve<ImeKlase>()* – metoda koja vraća sve objekte određene klase smeštene u vektoru, koji će poslužiti kao izvor za neku komponentu grafičkog interfejsa (tabelu, listu, kombo boks itd.)
- *int unesi<ImeKlase>(<ImeKlase> o)* – metod koji prihvata objekat određene klase koji je potrebno sačuvati u bazu. U zavisnosti od uspešnosti izvršene operacije, vraća odgovarajući signal koji je moguće koristiti kao kontrolni za uvid u status operacije.
- *int obriši<ImeKlase>(<primarniKljuc>)* – metod koji na osnovu prosleđenog primarnog ključa, briše slog u bazi, ukoliko postoji. U zavisnosti od uspešnosti izvršene operacije, vraća odgovarajući kontrolni signal.
- *int izmeni<ImeKlase>(<ImeKlase> o)* – metod koji prihvata objekat određene klase i vrši izmenu polja sloga u bazi. Takođe, u zavisnosti od uspešnosti operacije, vraća kontrolni signal.
- *Vector vratiPoKriterijumu<ImeKlase> (String kriterijum)* – metod koji vraća objekte smeštene u vektor koji zadovoljavaju određeni kriterijum prosleđen metodu.

Svaka od metoda je implementirana kao statička i vraća određeni tip podataka kao njen rezultat. Sledi primer poziva metode koja vraća studenta sa brojem indeksa „34/11“. Student s = Student.vratiStudent(“34/11”);

V. UPOTREBA GENERISANIH KLASA

Dobijanjem izlaza iz generatora, posao oko razvoja poslovne aplikacije nije završen. Upotreba generatora klasa predstavlja veliku vrednost za jedan razvojni tim, smanjuje vreme izrade aplikacije, time što generiše veći deo aplikacije (domenske klase čija struktura odgovara strukturi tabela u bazi podataka), ali preostaje posao oko razvoja korisničkog interfejsa koji će prikazivati podatke dobijene pozivanjem generisanih metoda.

Nastavni kadar

JMBG	Prezime	Ime	Zvanje	Titula	Maticni fakultet	Status
010179718854	Lukic	Zeljko	Vanredni prof...	prof. dr	FIT - Fakultet ...	Stalno zaposleni
1	Pera	Peric	Docent	dr	FEM - Fakultet...	Stalno zaposleni
159	Simo	Simojovic	Redovni profe...	dr	FIT - Fakultet ...	Stalno zaposleni
2	Jovo	Jovic	Asistent	dipl. ing	PF - Pravni Fa...	Honorarno za...
2309988183908	Vidovic	Jelena	Redovni profe...	prof. dr	FIT - Fakultet ...	Stalno zaposleni
2710988183908	Vidovic	Drago	Asistent	dipl. ing. infor...	FIT - Fakultet ...	Stalno zaposleni
345	Pera	Simic	Redovni profe...	prof. dr	PA - Poreska ...	Stalno zaposleni

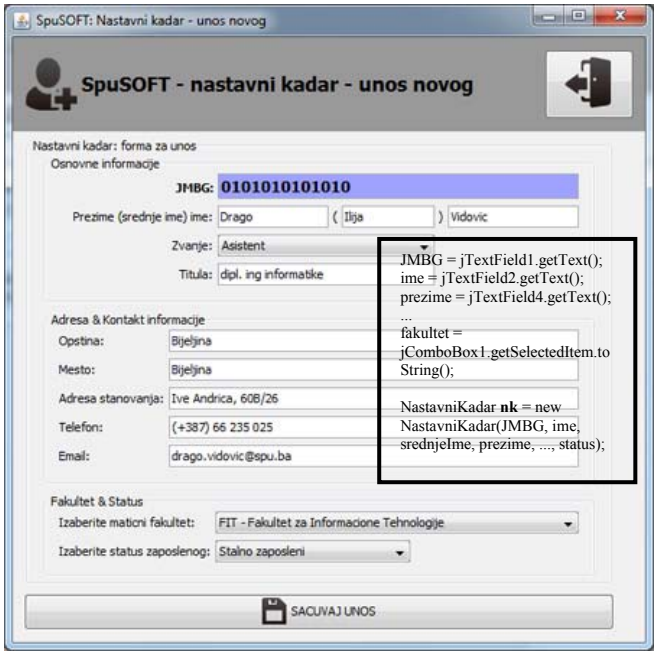
```
// popunjavanje tabele slogovima iz baze
Vector vNastavniKadar = NastavniKadar.vratiNastavniKadar();
NastavniKadar nk;
for(int i=0; i < vNastavniKadar.size(); i++){
    nk = (NastavniKadar) vNastavniKadar.elementAt(i);
    // dodavanje novog reda u tabelu
}
```

Slika 5. Popunjavanje tabele slogovima iz tabele u bazi podataka

Sa prethodne slike, možemo videti jedan od načina da popunimo tabelu sa konkretnim podacima iz baze podataka. Na ovaj način, dobijamo podatke o slogovima iz baze smešene u vektor kao objekte, čime je potrebno proći kroz sve elemente vektora i dodati ih kao nove redove u tabelu.

Prilikom unosa novog sloga u bazu, dovoljno je sa korisničke forme pokupiti (uz proveru ispravnosti unetih

podataka, ako se to zahteva) sve podatke, kreirati objekat klase koja odgovara tabeli u bazi podataka i kao parametar ga proslediti metodi unesi<ImeKlase>(<ImeKlase>);



int signal = **NastavniKadar.unesiNastavniKadar(nk);**

Slika 6. Unos novog sloga u bazu podataka

Proverom vrednosti promenjive „signal“ imamo uvid u uspešnost izvršenja pozvane operacije (0 – *slog sa unetim primarnim ključem već postoji,* 1 – *slog je uspešno unešen,* 10 – *greška prilikom izvršavanja upita*).

VI. ZAKLJUČAK I DALJE SMERNICE

U današnje vreme IT organizacije moraju odgovoriti na dva kontradiktorna zahteva: što brže kreiranje novih usluga uz istovremeno smanjivanje cena. Pored toga, IT okolina je postala izuzetno kompleksna, kako u aplikativnom, tako i u infrastrukturnom delu.

Najveće mogućnosti odgovora na navedene izazove nalaze se u operativnom delu IT organizacije i to automatizacijom procesa koja za sobom povlači standardizaciju, koja se potom preslikava i prema krajnjem korisniku. Naime, automatizacija omogućava veću usklađenost te kvalitetniju i standardiziranu komunikaciju između poslovnog i IT sveta, a najveću korist od svega imaju krajnji korisnici.

Automatizacija svakodnevnih zadataka i poslova u velikoj meri može povećati efikasnost IT-a na način da se manuelni rad smanji u najvećoj mogućoj meri.

Poštujući gore navedeno, opravdavamo razloge upotrebe generatora za razvoj poslovnih aplikacija i potvrđujemo ispravnost odluke za kreiranje istih, sa ciljem za stalnim unapređenjem u pogledu performansi i proširenju automatizacije na polju korisničkog interfejsa.

ZAHVALNICA

Veliku zahvalnost dugujem profesoru dr Draganu Uroševiću na nesebičnoj pomoći i prenesenom znanju tokom I i II cuklusa studija na predmetu Programski jezici.

LITERATURA

[1] Shari Lawrence, Joanne M. Atlee, “Softversko inženjerstvo – teorija i praksa”, prevod trećeg izdanja

[2] Strahinja Lažetić, “Razvoj generatora Spring aplikacija primenom Freemarker šablona i Hibernate okvira”

[3] Nikolche Mihajlovski, “Innovative and Pragmatic Java Source Code Generation”, geecon 2012

[4] http://en.wikipedia.org/wiki/Automatic_programming

[5] Doc. dr. sc. Danijel Radošević, “Generativno programiranje”

[6] http://blog.cedarsoft.com/2010/08/code-generation-done-right/

[7] http://onjava.com/pub/a/onjava/2003/09/03/generation.html

[8] Erich Gamme, Richard Helm, Ralph Johnson, John Vlissides, “Desigh Patterns: Elements of Reusable Objected-Orijented Software”

[9] Mr Predrag Oreški, “O standardima za razvoj softvera”, stručni rad

[10] Bertrand Meyer, “Object-Oriented Software Construction, 2nd Edition”

[11] Craig Larman, “Applying UML and Patterns: An Introduction to Object-Orijented Analysis and Design and the Unified Process”

ABSTRACT

The paper highlights the importance of the development of business applications, according to their current needs in the software market and insight into the needs of all applications for the realization of basic operations (CRUD - Create Read Update Delete) on the database, which are the basis for other more complex operations. You will be presented and automate business applications development, completion of an automatic generator of domain classes with implemented methods to perform CRUD operations on the database and detailed description of the use of generated classes for rapid development of specific business applications.

DEVELOPMENT PROCESS OF THE GENERATOR CLASS WITH IMPLEMENTED METHODS TO PERFORM CRUD OPERATIONS

Using the generated class for developing JAVA desktop applications

Drago Vidović, Dragoljub Pilipović