Asocijacija za afirmaciju mladih

MODEM

Projekat:
Razvoj softera zanatskih radnji

Autor: Dabić Aleksandar

1. SPECIFIKACIJA ZAHTEVA I SLUČAJEVI KORIŠĆENJA

1.1.OPIS PROBLEMA

Verbalni opis: Predmet ovog seminarskog rada je deo informacionog sistema zanatske radnje "HEMIJSKO CISCENJE" koji se odnosi na obaveze koje imaju kolijenti prema istom.

Korisnik: Radnik.

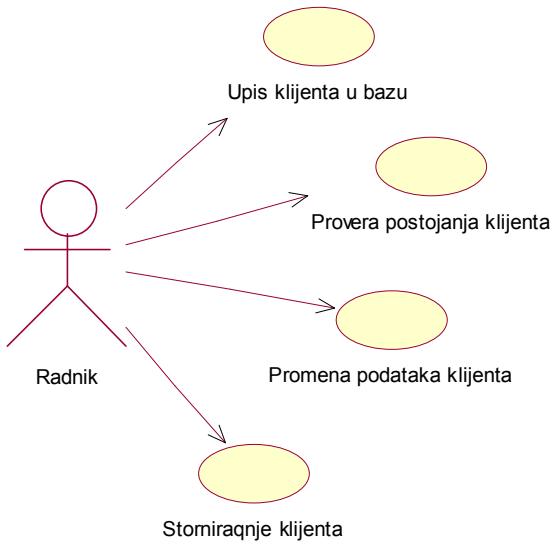
Ciljevi: Vođenje evidencije o klijentima koji koriste uskluge hemijskog čišcenja ,kao i njihovoj ceni. Prilikom obrade i konačne izrade jednog informacionog sistema treba voditi računa o tome da se prilikom projektovanja kvalitetno obrade sve ključne funkcije, što znači minimalne greške i redundansu, kao i zaštitu integriteta podataka.

1.2. SPECIFIKACIJA ZAHTEVA POMOĆU MODELA SLUČAJEVA KORIŠĆENJA

U ovom seminarskom radu imao sam sledeće slučajeve korišćenja:

1. Upis novog klijenta u «HEMIJSKO»
2. Provera postojanja klijenta
3. Promena podataka klijenata
4. Storniranje klijenta iz «HEMIJSKO-g»

što se može predstaviti sledećim dijagramom:



SLUČAJ KORIŠĆNJA 1: UPIS NOVOG KLIJENTA U «HEMIJSKO»

Naziv slučaja korišćenja:
Upis novog klijenta koji koristi usluge hemijskog čišćenja.

Aktori slučaja korišćenja:
«HEMIJSKO ČIŠĆENJE»(Radnik).

Učesnici slučaja korišćenja:
HEMIJSKO ČIŠĆENJE (radnik) i program (sistem).

Preduslovi koji moraju biti zadovoljeni da bi SK počeo da se izvršava:
Sistem je uključen i radnik je ulogovan pod svojom šifrom. Kada radnik pozove sistem da se izvrši, sistem prikazuje formu za vođenje evidencije o klijentima, na kojoj se nalazi poslednji upisan klijent.

Osnovni scenario izvršenja SK:
1. Radnik poziva sistem da ubaci novog klijenta.
2. Sistem ubacuje novog klijenta.
3. Sistem prikazuje radniku novog klijenta.
4. Radnik unosi podatke o klijentu.
5. Radnik poziva sistem da izračuna iznose po uslugama (stavkama) klijenata i ukupnu vrednost .
6. Sistem računa iznose po uslugama i ukupnu vrednost.
7. Sistem prikazuje radniku izmenjene podatke o klijentima.
8. Radnik kontroliše da li je uneo sve potrebne podatke o klijentima.
9. Radnik poziva sistem da izvrši konačnu obradu.
10. Sistem obrađuje podatke o klijentu.
11. Sistem prikazuje radniku obrađene podatke.

Alternativna scenarija izvršenja SK:
- 5.1. Ukoliko sistem ne može da izračuna iznose uskluga i ukupnu vrednost, prikazuje se poruka radniku da ne može da izračuna potrebne podatke. Prekida se izvršenje scenarija.
- 9.1. Ukoliko sistem ne može da obradi podatke prikazuje se radniku poruka da ne može da obradi podatke. Prekida se izvršenje scenarija.

Specijalni zahtevi, tehnološka pitanja, otvorena pitanja:
Nema.

SLUČAJ KORIŠĆENJA 2: PROVERA POSTOJANJA KLIJENTA

Naziv slučaja korišćenja:
Provera postojanja klijenta.

Aktori slučaja korišćenja:
Radnik.

Učesnici slučaja korišćenja:
HEMIJSKO ČIŠĆENJE(Radnik) i program (sistem).

Preduslovi koji moraju biti zadovoljeni da bi SK počeo da se izvršava:
Sistem je uključen i radnik je ulogovan pod svojom šifrom. Kada radnik pozove sistem da se izvrši, sistem prikazuje formu za vođenje evidencije o klijentima, na kojoj se nalazi poslednji upisan klijent.

Osnovni scenario izvršenja SK:
1. Radnik unosi broj klijenta koje želi da proveri.
2. Radnik poziva sistem da proveri da li postoji klijent sa zadatim brojem.
3. Sistem proverava postojanje klijenta.
4. Sistem prikazuje radniku kljijenta.

Alternativna scenarija izvršenja SK:
- 3.1. Ukoliko ne postoji klijent sa zadatim brojem sistem prikazje radniku poruku da osoba nije koristila usluge HEMIJSKOG ČIŠĆENJA. Prekida se izvršenje scenarija.

Specijalni zahtevi, tehnološka pitanja, otvorena pitanja:
Nema.

SLUČAJ KORIŠĆENJA 3: PROMENA PODATAKA KLIJENTA

Naziv slučaja korišćenja:
Promena podataka klijenta.

Aktori slučaja korišćenja:
Radnik.

Učesnici slučaja korišćenja:
HEMIJSKO ČIŠĆENJE(Radnik) i program (sistem).

Preduslovi koji moraju biti zadovoljeni da bi SK počeo da se izvršava:
Sistem je uključen i Radnik je ulogovan pod svojom šifrom. Kada Radnik pozove sistem da se izvrši, sistem prikazuje formu za vođenje evidencije o klijentima, na kojoj se nalazi poslednji upisan klijent.

Osnovni scenario izvršenja SK:
1. Radnik unosi broj klijenta čije usluge želi da promeni.
2. Radnik poziva sistem da proveri da li postoji klijent sa zadatim brojem.
3. Sistem proverava postojanje klijenta.

4. Sistem prikazuje radniku podatke o klijentu.
5. Radnik unosi dopunske/izmenjene podatke u evidenciju o klijentu.
6. Radnik poziva sistem da izračuna iznose po uslugama i ukupnu vrednost.
7. Sistem računa iznose po uslugama i ukupnu vrednost.
8. Sistem prikazuje radnoku izmenjene podatke.
9. Radnik kontroliše da li je uneo sve potrebne podatke.
10. Radnik poziva sistem da izvrši konačnu obradu podataka.
11. Sistem obrađuje podatke.
12. Sistem javlja radniku da je obradio podatke.

Alternativna scenarija izvršenja SK:

- 3.1. Ukoliko ne postoji klijent sa zadatim brojem sistem prikazje radniku poruku da klijent nije upisan u korišćenje usluga HEMIJSKOG. Prekida se izvršenje scenarija.
- 6.1. Ukoliko sistem ne može da izračuna iznose po uslugama klijenata i ukupnu vrednost, on prikazuje radniku poruku da ne može da izračuna. Prekida se izvršenje scenarija.
- 10.1. Ukoliko sistem ne može da obradi podatke prikazuje se radniku poruka da ne može da obradi podatke. Prekida se izvršenje scenarija.

Specijalni zahtevi, tehnološka pitanja, otvorena pitanja:
Nema.

SLUČAJ KORIŠĆENJA 4: STORNIRANJE KLIJENATA

Naziv slučaja korišćenja:
Storniranje klijenata.

Aktori slučaja korišćenja:
Radnik.

Učesnici slučaja korišćenja:
HEMIJSKO ČIŠĆENJE(Radnik) i program (sistem).

Preduslovi koji moraju biti zadovoljeni da bi SK počeo da se izvršava:
Sistem je uključen i Radnik je ulogovan pod svojom šifrom. Kada radnik pozove sistem da se izvrši, sistem prikazuje formu za vođenje evidencije o klijentima, na kojoj se nalazi poslednji upisan klijent.

Osnovni scenario izvršenja SK:

1. Radnik unosi broj klijenta koje želi da stornira iz evidencije.
2. Radnik poziva sistem da proveri da li postoji klijent sa zadatim brojem u bazi.
3. Sistem proverava postojanje klijenta.
4. Sistem prikazuje Radniku podatke o klijentu ukoliko postoji.
5. Radnik poziva sistem da ispiše klijenta iz daljeg obradjivanja.
6. Sistem ispisuje klijenta.

7. Sistem prikazuje radniku podatke o ispisanom klijentu.

Alternativna scenarija izvršenja SK:

- 2.1. Ukoliko ne postoji klijent sa zadatim brojem sistem prikazje klijentu poruku da klijent nije upisan u bazu. Prekida se izvršenje scenarija.
- 5.1. Ukoliko klijent ne može da se ispiše sistem prikazuje radniku poruku o nemogućnosti ispisa. Prekida se izvršenje scenarija.

Specijalni zahtevi, tehnološka pitanja, otvorena pitanja:
Nema.

2. FAZA ANALIZE

2.1. SISTEMSKI DIJAGRAMI SEKVENCI

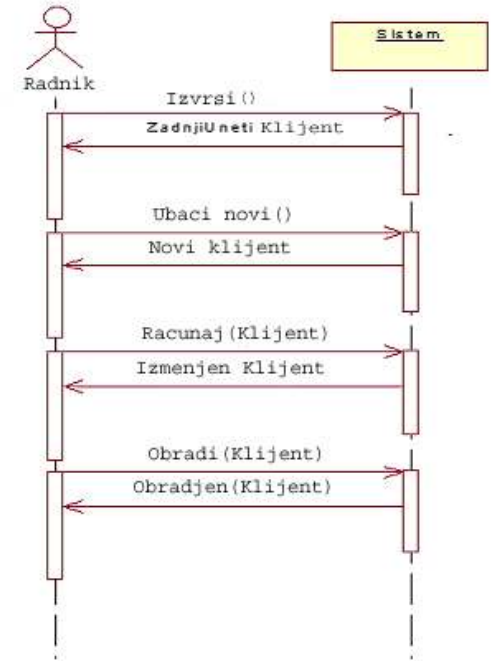
1.DIJAGRAM SEKVENCI SLUČAJA KORIŠĆENJA: **UPIS NOVOG KLIJENTA U "HEMIJSKO ČIŠĆENJE"**

Osnovni scenario:

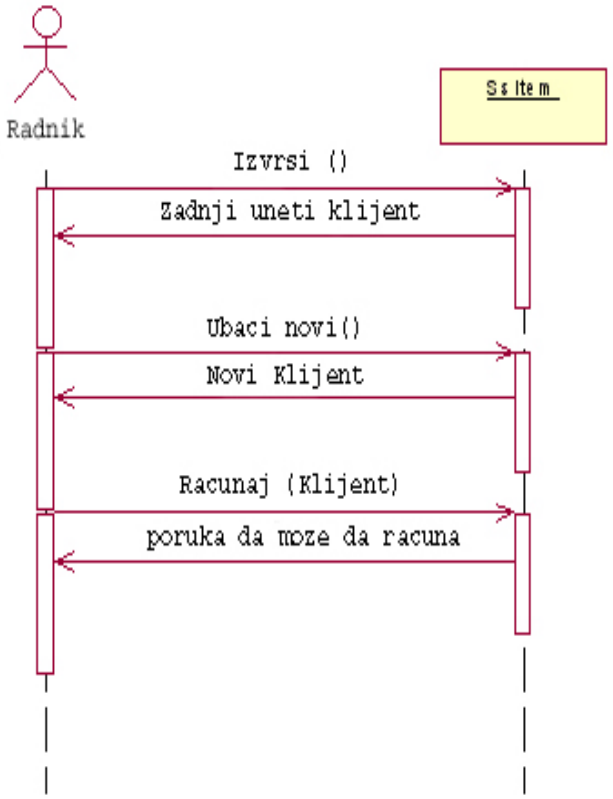
1. Korisnik poziva sistem da vrati zadnjeg upisanog klijenta.
2. Sistem prikazuje zadnjeg upisanog klijenta.
3. Radnik poziva sistem da upiše novog klijenta.
4. Sistem prikazuje radniku novog klijenta.
5. Radnik poziva sistem da izračuna iznose usluga klijenata i ukupnu vrednost.
6. Sistem prikazuje radniku izmenjene podatke.
7. Rddnik poziva sistem da izvrši konačnu obradu podataka.
8. Sistem prikazuje zaposlenom obrađene podatke.

Alternativna scenarija:

- 6.1. Ukoliko sistem ne može da izračuna iznose po uslugama i ukupnu vrednost, prikazuje se poruka radniku da ne može da izračuna potrebne podatke.



- 8.1. Ukoliko sistem ne može da obradi klijenta prikazuje se radniku poruka da ne može da obradi podatke.



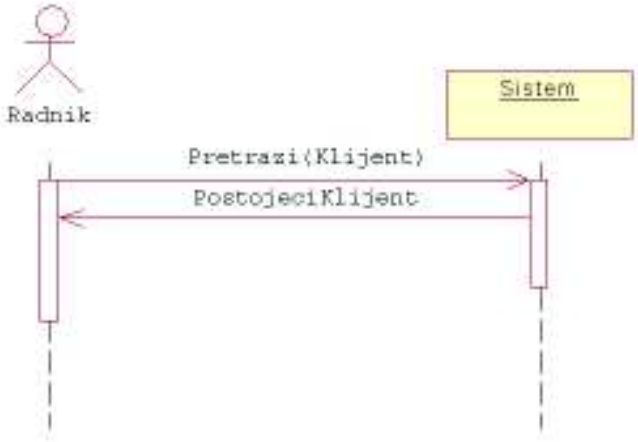
Sa navedenih sekvencnih dijagrama uocavaju se 4 sistemske operacije koje treba projektovati:

1. ZadnjiUnetiKlijent Izvršavanje();
2. NoviKlijent UpišiNovog();
3. IzmenjenKlijent Računaj(Klijent);
4. ObrađenKlijent Obradi(Klijent).

2. DIJAGRAM SEKVENCI SLUČAJA KORIŠĆENJA: **PROVERA POSTOJANJA KLIJENTA**

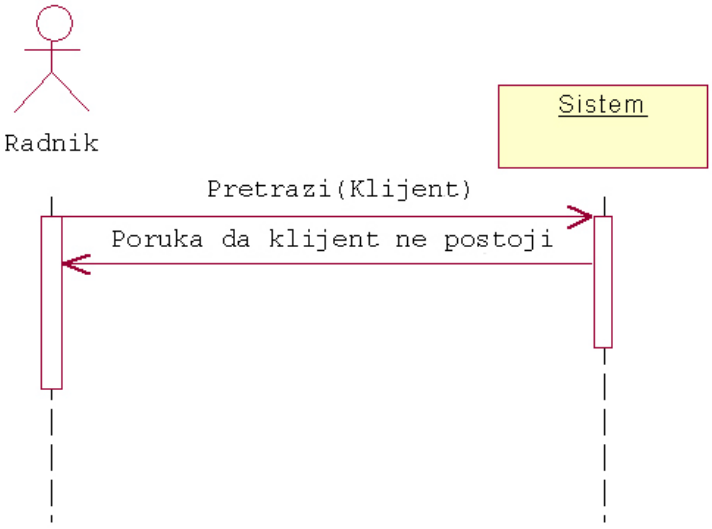
Osnovni scenario:

1. Radnik poziva sistem da proveri da li klijent sa zadatim brojem postoji.
2. Sistem prikazuje radniku klijenta ukoliko je upisan u HEMIJSKO ČIŠČENJE .



Alternativna scenarija:

- 2.1. Ukoliko ne postoji klijent sa zadatim brojem sistem prikazje radniku poruku da klijent nije upisan u HEMIJSKOČIŠĆENJE.

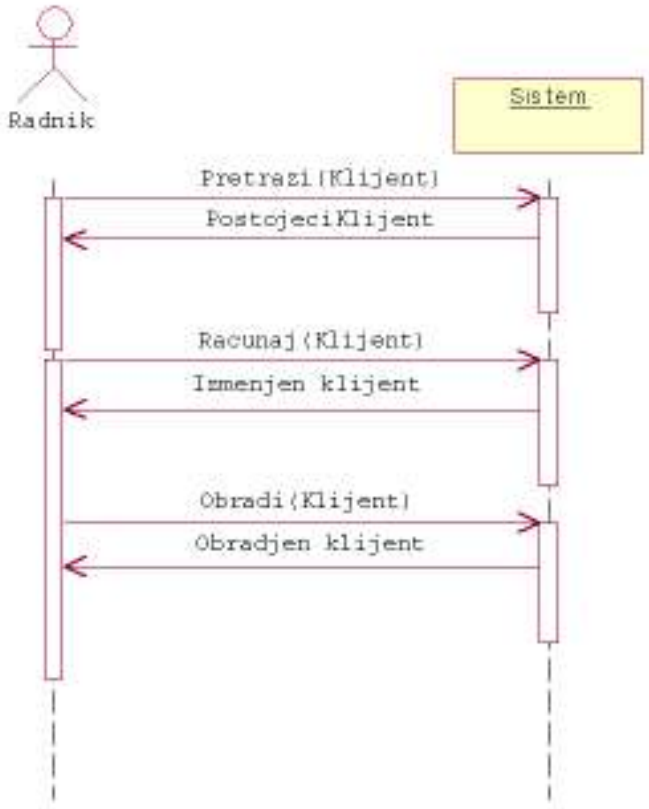


Sa navedenih sekvencnih dijagrama uočava se još jedna sistemska operacija koju treba projektovati:
1. ProveraPostojanjaKlijenta(Klijent).

3. DIJAGRAM SEKVENCI SLUČAJA KORIŠĆENJA: **PROMENA PODATAKA KLIJENTA**

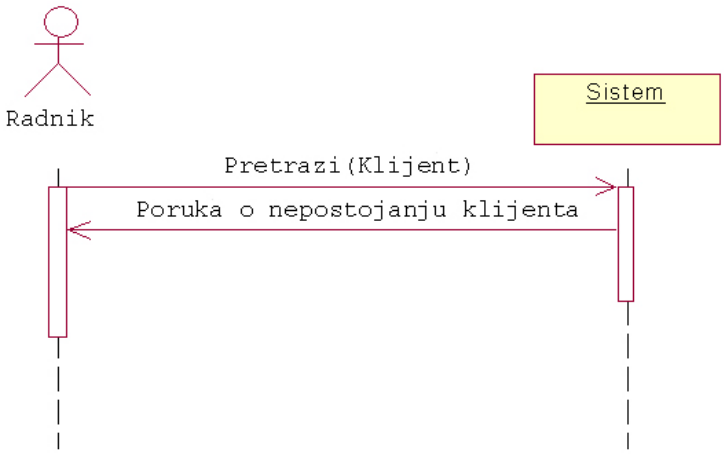
Osnovni scenario:

1. Radnik poziva sistem da proveri da li postoji klijent sa zadatim brojem.
2. Sistem prikazuje radniku klijenta .
3. Radnik poziva sistem da izračuna iznose usluga i ukupnu vrednost.
4. Sistem prikazuje radniku izmenjen račun.
5. Radnik poziva sistem da izvrši konačnu obradu klijenta.
6. Sistem prikazuje radniku obrađenog klijenta.

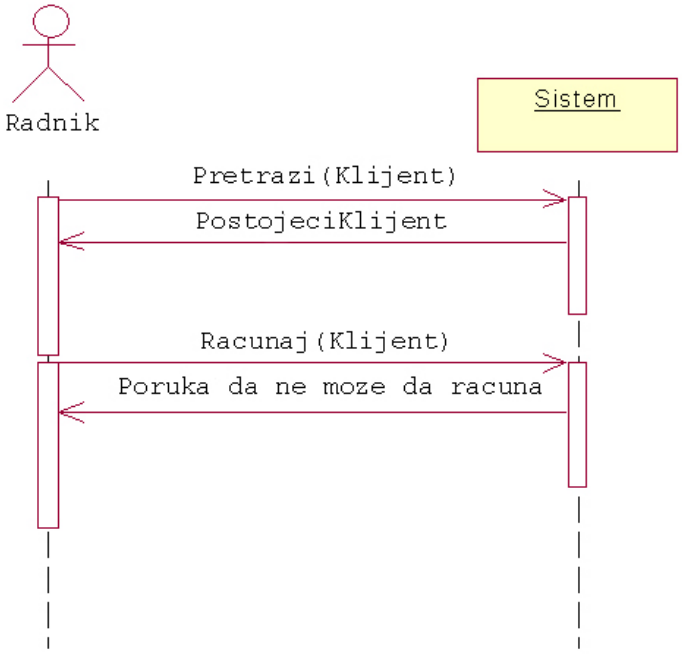


Alternativna scenarija:

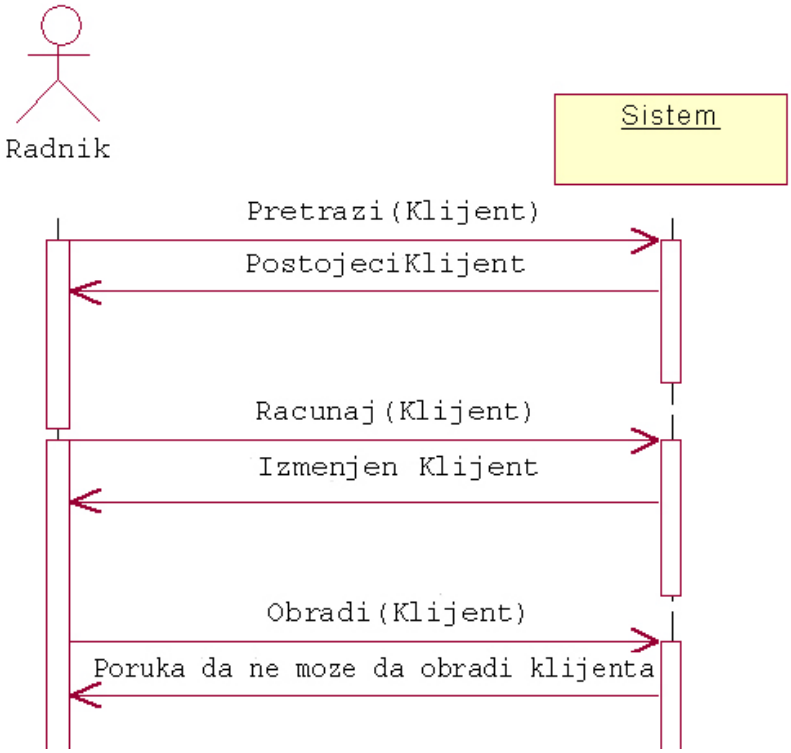
- 2.1. Ukoliko klijent ne postoji sistem prikazuje radniku poruku da klijent ne postoji. Prekida se izvršenje scenarija.



- 4.1. Ukoliko sistem ne može da računa iznose usluga i ukupnu vrednost, on prikazuje radniku poruku da ne može da računa.



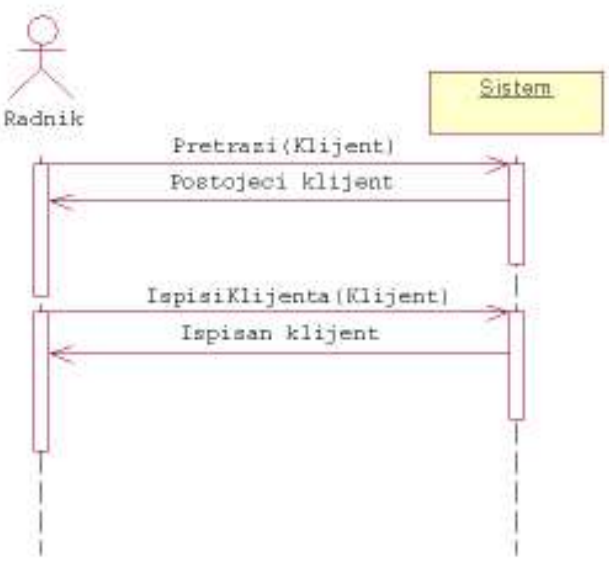
- 6.1. Ukoliko sistem ne može da obradi klijenta, on prikazuje radniku poruku da ne može da obradi klijenta. Prekida se izvršenje scenarija.



4. DIJAGRAM SEKVENCI SLUČAJA KORIŠĆENJA: **STORNIRANJE KLIJENTA IZ HEMIJSKOG ČIŠĆENJA**

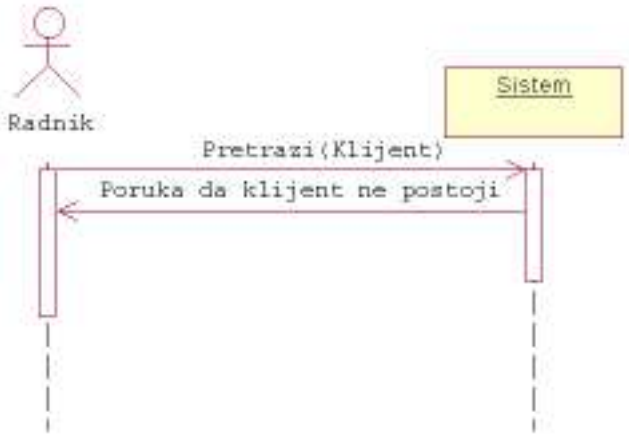
Osnovni scenario:

1. Radnik poziva sistem da proveri da li klijent sa zadatim brojem postoji.
2. Sistem prikazuje radniku klijenta ukoliko postoji.
3. Radnik poziva sistem da stornira klijenta iz hemijskog ciscenja
4. Sistem prikazuje radniku storniranog klijenta.

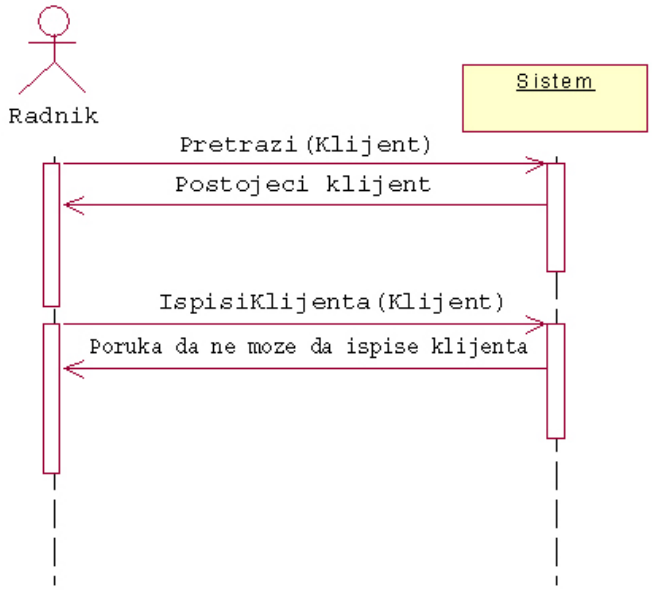


Alternativna scenarija:

- 2.1. Ukoliko klijent sa zadatim brojem ne postoji sistem prikazuje radniku poruku da klijent ne postoji. Prekida se izvršenje scenarija.



- 4.1. Ukoliko klijent ne može da se stornira sistem prikazuje radniku poruku da ne može da stornira.



Rezultat analize sistemskih dijagrama sekvenci
Kao rezultat analize dobijeno je 6 sistemskih operacija koje treba projektovati:

1. ZadnjiBrojKlijenta Izvrši();
2. NoviKlijent UbaciNovog();
3. IzmenjenKlijent Računaj(Klijent);
4. Obrađen Klijent Obradi(Klijent);
5. Postojeći Klijent Pretraži(Klijent);
6. Ispisan Klijent Ispiši(Klijent).

Svaku od SO u zavisnosti od upešnosti izvršenja vraća odgovarajući signal, pa će SO imati izgled:

1. signal Izvrši(Klijent);
2. signal UbaciNovog(Klijent):
3. signal Računaj(Klijent);
4. signal Obradi(Klijent);
5. signal Pretraži(Klijent);
6. signal Ispiši(Klijent).

2.2 UGOVORI O SISTEMSKIM OPERACIJAMA

UGOVOR 1:**Izvrši**

Operacija : Izvrši(Klijent):signal;
Veza sa SK : Dijagram sekvenci sk: Evidentiraj novog klijenta u hemijsko čišćenje.
Preduslovi : -
Postuslovi: Pročitan je zadnji zapamćen klijent.

UGOVOR 2:**UbaciNovog**

Operacija: UbaciNovog(Klijent):signal;
Veza sa SK: Dijagram sekvenci sk: Evidentiranje novog klijenta u hemijsko čišćenje.
Preduslovi: -
Postuslovi. Evidentiran je novi klijent.

UGOVOR 3: **Računaj**

Operacija: Računaj(Klijent):signal;
Veza sa SK: Dijagram sekvenci sk: Evidentiranje novog klijenta u hemijsko čišćenje.
Dijagram sekvenci sk: Promena podataka klijenta.
Preduslovi: Ako je klijent obrađen ili storniran ne može se izvršiti sistemska operacija.
Postuslovi: Izračunata je vrednost svake usluge
Izračunata je ukupna vrednost.

UGOVOR 4: **Obradi**

Operacija: Obradi(Klijent):signal;
Veza sa SK: Dijagram sekvenci sk: Evidentiranje novog klijenta u hemijsko čišćenje.
Dijagram sekvenci sk: Promena podataka klijenta.
Preduslovi. Ako je klijent obrađen ili storniran ne može se izvršiti sistemska operacija.
Postuslovi: Izračunata je vrednost svake usluge
Izračunata je ukupna vrednost
Klijent je obrađen.

UGOVOR 5: **Pretraži**

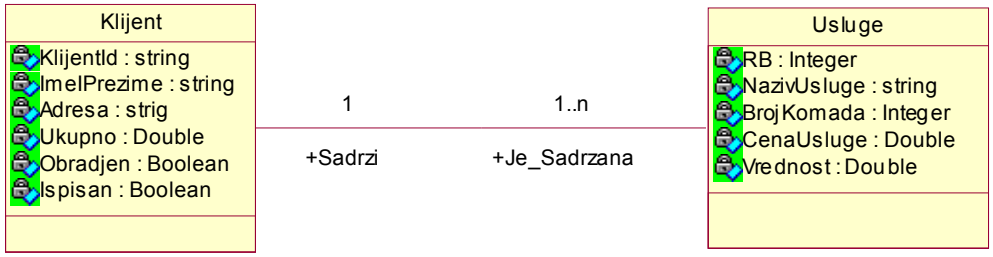
Operacija: Pretraži(Klijent):signal;
Veza sa SK: Dijagram sekvenci sk: Provera postojanja klijenta
Dijagram sekvenci sk: Promena podataka klijenta.
Dijagram sekvenci sk:Storniranje klijenta iz hemijskog čišćenja.
Preduslovi: -
Postuslovi: Pročitan je klijent ukoliko postoji.

UGOVOR 6: **Ispiši**

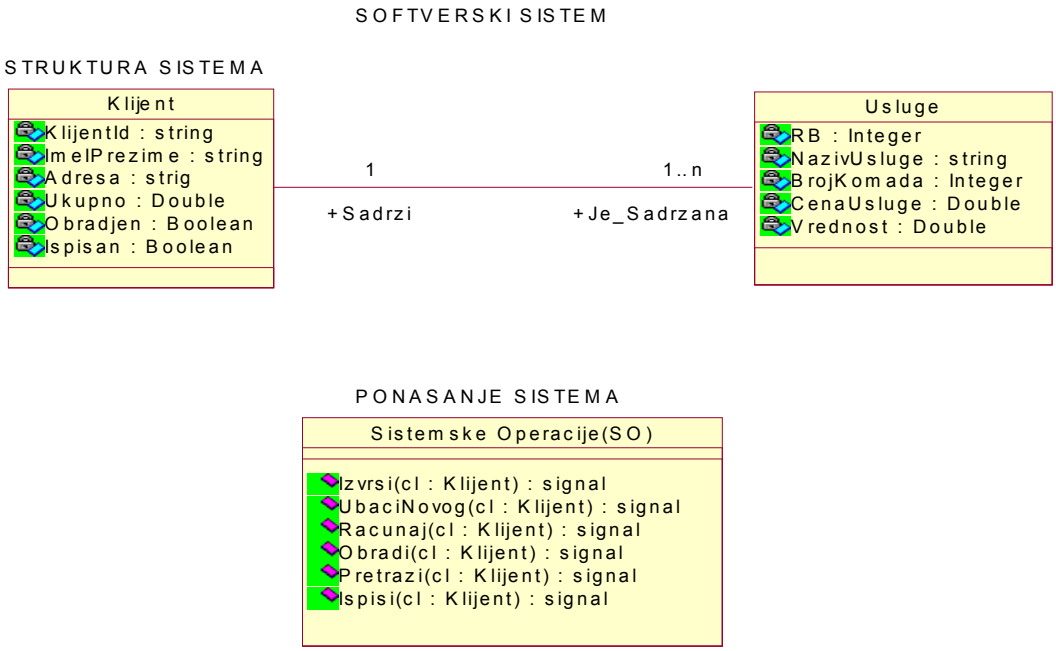
Operacija: Ispiši(Klijent):signal;
Veza sa SK: Dijagram sekvenci sk:Storniranje klijenta iz hemijskog čišćenja.
Preduslov:i -
Postuslovi: Klijent je storniran.

2.3. KONCEPTUALNI MODEL

Konceptualni model opisuje konceptualne klase- domenske objekte domena problema i veze između konceptualnih klasa.



Rezultat analize scenarija slučajeva korišćenja i pravljenja konceptualnog modela je logička struktura i ponašanje softverskog sistema.



2.4. RELACIONI MODEL

Na osnovu konceptualnog modela formiran je relacioni model:

Klijent(KlijentId, ImeIPrezime, Ukupno, Obrađen, Ispisan);
Usluge(KlijentId,RB, NazivUsluge, BrojKomada, CenaUsluge, Vrednost).

3.FAZA PROJEKTOVANJA

3.1. ARHITEKTURA SOFTVERSKOG SISTEMA

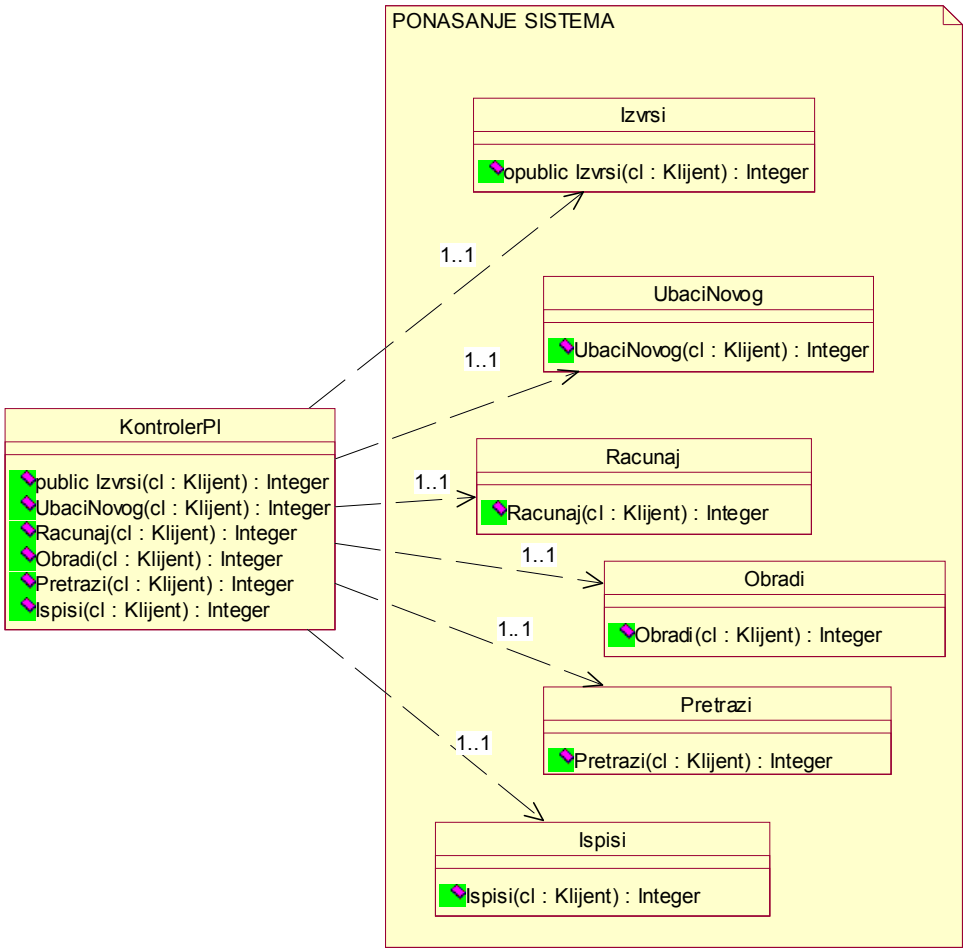
U ovom radu je korišćena klasična tro-nivojska arhitektura koja obuhvata:

1. Korisnički interfejs koji predstavlja ulazno-izlanu reprezentaciju softverskog sistema.
2. Aplikacionu logiku koja opisuje strukturu i ponašanje softverskog sistema.
3. Skladište podataka koji čuva stanje atributa softverskog sistema.

Projektovani su svi od navedenih elemenata tro-nivojske arhitekture:

1. Kontroler
2. Poslovna logika- domenske klase
3. Poslovna logika- sistemske operacije
4. Databasebroker
5. Skladište podataka
6. Korisnički interfejs.

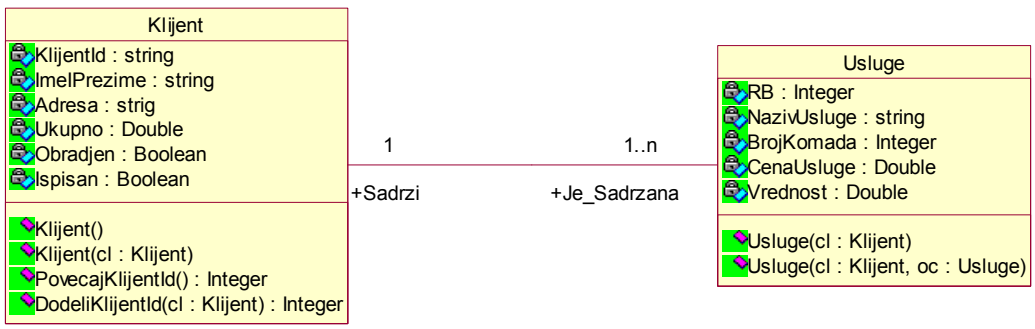
3.2. PROJEKTOVANJE APLIKACIONE LOGIKE – KONTROLER



Dijagram klasa koji prikazuje odnos između kontrolera i klasa odgovornih za izvrsenje SO

3.3. PROJEKTOVANJE DOMENSKIH KLASA

Softverske klase strukture prave se na osnovu konceptualnih klasa.



Dijagram softverskih klasa strusture

3.4. APLIKACIONA LOGIKA- POSLOVNA LOGIKA- SISTEMSKE OPERACIJE

Za svaki od ugovora projektuju se konceptualna rešenja.
Konceptualne realizacije se mogu opisati preko objektnog pseudokoda (koji je dat u sledećem poglavlju), dijagrama saradnje, sekvencnih dijagrama, dijagrama aktivnosti, dijagrama prelaza stanja ili dijagrama strukture.

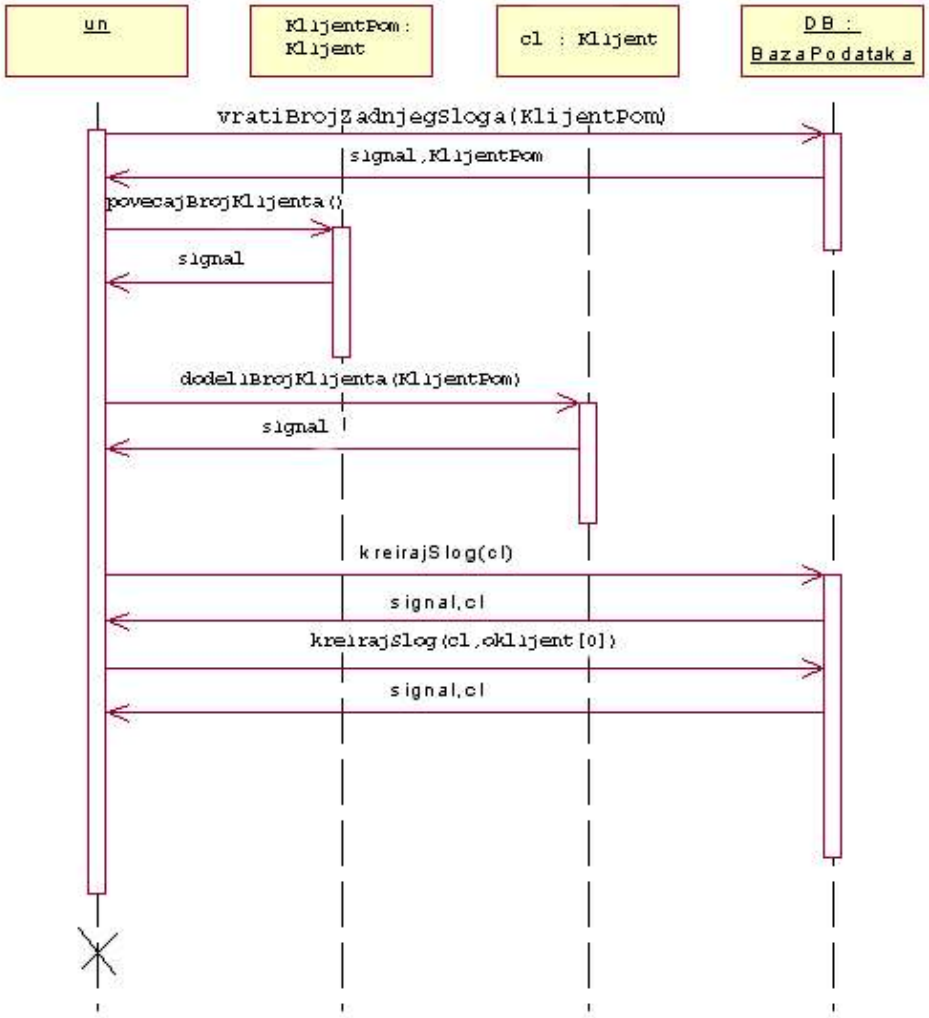
UGOVOR 1: Izvrši

Operacija: Izvrši(Klijent): signal;
Postuslovi: Pročitan je zadnji upisan klijent

Sekvencni dijagram SO Izvrši

UGOVOR 2: UbaciNovog

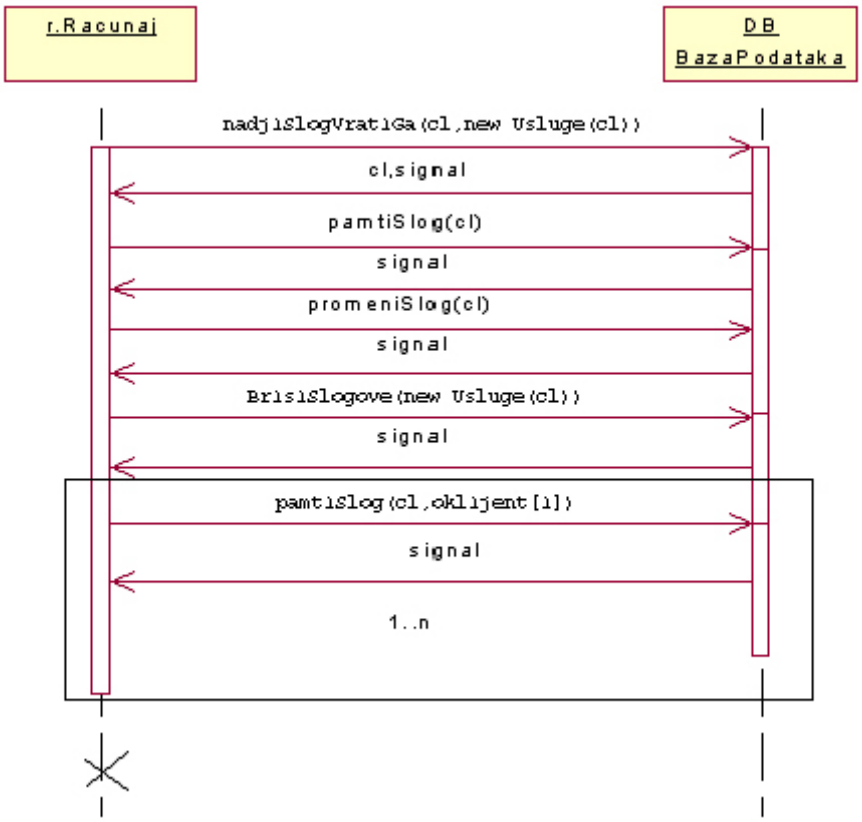
Operacija : UbaciNovog(Klijent):signal
Postuslovi: Upisan je novi klijent.



Sekvencni dijagram SO UbaciNovi

UGOVOR 3: Računaj

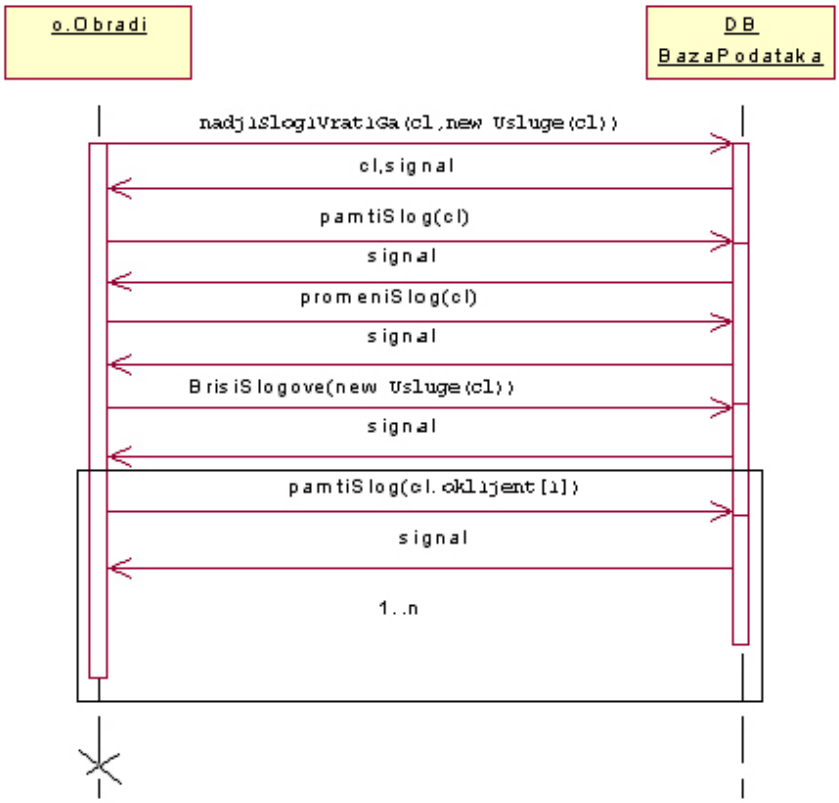
Operacija : Računaj(Klijent):signal
Postuslovi: Izračunata je vrednost svake usluge kao i ukupna vrednost za klijenta.



Sekvencni dijagram SO Računaj

UGOVOR 4: Obradi

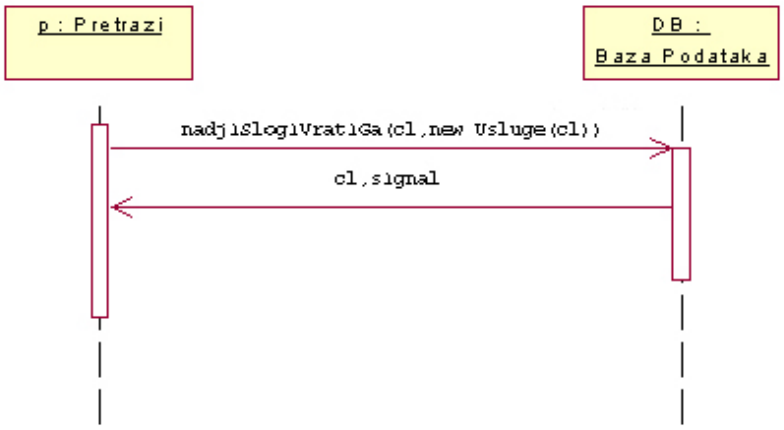
Operacija : Obradi(Klijent):signal
Postuslovi: Izračunata je vrednost svake usluge,
Izračunata je ukupna vrednost za klijenta,
Klijent je obrađen.



Sekvencni dijagram SO Obradi

UGOVOR 5: Pretraži

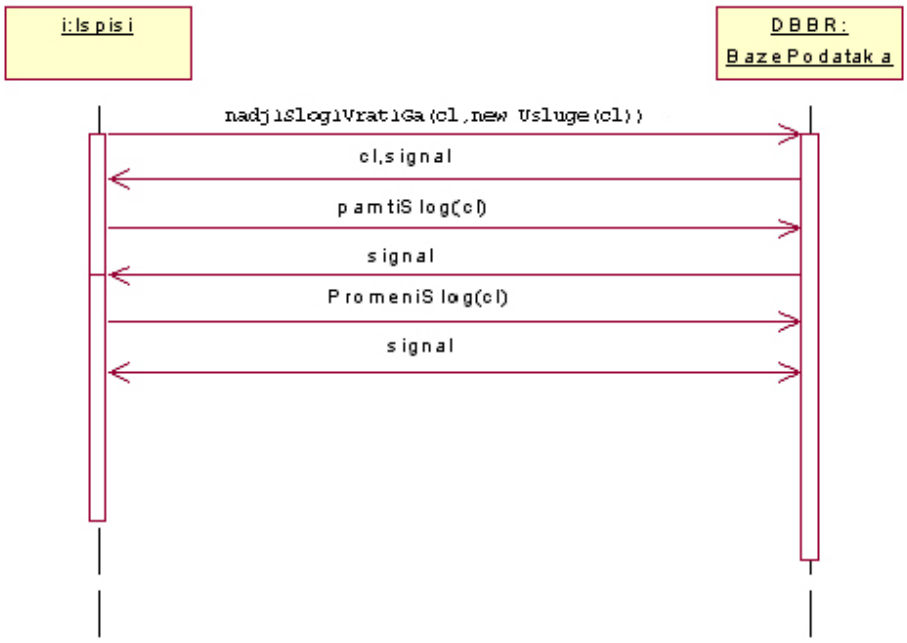
Operacija : Pretraži(Klijent):signal
Postuslovi: Pročitan je klijent ukoliko postoji.



Sekvencni dijagram SO Pretraži

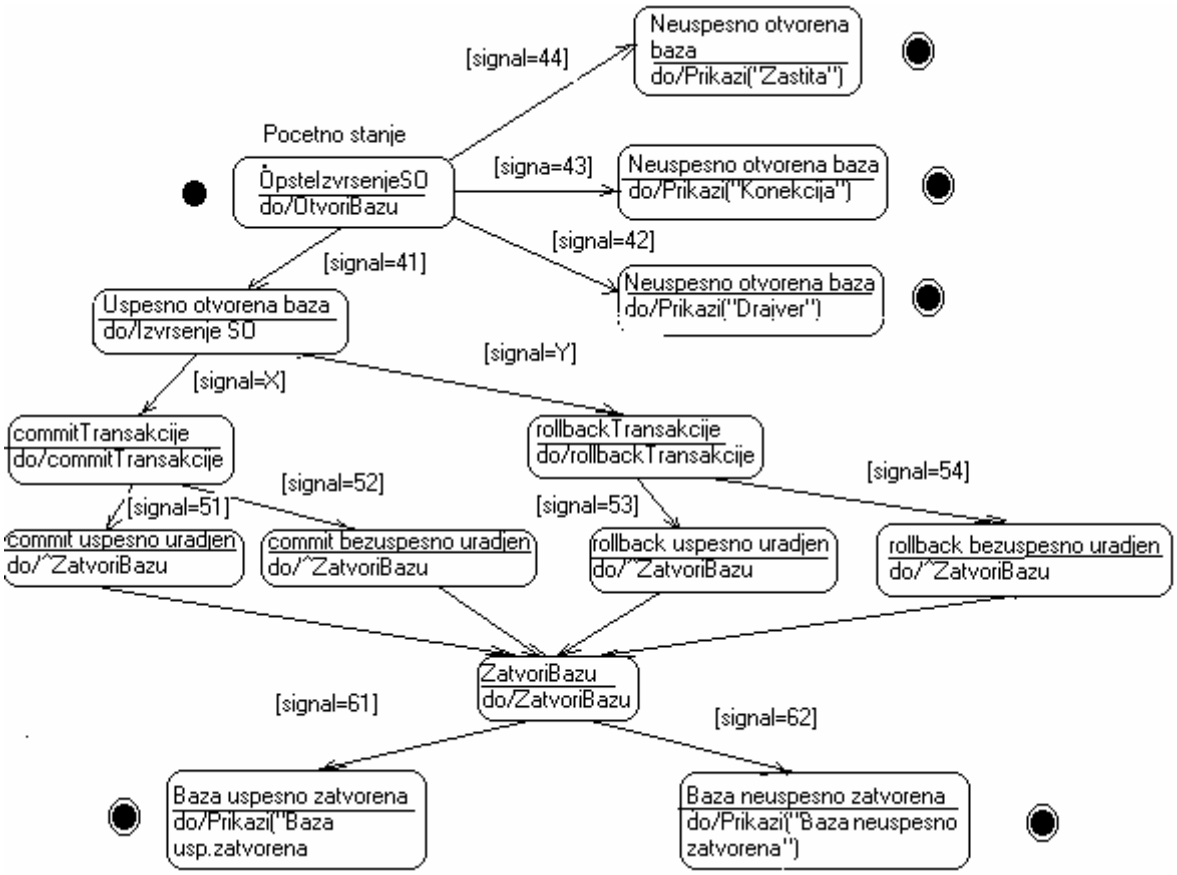
UGOVOR 6: Ispiši

Operacija : Ispiši(Klijent):signal
Postuslovi: Klijent je storniran.

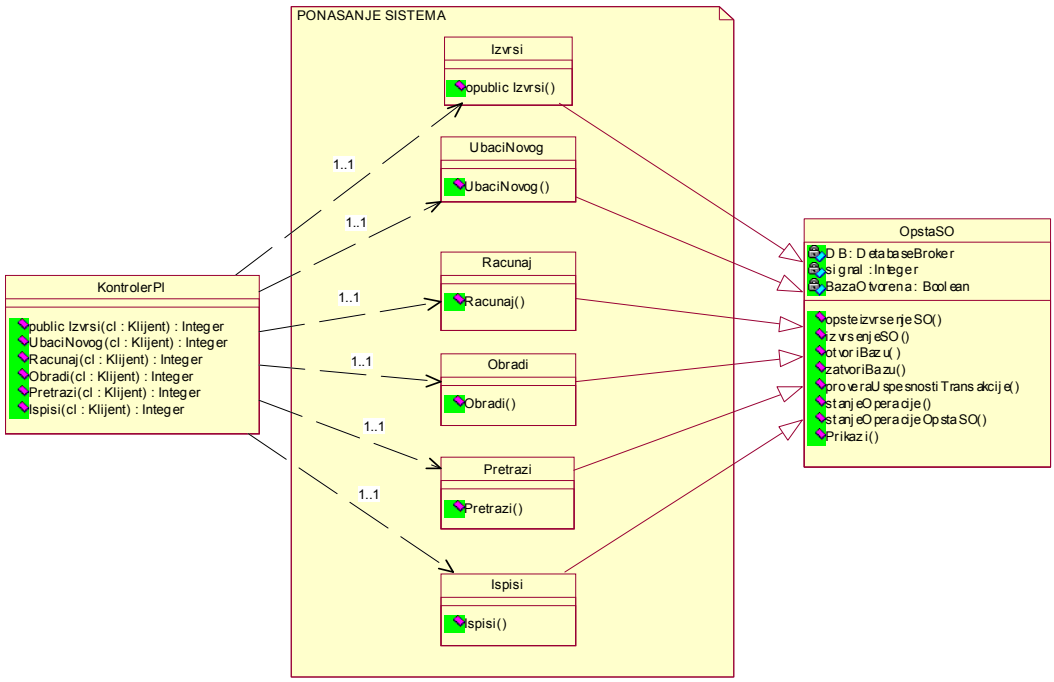


Sekvencni dijagram SO Ispisi

Svaka od SO treba da nasledi klasu OpštaSO koja je odgovorna za konekciju sa bazom I za kontrolu izvršenja transakcije.Metoda koja to obezbeđuje zove se opšteIzvršenjeSO().



Dijagram prelaza stanja metode opšteIzvršenjeSO()



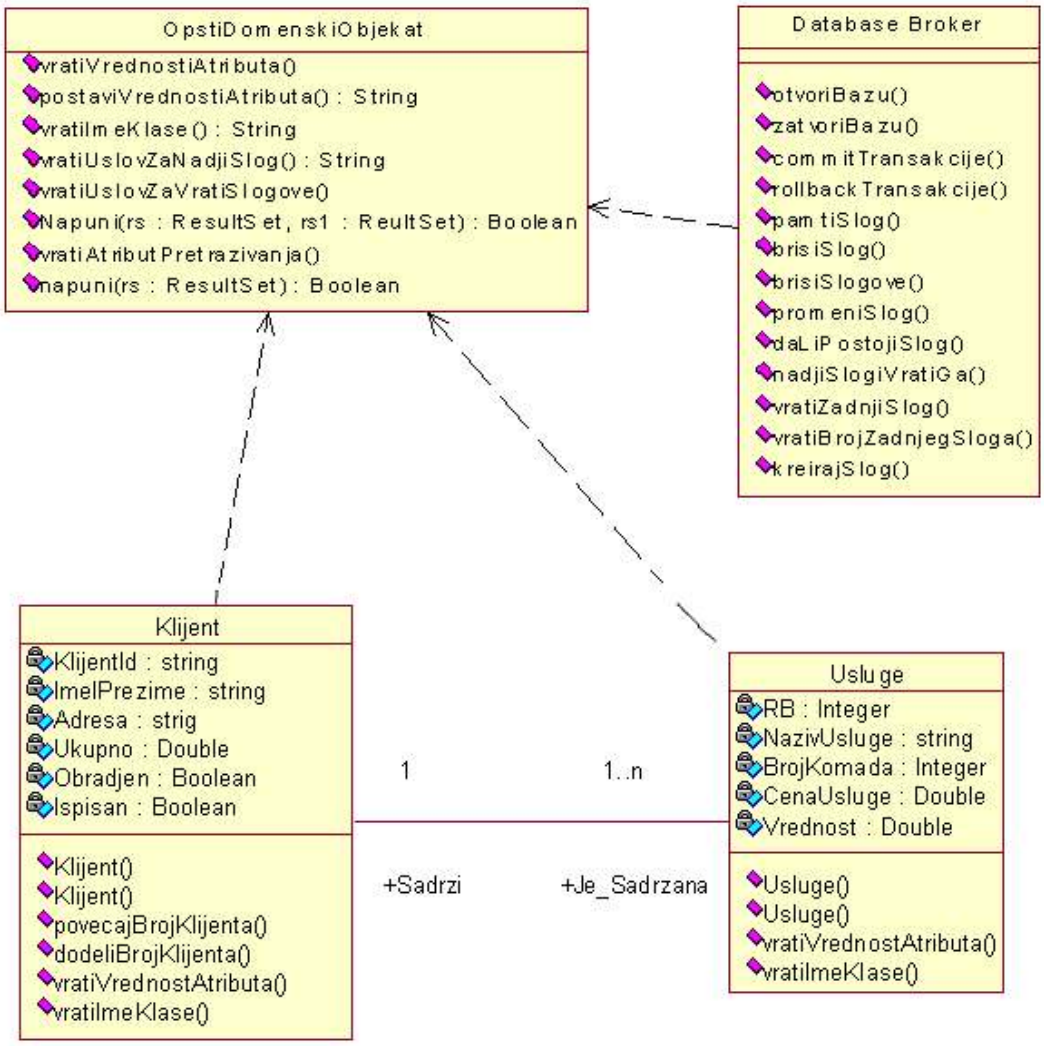
Klase koje su odgovorne za SO nasleđuju klasu OpštaSO

3.5. PROJEKTOVANJE APLIKACIONE LOGIKE-DATABASE BROKER

Projektovana je klasa DatabaseBroker, kao perzistentni okvir koji realizuje metode:

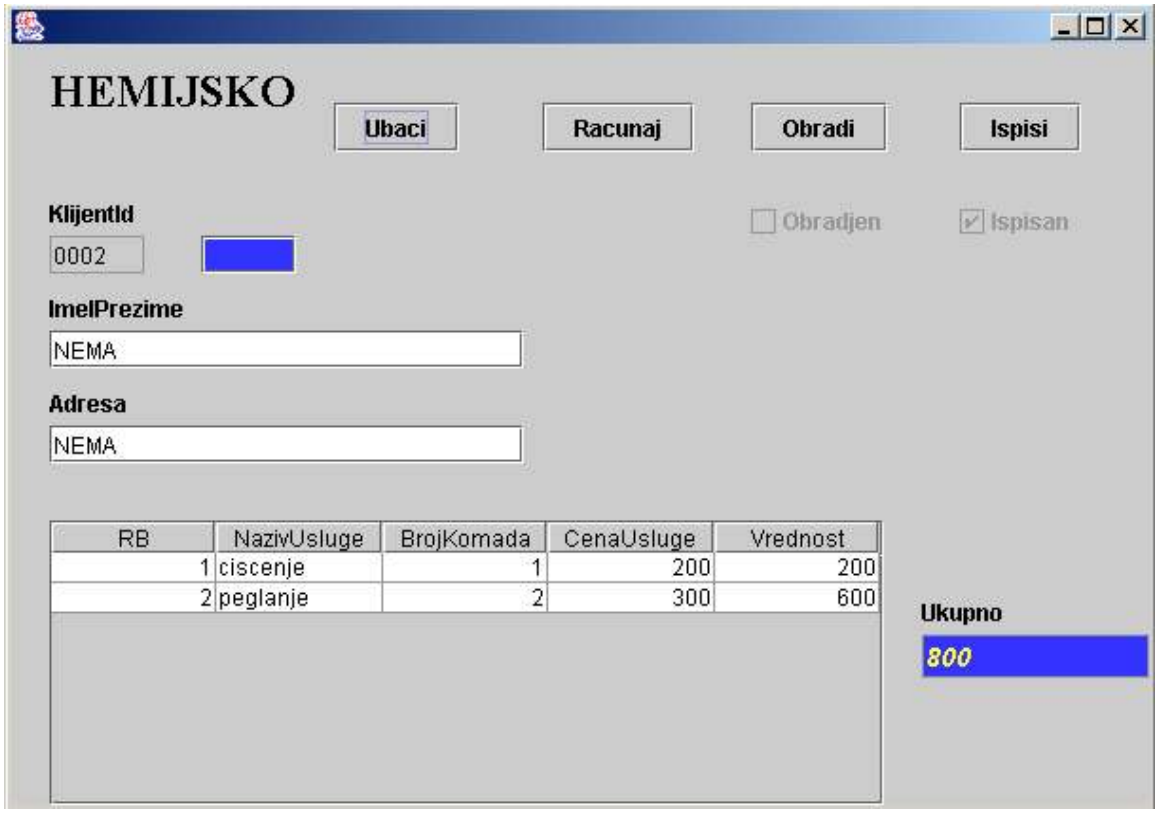
1. int otvoriBazu()
2. int zatvoriBazu()
3. int commitTransakcije()
4. int rollbackTransakcije()
5. int pamtiSlog(Objekat)
6. int brisiSlog(Objekat)
7. int brisiSlogove(Objekat)
8. int promeniSlog(Objekat)
9. int daLiPostojiSlog(Objekat)
10. int nađiSlogiVratiga(Objekat,Objekat)
11. int vratiZadnji Slog(Objekat,Objekat)
12. int vratiBrojZadnjegSloga(Objekat)
13. int kreirajSlog(Objekat).

U procesu pravljenja generičkih metoda DatabaseBroker klase dobili smo metode interfejsa OpštiDomenskiObjekat.



Database broker klasa se povezuje sa klasom OpštiDomenskiObjekat

3.6. STRUKTURA KORISNIČKOG INTERFEJSA



Svaki scenario SK koji se izvodi nad ekranskom formom se projektuje.

<u>SLUČAJ KORIŠĆENJA1</u>- EVIDENTIRANJE NOVOG KLIJENTA U HEMIJSKO ČIŠĆENJE

Preduslov: Sistem je uključen I radnik je ulogovan pod svojom šifrom.Sistem prikazuje formu za obradu klijenta.

Napomena: Polazimo od pretpostavke da je baza podataka u početku prazna.

Osnovni scenarijo SK
1. Radnik poziva sistem da ubaci novog klijenta.

<u>Opis akcije</u>: Radnik pritiska(jedan klik miša) dugme Ubaci nakon čega se poziva sistemska operacija UbaciNovog() koja pravi novog klijenta.

2. Sistem ubacuje novog klijenta.

3.Sistem prikazuje radniku novog klijenta.
Pre nego što se na ekranskoj formi prikaže novi klijent treba da bude prikazana poruka: Slog je uspešno kreiran i zapamćen u bazi ili slog nije uspešno kreiran i zapamćen u bazi u zavisnosti od uspešnosti izvršenja operacije.

4.Radnik unosi podatke o klijentu.
5.Radnik poziva sistem da izračuna iznose usluga i ukupnu vrednost usluge .

Opis akcije: Radnik pritiska dugme Računaj ili pritiska tipku /Enter/, dok unosi podatke u poljima usluga, nakon ćega se poziva sistemska operacija Računaj() koja računa Cenu po svakoj usluzi i Ukupnu vrednost.

6.Sistem računa iznose po svakoj usluzi i ukupnu vrednost.
7.Sistem pokazuje radniku izmenjenog klijenta.

Pre nego što se na ekranskoj formi prikaže klijent sa promenjenim vrednostima navedenih polja treba da bude prikazana poruka:Slog je uspešno zapamćen u bazi ili slog nije uspešno zapamćen u bazi u zavisnosti od uspešnosti izvršenja operacije.

8.Radnik kontroliše da li je uneo sve potrebne podatke za klijenta.
9.Radnik poziva sistem da izvrši konačnu obradu klijenta.

Opis akcije: Radnik pritiska dugme Obradi nakon čega se poziva sistemska operacija Obradi().

10.Sistem obrađuje klijenta.
11.Sistem pokazuje radniku obrađenog klijenta.

Pre nego što se na ekranskoj formi prikaže klijent sa promenjenim poljem Obrađen treba da bude prikazana poruka:Slog je uspešno zapamćen u bazi ili slog nije uspešno zapamćen u bazi u zavisnosti od uspešnosti izvršenja operacije.

Polje Obrađen nakon uspešno izvršene obrade klijenta biva označeno.

Alternativna scenarija izvršenja SK:

- 5.1. Ukoliko sistem ne može da izračuna iznose usluga i ukupnu vrednost, prikazuje se poruka radniku da ne može da izračuna potrebne podatke. Prekida se izvršenje scenarija.
- 9.1. Ukoliko sistem ne može da obradi podatke prikazuje se radniku poruka da ne može da obradi podatke. Prekida se izvršenje scenarija.

SLUČAJ KORIŠĆENJA 2: PROVERA POSTOJANJA KLIJENTA

Preduslov: Sistem je uključen I radnik je ulogovan pod svojom šifrom.Sistem prikazuje formu za obradu klijenta.

Osnovni scenarijo SK
1.Radnik unosi broj klijenta koji želi da proveri.
Opis akcije: Radnik unosi broj klijenta u polje koje ima plavu pozadinu.

2. Radnik poziva sistem da proveri da li postoji klijent sa zadatim brojem.

Opis akcije: Unos broja klijenta treba završiti pritiskom na tipku /Enter/ nakon čega se poziva sistemska operacija Pretraži().

3. Sistem proverava postojanje klijenta.
4.Sistem prikazuje radniku klijenta.

Ukoliko je operacija uspešno izvršena na ekranskoj formi se prikazuje traženi klijent, u suprotnom se prikazuje ekranska forma sa podrazumevanim vrednostima.

Alternativna scenarija izvršenja SK:

- 3.1. Ukoliko ne postoji klijent sa zadatim brojem sistem prikazje radniku poruku da osoba nije upisana u HEMIJSKO ČIŠĆENJE. Prekida se izvršenje scenarija.

SLUČAJ KORIŠĆENJA 3: PROMENA PODATAKA KLIJENTA

Preduslov: Sistem je uključen i radnik je ulogovan pod svojom šifrom.Sistem prikazuje formu za obradu klijenta.

Osnovni scenario izvršenja SK:
1. Radnik unosi broj klijenta čije usluge želi da promeni.

Opis akcije: Radnik unosi broj klijenta u polje pretraživanja koje ima plavu pozadinu.

2. Radnik poziva sistem da proveri da li postoji klijent sa zadatim brojem.

Opis akcije: Unos broja klijenta treba završiti pritiskom na tipku /Enter/ nakon čega se poziva sistemska operacija Pretraži().

3. Sistem proverava postojanje klijenta.
4. Sistem prikazuje radniku podatke o klijentu.
5. Radnik unosi dopunske/izmenjene podatke u evidenciju o klijentu.
6. Radnik poziva sistem da izračuna iznose po uslugama i ukupnu vrednost.

Opis akcije: Radnik pritiska dugme Računaj ili pritiska tipku /Enter/, dok unosi podatke u polja usluge klijenta, nakon čega se poziva sistemska operacija Računaj() koja računa cenu svakoj usluge klijenta i Ukupnu vrednost.

7. Sistem računa iznose po uslugama i ukupnu vrednost.

8. Sistem prikazuje radniku izmenjene podatke.
9. Radnik kontroliše da li je uneo sve potrebne podatke.
10. Radnik poziva sistem da izvrši konačnu obradu podataka.
Opis akcije: Radnik pritiska dugme Obradi nakon čega se poziva sistemska operacija Obradi().

11. Sistem obrađuje podatke.
12. Sistem javlja radniku da je obradio podatke.

Alternativna scenarija izvršenja SK:

- 3.1. Ukoliko ne postoji klijent sa zadatim brojem sistem prikazje radniku poruku da klijent nije upisan u HEMIJSKO ČIŠĆENJE. Prekida se izvršenje scenarija.
- 6.1. Ukoliko sistem ne može da izračuna iznose po uslugama i ukupnu vrednost, on prikazuje radniku poruku da ne može da izračuna. Prekida se izvršenje scenarija.
- 10.1. Ukoliko sistem ne može da obradi podatke prikazuje se radniku poruka da ne može da obradi podatke. Prekida se izvršenje scenarija.

SLUČAJ KORIŠĆENJA 4: ISPIS KLIJENTA IZ HEMIJSKOG ČIŠĆENJA

Preduslov: Sistem je uključen i radnik je ulogovan pod svojom šifrom.Sistem prikazuje formu za obradu klijenta.

Osnovni scenario izvršenja SK:
1. Radnik unosi broj klijenta koje želi da ispiše iz HEMIJSKOG ČIŠĆENJA.

Opis akcije:Radnik unosi broj klijenta u polje pretraživanja koje ima plavu pozadinu.

2. Radnik poziva sistem da proveri da li postoji klijent sa zadatim brojem u bazi.

Opis akcije: Unos broja klijenta treba završiti pritiskom na tipku /Enter/ nakon čega se poziva sistemska operacija Pretraži().

3. Sistem proverava postojanje klijenta.
4. Sistem prikazuje radniku podatke o klijentu ukoliko postoji.
5. Radnik poziva sistem da ispiše klijenta iz HEMIJSKOG ČIŠĆENJA.

Opis akcije: Radnik pritiska dugme Ispiši nakon čega se poziva sistemska operacija Ispiši() koja vrši ispis klijenta iz hemijskog čišćenja.

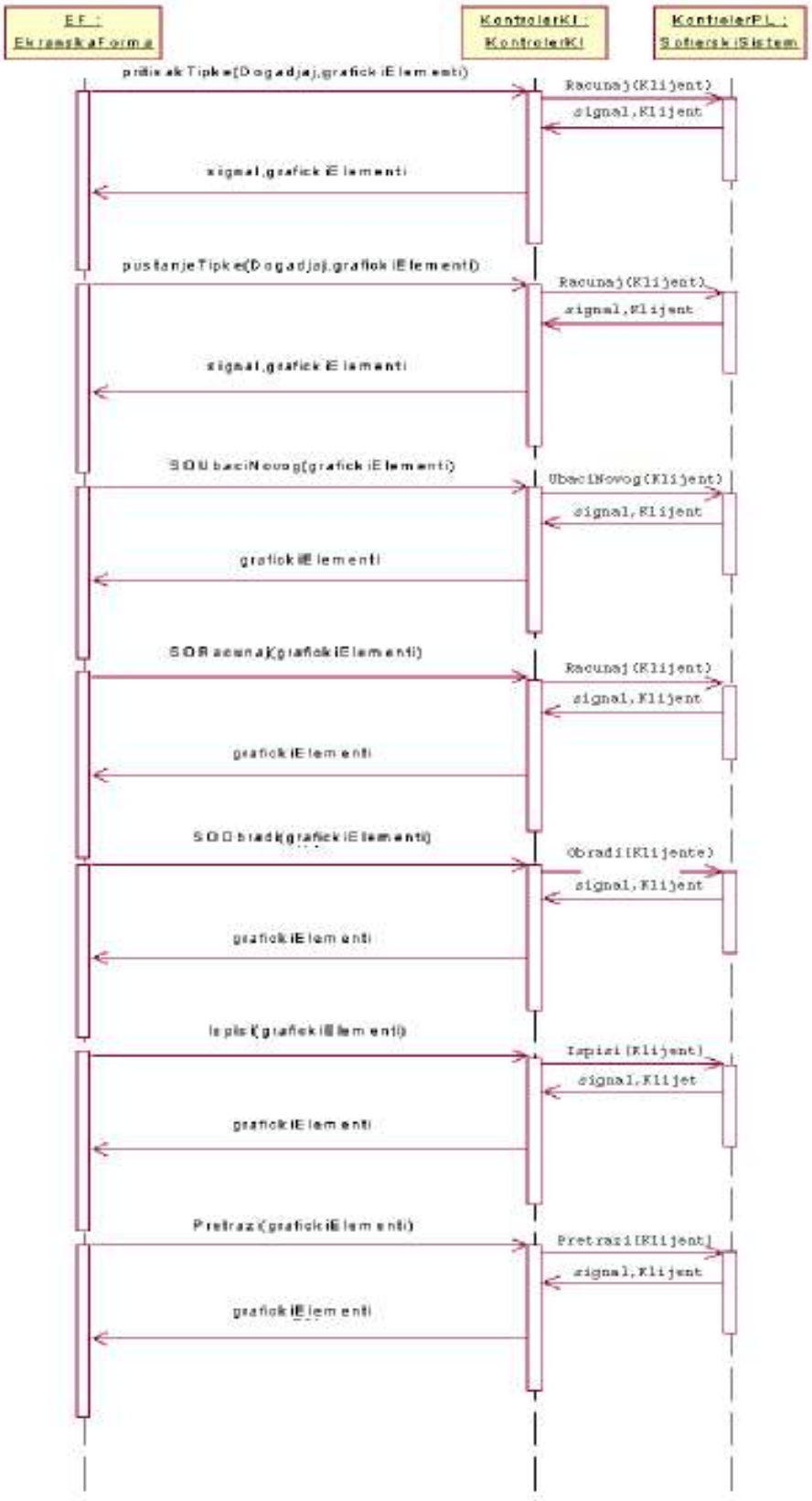
6. Sistem ispisuje klijenta.
7. Sistem prikazuje radniku podatke o ispisanom klijentu.

Pre nego što se na ekranskoj formi prikaže klijent sa promenjenim poljem Ispisan treba da bude prikazana poruka:Slog je uspešno promenjen u bazi ili slog nije uspešno promenjen u bazi u zavisnosti od uspešnosti izvršenja operacije.
Polje Ispisan nakon uspešno izvršenog ispisa klijenta biva označeno.
Alternativna scenarija izvršenja SK:

- 2.1. Ukoliko ne postoji klijent sa zadatim brojem sistem prikazje radniku poruku da klijent nije upisan u HEMIJSKO ČIŠĆENJE. Prekida se izvršenje scenarija.
- 5.1. Ukoliko klijent ne može da se ispiše sistem prikazuje radniku poruku o nemogućnosti ispisa. Prekida se izvršenje scenarija.

Nakon projektovanja korisnočkog interfejsa dobija se sledeći dijagram klasa

I sekvencni dijagram:


EF : EkranskaForma
KontrolerKI : KontrolerKI
KontrolerPL : SoftverskiSistem
pritisakTipke(Dogadjaj,grafickiElementi)
Racunaj(Klijent)
signal,Klijent
signal,grafickiElementi
pustanjeTipke(Dogadjaj,grafickiElement)
Racunaj(Klijent)
signal,Klijent
signal,grafickiElementi
SOUbaciNovog(grafickiElementi)
UbaciNovog(Klijent)
signal,Klijent
grafickiElementi
SORacunaj(grafickiElementi)
Racunaj(Klijent)
signal,Klijent
grafickiElementi
SOObradi(grafickiElementi)
Obradi(Klijente)
signal,Klijent
grafickiElementi
Ispisi(grafickiElementi)
Ispisi(Klijent)
signal,Klijet
grafickiElementi
Pretrazi(grafickiElementi)
Pretrazi(Klijent)
signal,Klijent
grafickiElementi

4. IMPLEMENTACIJA

AbsoluteConstraints.java

```
import java.awt.Dimension;
import java.awt.Point;

class AbsoluteConstraints implements java.io.Serializable {
   static final long serialVersionUID = 5261460716622152494L;

   /** The X position of the component */
   public int x;
   /** The Y position of the component */
   public int y;
   /** The width of the component or -1 if the component's preferred width should be
used */
   public int width = -1;
   /** The height of the component or -1 if the component's preferred height should be
used */
   public int height = -1;
   public AbsoluteConstraints(Point pos) {
      this (pos.x, pos.y);}
   public AbsoluteConstraints(int x, int y) {
      this.x = x;
      this.y = y;}
   public AbsoluteConstraints(Point pos, Dimension size) {
      this.x = pos.x;
      this.y = pos.y;
      if (size != null) {
         this.width = size.width;
         this.height = size.height; }
 }
   public AbsoluteConstraints(int x, int y, int width, int height) {
      this.x = x;
      this.y = y;
      this.width = width;
      this.height = height;
   }
   public int getX () {
      return x;
   }
   public int getY () {
      return y;
   }
   public int getWidth () {
```

```
        return width;
    }
    public int getHeight () {
        return height;
    }
    public String toString () {
        return super.toString ()
        +" [x="+x+", y="+y+", width="+width+", height="+height+"]";
    }

}
```

AbsoluteLayout.java

```
import java.awt.*;

class AbsoluteLayout implements LayoutManager2, java.io.Serializable {
    static final long serialVersionUID = -1919857869177070440L;

    public void addLayoutComponent(String name, Component comp) {
        throw new IllegalArgumentException();
    }
    public void removeLayoutComponent(Component comp) {
        constraints.remove(comp);
    }
    public Dimension preferredLayoutSize(Container parent) {
        int maxWidth = 0;
        int maxHeight = 0;
        for (java.util.Enumeration e = constraints.keys(); e.hasMoreElements();) {
            Component comp = (Component)e.nextElement();
            AbsoluteConstraints ac = (AbsoluteConstraints)constraints.get(comp);
            Dimension size = comp.getPreferredSize();

            int width = ac.getWidth ();
            if (width == -1) width = size.width;
            int height = ac.getHeight ();
            if (height == -1) height = size.height;

            if (ac.x + width > maxWidth)
                maxWidth = ac.x + width;
            if (ac.y + height > maxHeight)
                maxHeight = ac.y + height;
        }
        return new Dimension (maxWidth, maxHeight);
```

```
}
public Dimension minimumLayoutSize(Container parent) {
   int maxWidth = 0;
   int maxHeight = 0;
   for (java.util.Enumeration e = constraints.keys(); e.hasMoreElements();) {
      Component comp = (Component)e.nextElement();
      AbsoluteConstraints ac = (AbsoluteConstraints)constraints.get(comp);
      Dimension size = comp.getMinimumSize();

      int width = ac.getWidth ();
      if (width == -1) width = size.width;
      int height = ac.getHeight ();
      if (height == -1) height = size.height;

      if (ac.x + width > maxWidth)
         maxWidth = ac.x + width;
      if (ac.y + height > maxHeight)
         maxHeight = ac.y + height;
   }
   return new Dimension (maxWidth, maxHeight);
}
public void layoutContainer(Container parent) {
   for (java.util.Enumeration e = constraints.keys(); e.hasMoreElements();) {
      Component comp = (Component)e.nextElement();
      AbsoluteConstraints ac = (AbsoluteConstraints)constraints.get(comp);
      Dimension size = comp.getPreferredSize();
      int width = ac.getWidth ();
      if (width == -1) width = size.width;
      int height = ac.getHeight ();
      if (height == -1) height = size.height;

      comp.setBounds(ac.x, ac.y, width, height);
   }
}
public void addLayoutComponent(Component comp, Object constr) {
   if (!(constr instanceof AbsoluteConstraints))
      throw new IllegalArgumentException();
   constraints.put(comp, constr);
}
public Dimension maximumLayoutSize(Container target) {
   return new Dimension(Integer.MAX_VALUE, Integer.MAX_VALUE);
}

public float getLayoutAlignmentX(Container target) {
   return 0;
}
```

```
    public float getLayoutAlignmentY(Container target) {
        return 0;
    }
    public void invalidateLayout(Container target) {
    }
        protected java.util.Hashtable constraints = new java.util.Hashtable();
}
```

DomenskiObjekti.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;


class Klijent implements OpstiDomenskiObjekat
{
 String KlijentId;
 String ImeIPrezime;
 String Adresa;
 Double Ukupno;
 boolean Obradjen;
 boolean Ispisan;
 Usluge []oklijent;


Klijent()
  { KlijentId = "NEMA";
    ImeIPrezime= "NEMA";
    Adresa="NEMA";
    Ukupno= new Double(0);
    Obradjen = false;
    Ispisan=false;
    oklijent = new Usluge[1];
    oklijent[0] = new Usluge(this);
  }


Klijent(Klijent cl)
  { KlijentId= new String(cl.KlijentId);
    ImeIPrezime = new String(cl.ImeIPrezime);
    Adresa = new String(cl.Adresa);
    Ukupno = new Double(cl.Ukupno.doubleValue());
    Obradjen = cl.Obradjen;
```

```
    Ispisan= cl.Ispisan;
    oklijent = new Usluge[cl.oklijent.length];
    for(int i=0;i<cl.oklijent.length;i++)
      { oklijent[i] = new Usluge(this,cl.oklijent[i]);
      }

   }


 void Napuni(JFormattedTextField KlijentId1, JFormattedTextField ImeIPrezime1,
JFormattedTextField Adresa1,JFormattedTextField Ukupno1,JCheckBox
Obradjen1,JCheckBox Ispisan1)
      { KlijentId = (String)KlijentId1.getValue();
        ImeIPrezime = (String)ImeIPrezime1.getValue();
        Adresa = (String)Adresa1.getValue();
        Ukupno = (Double)Ukupno1.getValue();
        Obradjen = Obradjen1.isSelected();
        Ispisan = Ispisan1.isSelected();
          }

 void Vrati(JFormattedTextField KlijentId1, JFormattedTextField
ImeIPrezime1,JFormattedTextField Adresa1,JFormattedTextField Ukupno1,JCheckBox
Obradjen1,JCheckBox Ispisan1)
      { KlijentId1.setValue(KlijentId);
        ImeIPrezime1.setValue(ImeIPrezime);
        Adresa1.setValue(Adresa);
        Ukupno1.setValue(Ukupno);
        Obradjen1.setSelected(Obradjen);
        Ispisan1.setSelected(Ispisan);
      }


      public java.lang.String vratiVrednostiAtributa()
  { return ""+ KlijentId+ "', '" + ImeIPrezime + "','"+ Adresa +"',"
+Ukupno.doubleValue() + ", " + Obradjen + ", " +Ispisan+"";}

  public java.lang.String postaviVrednostiAtributa()
  { return "KlijentId = '"+ KlijentId + "',ImeIPrezime ='" + ImeIPrezime +
"',Adresa='"+Adresa+"',Ukupno = " +
           Ukupno.doubleValue() + ", Obradjen = " + Obradjen + ",Ispisan = "+Ispisan+"";
  }

  public java.lang.String vratiImeKlase() { return " Klijent ";}

public java.lang.String vratiAtributPretrazivanja() { return "KlijentId";}
```

```
public java.lang.String vratiUslovZaNadjiSlog()
{ return " KlijentId = '"+ KlijentId + "'";
}

public java.lang.String vratiUslovZaNadjiSlogove()
  { return "";}

public boolean Napuni(ResultSet RSslog, ResultSet RSstavkeSlogova)
{ int i = 0;
  int BrojStavki = 0;
  ResultSet RSPom = RSstavkeSlogova;
  oklijent = null;

  try {
                    if (RSslog.next()==false)
                       { KlijentId = "NEMA";
                         ImeIPrezime = "NEMA"; Ukupno = new Double(0); Obradjen = false;
Ispisan = false;
                         oklijent= new Usluge[1];
                         oklijent[0]=new Usluge(this);
                         oklijent[0].RB = new Integer(0);
                         oklijent[0].NazivUsluge = "NEMA";
                         oklijent[0].BrojKomada = new Integer(0);
                   oklijent[0].CenaUsluge = new Double(0);
                         oklijent[0].Vrednost = new Double(0);
                         System.out.println("72");
                        return true;
                       }

             RSslog.previous();

                    while (RSPom.next())
          { BrojStavki++;}
        if (BrojStavki > 0 ) oklijent = new Usluge[BrojStavki];


                    RSslog.next();
                    KlijentId = RSslog.getString("KlijentId");
                    ImeIPrezime = RSslog.getString("ImeIPrezime");
        Adresa = RSslog.getString("Adresa");
        Ukupno = new Double(RSslog.getDouble("Ukupno"));
                    Obradjen = RSslog.getBoolean("Obradjen");
        Ispisan = RSslog.getBoolean("Ispisan");

                    RSstavkeSlogova.beforeFirst();
        while(RSstavkeSlogova.next())
```

```
        { oklijent[i] = new Usluge(this);
                        oklijent[i].RB = new Integer(RSstavkeSlogova.getInt("RB"));
                        oklijent[i].NazivUsluge = new String
(RSstavkeSlogova.getString("NazivUsluge"));
          oklijent[i].BrojKomada = new Integer(RSstavkeSlogova.getInt("BrojKomada"));
                        oklijent[i].CenaUsluge = new
Double(RSstavkeSlogova.getDouble("CenaUsluge"));
                        oklijent[i].Vrednost = new
Double(RSstavkeSlogova.getDouble("Vrednost"));
                        i++;
              }

                  } catch(Exception e) {System.out.println("73");
                        System.out.println(e);return false;}

      System.out.println("71");
            return true;
 }


public boolean Napuni(ResultSet RSslog)
{ int i = 0;
  int BrojStavki = 0;

  try {
                  if (RSslog.next()==false)
                     { KlijentId = "NEMA";}
                  else
                        { KlijentId = RSslog.getString("Max");
                          if (KlijentId == null) KlijentId = "NEMA";
                        }

                  } catch(Exception e) {
                        System.out.println(e);return false;}


            return true;
 }


int povecajKlijentId()
{ int broj;

  try {
          if (KlijentId.equals("NEMA") || KlijentId == null)
         {KlijentId = "0001";}
```

```
                else
                   if (KlijentId.equals("9999"))
                     { return 48;}
                         else
          {
                         broj = Integer.parseInt(KlijentId);
                         broj++;
                         KlijentId = String.valueOf(broj);

                          String       pom1= new String("");
                          for(int j=0; j<4-KlijentId.length(); j++)
                                { pom1 = pom1 + "0";
                                }
                         KlijentId = pom1 + KlijentId;

          }
    } catch(Exception e)
              { System.out.println("Izuzetak kod generisanja novog broja racuna: " + e);
return 48;}
   return 47;


}

int dodeliKlijentId(Klijent cl)
{   try {
           KlijentId = new String(cl.KlijentId);
             }
             catch(Exception e)
             { System.out.println("Nije dobro dodeljen klijent id " + e); return 56;}

   return 55;
}
}
class Usluge implements OpstiDomenskiObjekat
{ Integer RB;
  String NazivUsluge;
  Integer BrojKomada;
  Double CenaUsluge;
  Double Vrednost;
  Klijent cl;

  Usluge(Klijent cl1)
    { RB = new Integer(0);
      NazivUsluge = new String("NEMA");
      BrojKomada = new Integer(0);
```

```
   CenaUsluge = new Double(0);
   Vrednost = new Double(0);
   cl = cl1;}

Usluge(Klijent cl1,Usluge oc)
  { RB = new Integer(oc.RB.intValue());
    NazivUsluge = new String(oc.NazivUsluge);
    BrojKomada= new Integer(oc.BrojKomada.intValue());
    CenaUsluge = new Double(oc.CenaUsluge.doubleValue());
    Vrednost = new Double(oc.Vrednost.doubleValue());
    cl = cl1;}


 static String [] ZagSlogoviKlijenta()
        { String [] s = new String [5];
            s[0]=new String("RB");
            s[1]=new String("NazivUsluge");
            s[2]=new String("BrojKomada");
            s[3]=new String("CenaUsluge");
            s[4]=new String("Vrednost");
            return s;
     }

 static Class[] vratiTipove()
     { return new Class []
                  {Integer.class, String.class, Integer.class, Double.class, Double.class };
     }


 static Object[] vratiPocetneVrednosti()
     { Object [] obj = new Object [5];
            obj[0]=null;
            obj[1]=null;
            obj[2]=null;
            obj[3]=null;
            obj[4]=null;
            return obj;
     }


 void Napuni(int j,Object ob)
     {
       switch  (j)
         {
                 case 0:if (ob==null)
                        RB = new Integer(0);
```

```
                    else
                      RB = (Integer) ob;
                  break;

              case 1: if (ob==null)
                        NazivUsluge = (String) "NEMA";
                    else
                        NazivUsluge = (String) ob;
                      break;

                 case 2: if(ob==null)
                         BrojKomada = new Integer(0);
                       else
                      BrojKomada = (Integer)ob;
                   break;

                 case 3: if (ob==null)
                      CenaUsluge = new Double(0);
                        else
                         CenaUsluge = (Double) ob;
                        break;

                 case 4: if(ob==null)
                             Vrednost = new Double(0);
                        else
                             Vrednost = (Double) ob;

                           }
        }


Object Vrati(int j)
        {
      switch  (j)
        {
                 case 0:return (Object)RB;
                 case 1:return (Object)NazivUsluge;
                 case 2:return (Object)BrojKomada;
                 case 3:return (Object)CenaUsluge;
                 case 4:return (Object)Vrednost;
           }
         return null;
        }


String Prikazi()
```

```
      { return RB + " " + NazivUsluge + " " + BrojKomada + " " + CenaUsluge + " " +
Vrednost;}



public String vratiVrednostiAtributa()
  { return "'" + cl.KlijentId + "' , " + RB.intValue() + ", '" + NazivUsluge + "', " +
BrojKomada.intValue() + ", " + CenaUsluge.doubleValue() + ", " +
Vrednost.doubleValue();}

public String postaviVrednostiAtributa()
  { return "KlijentId= '"+ cl.KlijentId + "', RB = "+ RB.intValue() + ", NazivUsluge ='" +
NazivUsluge + "', BrojKomada = " +
          BrojKomada.intValue() + ", CenaUsluge = " + CenaUsluge.doubleValue() + ",
Vrednost = " + Vrednost.doubleValue();
  }

public String vratiImeKlase() { return "Usluge";}

public String vratiUslovZaNadjiSlog()
   { return " KlijentId = '"
      + cl.KlijentId + "' and RB = " + RB.intValue();
   }

 public String vratiUslovZaNadjiSlogove()
  { return " KlijentId = '" + cl.KlijentId + "'";}


public boolean Napuni(ResultSet RSslog, ResultSet RSstavkeSlogova)
{return false;}

public String vratiAtributPretrazivanja() { return "";}

public boolean Napuni(ResultSet RSslog)
{return false;}
}
```

DetabaseBroker.java

```
import java.io.*;
import java.sql.*;

 class DatabaseBroker
{
   static Connection con;
```

```
  static Statement stat;

  public int otvoriBazu(String imeBaze)
    {
          String Urlbaze;
          try {
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
       con = DriverManager.getConnection("jdbc:odbc:HEMIJA");
       con.setAutoCommit(false);
              } catch(ClassNotFoundException e) { System.out.println("Drajver nije
ucitan:" + e);return 42;}
                catch(SQLException esql) { System.out.println("Greska kod konekcije:" +
esql);return 43;}
                catch(SecurityException ese) {System.out.println("Greska zastite:" +
ese);return 44;}
                return 41;
              }

  public int zatvoriBazu()
    {   try {  con.close();}
          catch(Exception e) {System.out.println(e);return 62;} return 61; }

  int commitTransakcije()
    {   try{ con.commit();}
          catch(SQLException esql)
            {  System.out.println("Nije uspesno uradjen commit. " + esql);
                return 52;
                          }
      return 51;
    }

  int rollbackTransakcije()
    {   try{ con.rollback();}
          catch(SQLException esql)
            {  System.out.println("Nije uspesno uradjen rollback." +
             "Ako nije uradjen rollback sistem prelazi u nestabilno stanje. " + esql);
                return 54;
                          }
      return 53;
    }

  int pamtiSlog(OpstiDomenskiObjekat odo)
     { String upit;
       try{
            stat = con.createStatement();
            upit ="INSERT INTO " + odo.vratiImeKlase()  +
```

```
            " VALUES (" + odo.vratiVrednostiAtributa() + ")";
      System.out.println("Upis unosa stavke pre izvesenja:" + upit);
      stat.executeUpdate(upit);
      stat.close();
           }   catch(SQLException esql)
                 {  System.out.println("Nije uspesno zapamcen slog u bazi: " + esql);
                        return 32;
                              }
       return 31;
     }

  public int brisiSlog(OpstiDomenskiObjekat od)
     { String upit;

       try {

              stat = con.createStatement();
           upit ="DELETE *  FROM " + od.vratiImeKlase() + " WHERE " +
od.vratiUslovZaNadjiSlog();

           stat.executeUpdate(upit);
           stat.close();
           }   catch(SQLException esql) {
              System.out.println("Nije uspesno obrisan slog u bazi: " + esql);
                     return 34;
                                  }
        return 33;
     }


  public int brisiSlogove(OpstiDomenskiObjekat od)
     { String upit;

       try {

              stat = con.createStatement();
           upit ="DELETE *  FROM " + od.vratiImeKlase() + " WHERE " +
od.vratiUslovZaNadjiSlogove();

           stat.executeUpdate(upit);
           stat.close();
           }   catch(SQLException esql) {
              System.out.println("Nije uspesno obrisan slog u bazi: " + esql);
                     return 34;
                                  }
        return 33;
```

```
  }

public int promeniSlog(OpstiDomenskiObjekat od)
  { String upit;

    try {

          stat = con.createStatement();
       upit ="UPDATE " + od.vratiImeKlase() +
         " SET " + od.postaviVrednostiAtributa() +
         " WHERE " + od.vratiUslovZaNadjiSlog();
       System.out.println("PROMENI SLOG" + upit);
       stat.executeUpdate(upit);
       stat.close();
       }   catch(SQLException esql) {
              System.out.println("Nije uspesno promenjen slog u bazu: " + esql);
                    return 36;
                          }
   return 35;
  }

public int daLiPostojiSlog(OpstiDomenskiObjekat od)
  { String upit;
    ResultSet RSslogovi;

    try {

          stat = con.createStatement();
       upit ="SELECT *" +
         " FROM " + od.vratiImeKlase() +
         " WHERE " + od.vratiUslovZaNadjiSlog();
       RSslogovi = stat.executeQuery(upit);
       boolean signal = RSslogovi.next();
       RSslogovi.close();
       stat.close();

    if (signal == false)
      return 38;

       }   catch(SQLException esql) {
             System.out.println("Nije uspesno pretrazena baza: " + esql);
                return 39;
                          }
   return 37;
  }
```

```
public  int nadjiSlogiVratiGa(OpstiDomenskiObjekat od,OpstiDomenskiObjekat od1)

   { ResultSet RSslogovi;
     ResultSet RSstavkeSlogova;

            String upit,upit1;
            Statement stat,stat1;


                  try {
          stat =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_R
EAD_ONLY);
                     stat1 =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_R
EAD_ONLY);

                     upit ="SELECT *" +
                             " FROM " + od.vratiImeKlase() +
                    " WHERE " + od.vratiUslovZaNadjiSlog();

                     upit1 ="SELECT *" +
                           " FROM " + od1.vratiImeKlase() +
                              " WHERE " + od1.vratiUslovZaNadjiSlogove();

        RSslogovi = stat.executeQuery(upit);
                     RSstavkeSlogova = stat1.executeQuery(upit1);

              if(!od.Napuni(RSslogovi,RSstavkeSlogova))
                return 72;


                     RSslogovi.close();
                     RSstavkeSlogova.close();

              stat.close();
              stat1.close();
                 } catch(Exception e) {
                   System.out.println("Greska kod citanja sloga iz baze" + e);
                                 return 72;
                                }
                     return 71;
     }

  int vratiZadnjiSlog(OpstiDomenskiObjekat od,OpstiDomenskiObjekat od1)
    {   int signal=0;
```

```
        String upit;
        ResultSet RSslogovi;

            try {
                    stat = con.createStatement();
                    upit ="SELECT Max(" + od.vratiAtributPretrazivanja()+ " )AS Max" +
                         " FROM " + od.vratiImeKlase();

                    RSslogovi = stat.executeQuery(upit);

              if (!od.Napuni(RSslogovi))
                 { System.out.println("KONTROLA 1 NAP");
                 return 76;}

              if (nadjiSlogiVratiGa(od,od1)!=71)
                 return 76;

                     RSslogovi.close();
                     stat.close();

                    } catch(Exception e) {
                     System.out.println("Greska kod citanja zadnjeg unetog sloga u bazu" +
e);
                                  return 76;
                                        }
                     return 75;

            }

  int vratiBrojZadnjegSloga(OpstiDomenskiObjekat od)
      {   int signal=0;
              String upit;
        ResultSet RSslogovi;

            try {
                    stat = con.createStatement();
                    upit ="SELECT Max(" + od.vratiAtributPretrazivanja()+ " )AS Max" +
                         " FROM " + od.vratiImeKlase();

                    RSslogovi = stat.executeQuery(upit);

              if(!od.Napuni(RSslogovi))
                 { return 173;}

                     RSslogovi.close();
                     stat.close();
```

```
                } catch(Exception e) {
                    System.out.println("Greska kod citanja zadnjeg unetog sloga u bazu" +
e);
                                return 173;
                                    }
                    return 171;

            }

  int kreirajSlog(OpstiDomenskiObjekat od)
     { String upit;

        try{
              stat = con.createStatement();
              upit ="INSERT INTO " + od.vratiImeKlase()  +
                " VALUES (" + od.vratiVrednostiAtributa() + ")";

           stat.executeUpdate(upit);
           stat.close();
          }  catch(SQLException esql) {
            System.out.println("Nije uspesno kreiran slog u bazi: " + esql);
                        return 46;
                                    }
     return 45;
           }

}
```

Ispisi.java

```
class Ispisi extends OpstaSO
  {

public static int Ispisi(Klijent cl)
   {
            Ispisi is = new Ispisi();
            return is.opsteIzvrsenjeSO(cl);
   }

int izvrsenjeSO(Klijent cl)
    {  Klijent cl1 = new Klijent(cl);

      if (!nadjiKlijentaiVratiGa(cl1)) return 0;
```

```
      if (!Preduslov(cl1))     return 0;
      if (!ispisiKlijenta(cl)) return 0;
      azurirajKlijent(cl1,cl);  return 0;
            }

boolean stanjeOperacije(int signal)
             {
                    switch (signal)
                      {

                        case 31: Prikazi(signal,"Uspesno zapamcen slog");return true;
                        case 32: Prikazi(signal,"Bezuspesno zapamcen slog");return false;
            case 35: Prikazi(signal,"Uspesno promenjen slog");return true;
                        case 36: Prikazi(signal,"Bezuspesno promenjen slog");return false;
                        case 71: Prikazi(signal,"Uspesno osvezavanje sloga kod rollback-
a");return true;
                        case 72: Prikazi(signal,"Neuspesno osvezavanje sloga kod rollback-
a");return false;
            case 77: Prikazi(signal,"Preduslov je zadovoljen");return true;
            case 93: Prikazi(signal,"Ne moze da radi sa klijentom koji je ispisan");return
false;
                                case 157: Prikazi(signal,"Uspesno izreklamiran klijent ");return
true;
                        case 158: Prikazi(signal,"Neuspesno izreklamiran klijent");return false;
                       }
               return false;
             }

 private boolean Preduslov(Klijent cl1)
     {
                     if (cl1.Ispisan == true)
                      {
                        signal = 93;
                        return false;
                }
          signal = 77;
                return true;

            }

 private boolean ispisiKlijenta(Klijent cl)
             { try{
                    cl.Ispisan = true;} catch(Exception e) { signal = 158; return false;}
               signal = 157;
               return true;
             }
```

```
private boolean nadjiKlijentaiVratiGa(Klijent cl)
  { signal = DB.nadjiSlogiVratiGa(cl,new Usluge(cl));
    if (!stanjeOperacije(signal)) return false;
    return true;
  }

private boolean azurirajKlijent (Klijent cl1,Klijent cl)
     {
                  if (cl1.KlijentId.equals("NEMA"))
        return zapamtiKlijenta(cl);
       else
        return promeniKlijenta(cl);
     }

private boolean zapamtiKlijenta(Klijent cl)
  {
         signal = DB.pamtiSlog(cl);
         if (!stanjeOperacije(signal))
           {
                  return false;
           }
         return true;
  }

private boolean promeniKlijenta(Klijent cl)
  {
          signal = DB.promeniSlog(cl);
          if (!stanjeOperacije(signal))
           { return false;}
          return true;
  }

}
```

Izvrsi.java

```
class Izvrsi extends OpstaSO
{


 public static int Izvrsi(Klijent cl)
  {
           Izvrsi iz = new Izvrsi();
           return iz.opsteIzvrsenjeSO(cl);
```

```
  }

int izvrsenjeSO(Klijent cl)
  { signal= DB.vratiZadnjiSlog(cl,new Usluge(cl));
         if (!stanjeOperacije(signal)) return 0;
   return 1;
  }

boolean stanjeOperacije(int signal)
            {
                  switch (signal)
                    { case 75: Prikazi(signal,"Uspesno procitan zadnji slog iz baze, ako
postoji");return true;
                      case 76: Prikazi(signal,"Neuspesno procitan zadnji slog iz baze");return
false;
                    }
              return false;
            }

}
```

KontrolerKI.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.table.*;


class KontrolerKI
  {
  static public int pritisakTipke(KeyEvent evt,JFormattedTextField
KlijentId,JFormattedTextField ImeIPrezime,JFormattedTextField
Adresa,JFormattedTextField Ukupno,JCheckBox Obradjen,JCheckBox Ispisan,JTable
TabSlogoviKlijenta,DefaultTableModel DTM)
     {  if (evt.getKeyCode() == KeyEvent.VK_INSERT)
                { DTM.addRow(Usluge.vratiPocetneVrednosti());
                    return 21;}

             if (evt.getKeyCode() == KeyEvent.VK_F1)
                    { int selRed = TabSlogoviKlijenta.getSelectedRow();
                          if ( selRed>=0)
                            { DTM.removeRow(selRed);
```

```
                                    return
SORacunaj(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,
TabSlogoviKlijenta,DTM);
                                }
                               else
                                return 22;
                        }

                    return 23;
    }

  static public int pustanjeTipke(KeyEvent evt,JFormattedTextField
KlijentId,JFormattedTextField ImeIPrezime,JFormattedTextField
Adresa,JFormattedTextField Ukupno,JCheckBox Obradjen,JCheckBox Ispisan,JTable
TabSlogoviKlijenta,DefaultTableModel DTM)
    {
                if (evt.getKeyCode() == KeyEvent.VK_ENTER)
                    {
                    return
SORacunaj(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,
DTM);

                }
      return 23;
    }

   static public int SOPretrazi(JFormattedTextField Pretrazivanje,JFormattedTextField
KlijentId,JFormattedTextField ImeIPrezime,JFormattedTextField
Adresa,JFormattedTextField Ukupno,JCheckBox Obradjen,JCheckBox Ispisan,JTable
TabSlogoviKlijenta,DefaultTableModel DTM)
   {   Klijent cl = new Klijent();

KonvertujGrafickeKomponenteUObjekatKlijent(cl,Pretrazivanje,ImeIPrezime,Adresa,Uk
upno,Obradjen,Ispisan,TabSlogoviKlijenta);
                                    int signal = KontrolerPL.Pretrazi(cl);

        KonvertujObjekatKlijentUGrafickeKomponente(cl,KlijentId,ImeIPrezime,Adresa
,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

                return signal;
         }

        static public int SOIzvrsi(JFormattedTextField KlijentId,JFormattedTextField
ImeIPrezime,JFormattedTextField Adresa,JFormattedTextField Ukupno,JCheckBox
Obradjen,JCheckBox Ispisan,JTable TabSlogoviKlijenta,DefaultTableModel DTM)
   {   Klijent cl = new Klijent();
```

```
KonvertujGrafickeKomponenteUObjekatKlijent(cl,KlijentId,ImeIPrezime,Adresa,Ukupn
o,Obradjen,Ispisan,TabSlogoviKlijenta);
                                        int signal = KontrolerPL.Izvrsi(cl);

        KonvertujObjekatKlijentUGrafickeKomponente(cl,KlijentId,ImeIPrezime,Adresa
,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

                return signal;
        }

   static public int SOUbaciNovog(JFormattedTextField KlijentId,JFormattedTextField
ImeIPrezime,JFormattedTextField Adresa,JFormattedTextField Ukupno,JCheckBox
Obradjen,JCheckBox Ispisan,JTable TabSlogoviKlijenta,DefaultTableModel DTM)
   {
                Klijent cl = new Klijent();
                                int signal = KontrolerPL.UbaciNovog(cl);

        KonvertujObjekatKlijentUGrafickeKomponente(cl,KlijentId,ImeIPrezime,Adresa
,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

                return signal;

        }

   static public int SORacunaj(JFormattedTextField KlijentId,JFormattedTextField
ImeIPrezime,JFormattedTextField Adresa,JFormattedTextField Ukupno,JCheckBox
Obradjen,JCheckBox Ispisan,JTable TabSlogoviKlijenta,DefaultTableModel DTM)
   {
                Klijent cl = new Klijent();

KonvertujGrafickeKomponenteUObjekatKlijent(cl,KlijentId,ImeIPrezime,Adresa,Ukupn
o,Obradjen,Ispisan,TabSlogoviKlijenta);
                                        int signal = KontrolerPL.Racunaj(cl);

        KonvertujObjekatKlijentUGrafickeKomponente(cl,KlijentId,ImeIPrezime,Adresa
,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

                return signal;
        }

   static public int SOIspisi(JFormattedTextField KlijentId,JFormattedTextField
ImeIPrezime,JFormattedTextField Adresa,JFormattedTextField Ukupno,JCheckBox
Obradjen,JCheckBox Ispisan,JTable TabSlogoviKlijenta,DefaultTableModel DTM)
   {  Klijent cl= new Klijent();
```

```
        KonvertujGrafickeKomponenteUObjekatKlijent(cl,KlijentId,ImeIPrezime,Adresa
,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta);
                                        int signal =  KontrolerPL.Ispisi(cl);

        KonvertujObjekatKlijentUGrafickeKomponente(cl,KlijentId,ImeIPrezime,Adresa
,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

                return signal;
        }

   static public int SOObradi(JFormattedTextField KlijentId,JFormattedTextField
ImeIPrezime,JFormattedTextField Adresa,JFormattedTextField Ukupno,JCheckBox
Obradjen,JCheckBox Ispisan,JTable TabSlogoviKlijenta,DefaultTableModel DTM)
   {  Klijent cl = new Klijent();

        KonvertujGrafickeKomponenteUObjekatKlijent(cl,KlijentId,ImeIPrezime,Adresa
,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta);
                                                int signal = KontrolerPL.Obradi(cl);

        KonvertujObjekatKlijentUGrafickeKomponente(cl,KlijentId,ImeIPrezime,Adresa
,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

                return signal;
        }

  static void KonvertujGrafickeKomponenteUObjekatKlijent(Klijent
cl,JFormattedTextField KlijentId,JFormattedTextField ImeIPrezime,JFormattedTextField
Adresa,JFormattedTextField Ukupno,JCheckBox Obradjen,JCheckBox Ispisan,JTable
TabSlogoviKlijenta)
  {
     cl.Napuni(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan);
     cl.oklijent = new Usluge[TabSlogoviKlijenta.getRowCount()];
      for(int i = 0; i< TabSlogoviKlijenta.getRowCount(); i++)
        { cl.oklijent[i] = new Usluge(cl);
         for (int j=0; j<TabSlogoviKlijenta.getColumnCount(); j++)
          {  Object ob = TabSlogoviKlijenta.getValueAt(i,j);
                        cl.oklijent[i].Napuni(j,ob);
                }
            }
  }

static void KonvertujObjekatKlijentUGrafickeKomponente(Klijent
cl,JFormattedTextField KlijentId,JFormattedTextField ImeIPrezime,JFormattedTextField
Adresa,JFormattedTextField Ukupno,JCheckBox Obradjen,JCheckBox Ispisan,JTable
TabSlogoviKlijenta,DefaultTableModel DTM)
```

```
 {
  cl.Vrati(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan);
  DTM.setRowCount(cl.oklijent.length);
  for(int i = 0; i< cl.oklijent.length; i++)
   { for (int j=0; j<TabSlogoviKlijenta.getColumnCount(); j++)
     {  System.out.println("Vrednost tabele: " + cl.oklijent[i].Vrati(j));
                    TabSlogoviKlijenta.setValueAt(cl.oklijent[i].Vrati(j),i,j);}
   }

 }


}
```

KontrolerPL.java

```
class KontrolerPL
{
   public static int Izvrsi(Klijent cl)
     { return Izvrsi.Izvrsi(cl); }

   public static int UbaciNovog(Klijent cl)
     { return UbaciNovog.UbaciNovog(cl);}

   public static int Pretrazi(Klijent cl)
     { return Pretrazi.Pretrazi(cl);}

   public static int Racunaj(Klijent cl)
     { return Racunaj.Racunaj(cl);}

   public static int Obradi(Klijent cl)
     { return Obradi.Obradi(cl);}


   public static int Ispisi(Klijent cl)
     { return Ispisi.Ispisi(cl);}

}
```

Obradi.java

```
class Obradi extends OpstaSO
  {
public static int Obradi(Klijent cl)
```

```
  {       Obradi ob = new Obradi();
          return ob.opsteIzvrsenjeSO(cl);
  }

int izvrsenjeSO(Klijent cl)
  { Klijent cl1 = new Klijent(cl);

          if (!nadjiKlijentaiVratiGa(cl1)) return 0;
          if (!Preduslov(cl1)) return 0;
          if (!racunajUkupno(cl)) return 0;
          if (!obradiKlijenta(cl)) return 0;
       if (!azurirajKlijent(cl1,cl)) return 0;
          if (!brisiStavkeKlijenta(cl1)) return 0;
          dodavanjeStavkiKlijenta(cl);return 0;
  }

   boolean stanjeOperacije(int signal)
            {
                    switch (signal)
                      {
              case 31: Prikazi(signal,"Uspesno zapamcen slog");return true;
                    case 32: Prikazi(signal,"Bezuspesno zapamcen slog");return false;
       case 33: Prikazi(signal,"Slogovi uspesno obrisani");return true;
                    case 34: Prikazi(signal,"Slogovi bezuspesno obrisani");return false;
       case 35: Prikazi(signal,"Uspesno promenjen slog");return true;
                    case 36: Prikazi(signal,"Bezuspesno promenjen slog");return false;
              case 57: Prikazi(signal,"Uspesno obradjen klijent");return true;
                    case 58: Prikazi(signal,"Neuspesno obradjen klijent");return false;
                    case 71: Prikazi(signal,"Uspesno pretrazivanje slogova");return true;
                    case 72: Prikazi(signal,"Neuspesno pretrazivanje slogova");return false;
       case 77: Prikazi(signal,"Preduslov je zadovoljen");return true;
       case 78: Prikazi(signal,"Neuspesno izracunata ukupna vrednost");return false;
       case 79: Prikazi(signal,"Uspesno izracunata ukupna vrednost");return true;
       case 94: Prikazi(signal,"Ne moze da radi sa klijentom koji je obradjen ili
ispisan");return false;
                      }
              return false;
            }
private boolean nadjiKlijentaiVratiGa(Klijent cl)
  { signal = DB.nadjiSlogiVratiGa(cl,new Usluge(cl));
    if (!stanjeOperacije(signal)) return false;
    return true;
  }
private boolean Preduslov(Klijent cl1)
     {
                    if ((cl1.Obradjen == true) || (cl1.Ispisan == true))
```

```
                    {
                     signal = 94;stanjeOperacije(signal);
                     return false;
                }
        signal = 77; stanjeOperacije(signal);
              return true;

            }

private boolean racunajUkupno(Klijent cl)
              { double Suma =0;
               try {
                        for(int i=0;i<cl.oklijent.length;i++)
                          { cl.oklijent[i].CenaUsluge = new Double
(cl.oklijent[i].CenaUsluge.doubleValue() * cl.oklijent[i].BrojKomada.intValue());
                            Suma = Suma +  cl.oklijent[i].CenaUsluge.doubleValue();
                           }
                    cl.Ukupno = new Double(Suma);
                        } catch(Exception e) { signal = 78; stanjeOperacije(signal);return
false;}
          signal = 79;stanjeOperacije(signal);
                return true;
              }
private boolean obradiKlijenta(Klijent cl)
             { try{
                    cl.Obradjen = true;} catch(Exception e) { signal = 58;
stanjeOperacije(signal);return false;}
               signal = 57;stanjeOperacije(signal);
               return true;
             }
private boolean azurirajKlijent (Klijent cl1,Klijent cl)
      {
                    if (cl1.KlijentId.equals("NEMA"))
          return zapamtiKlijenta(cl);
         else
          return promeniKlijenta(cl);
      }

private boolean zapamtiKlijenta(Klijent cl)
   {
           signal = DB.pamtiSlog(cl);
           if (!stanjeOperacije(signal))
             {
                    return false;
             }
           return true;
```

```
    }

private boolean promeniKlijenta(Klijent cl)
    {
            signal = DB.promeniSlog(cl);
            if (!stanjeOperacije(signal))
              {
                    return false;
              }
            return true;
    }
private boolean brisiStavkeKlijenta(Klijent cl)
             {
                        signal = DB.brisiSlogove(new Usluge(cl));
                if (!stanjeOperacije(signal))
                {
                        return false;
                }
               return true;
                   }
private boolean dodavanjeStavkiKlijenta(Klijent cl)
                  {

                        for(int i=0;i<cl.oklijent.length;i++)
                { signal = DB.pamtiSlog(cl.oklijent[i]);
                                if (!stanjeOperacije(signal))
                                  {
                                        return false;
                    }
                  }
                return true;
      }
 }
```

OpstaSO.java

```
class OpstaSO
{
 static  DatabaseBroker DB;
 static int signal;
 static boolean BazaOtvorena = false;

 int opsteIzvrsenjeSO(Klijent cl)
  {
```

```
        signal = 0;
        if (!otvoriBazu()) return signal;
        izvrsenjeSO(cl);
        proveraUspesnostiTransakcije(cl);
        zatvoriBazu();
        return signal;
  }


int izvrsenjeSO(Klijent cl){
            return 0;}

boolean otvoriBazu()
  {
   if (BazaOtvorena == false)
     { DB = new DatabaseBroker();
      signal = DB.otvoriBazu("Klijent");
      if (!stanjeOperacijeOpstaSO(signal)) return false;
      }
    BazaOtvorena = true;
    return true;
  }

boolean zatvoriBazu()
    {  int signal1 = DB.zatvoriBazu();
          if (!stanjeOperacijeOpstaSO(signal1))
            { signal = signal1;return false;}
          BazaOtvorena = false;
          return true;
    }

boolean proveraUspesnostiTransakcije(Klijent cl)
   { int signal1;
     if (stanjeOperacije(signal))
       { signal1 = DB.commitTransakcije();
        if (!stanjeOperacijeOpstaSO(signal1))
         {signal=signal1;
         return false;
         }
                }
          else
       { signal1 = DB.rollbackTransakcije();
        if (!stanjeOperacijeOpstaSO(signal1))
          { signal = signal1;
           return false;
               }
```

```
        }
            return true;
    }

 boolean stanjeOperacije(int signal){
                            return false;}

 boolean stanjeOperacijeOpstaSO(int signal)
   {
          switch (signal)
            {
                    case 41: Prikazi(signal,"Uspesno otvorena baza"); return true;
                    case 42: Prikazi(signal,"Neuspesno otvorena baza - drajver");return false;
                    case 43: Prikazi(signal,"Neuspesno otvorena baza - konekcija");return
false;
                    case 44: Prikazi(signal,"Neuspesno otvorena baza - zastita");return false;

        case 51: Prikazi(signal,"Commit uspesno obavljen");return true;
              case 52: Prikazi(signal,"Commit neuspesno obavljen");return false;


                    case 53: Prikazi(signal,"Rollback uspesno obavljen");return true;
                    case 54: Prikazi(signal,"Rollback neuspesno obavljen");return false;


                    case 61: Prikazi(signal,"Baza uspesno zatvorena");return true;
                    case 62: Prikazi(signal,"Baza neuspesno zatvorena");return false;

      }
    return false;
   }

 void Prikazi(int signal,String stanje)
   { System.out.println("SIGNAL: " + signal + ". Stanje: " + stanje);}

}
```

UbaciNovog.java

```
class UbaciNovog extends OpstaSO
{
public static int UbaciNovog(Klijent cl)
   {       UbaciNovog kn = new UbaciNovog();
           return kn.opsteIzvrsenjeSO(cl);
```

```
}
int izvrsenjeSO(Klijent cl)
  {Klijent clPom = new Klijent();

   signal= DB.vratiBrojZadnjegSloga(clPom);
   if (!stanjeOperacije(signal)) return 0;

   signal = clPom.povecajKlijentId();
   if (!stanjeOperacije(signal)) return 0;

   signal = cl.dodeliKlijentId(clPom);
   if (!stanjeOperacije(signal)) return 0;

   signal = DB.kreirajSlog(cl);
   if (!stanjeOperacije(signal)) return 0;

   signal = DB.kreirajSlog(cl.oklijent[0]);
   if (!stanjeOperacije(signal)) return 0;

   return 1;
}

boolean stanjeOperacije(int signal)
            {
           switch (signal)
                     {
               case 45:Prikazi(signal,"Uspesno kreiran novi slog"); return true;
          case 46:Prikazi(signal,"Neuspesno kreiran novi slog");return false;

               case 47:Prikazi(signal,"Uspesno povecan klijent id"); return true;
               case 48:Prikazi(signal,"Neuspesno povecan klijent id"); return false;

               case 55:Prikazi(signal, "Uspesno dodeljen klijent id");return true;
               case 56:Prikazi(signal, "Neuspesno dodeljen klijent id");return false;

          case 171:Prikazi(signal,"Uspesno vracen broj zadnjeg sloga"); return true;
               case 173:Prikazi(signal,"Neuspesno vracen broj zadnjeg sloga");return false;

                     }
             return false;
            }
}
```

Racunaj.java

```
class Racunaj extends OpstaSO
{
public static int Racunaj(Klijent cl)
  {      Racunaj rc = new Racunaj();
         return rc.opsteIzvrsenjeSO(cl);
  }

int izvrsenjeSO(Klijent cl)
 { Klijent cl1 = new Klijent(cl);
         if (!nadjiKlijentaiVratiGa(cl1)) return 0;
         if (!Preduslov(cl1)) return 0;
         if (!racunajUkupno(cl)) return 0;
         if (!azurirajKlijenta(cl1,cl)) return 0;
         if (!brisiStavkeKlijent(cl1)) return 0;
         dodavanjeUsluge(cl);  return 0;
 }

boolean stanjeOperacije(int signal)
    {
       switch (signal)
          { case 31: Prikazi(signal,"Uspesno zapamcen slog");return true;
            case 32: Prikazi(signal,"Bezuspesno zapamcen slog");return false;
     case 33: Prikazi(signal,"Slogovi uspesno obrisani");return true;
            case 34: Prikazi(signal,"Slogovi bezuspesno obrisani");return false;
     case 35: Prikazi(signal,"Uspesno promenjen slog");return true;
            case 36: Prikazi(signal,"Bezuspesno promenjen slog");return false;
            case 71: Prikazi(signal,"Uspesno pretrazivanje slogova");return true;
            case 72: Prikazi(signal,"Neuspesno pretrazivanje slogova");return false;
     case 77: Prikazi(signal,"Preduslov je zadovoljen");return true;
     case 78: Prikazi(signal,"Bezuspesno izracunata ukupna vrednost");return false;
     case 79: Prikazi(signal,"Uspesno izracunata ukupna vrednost");return true;
     case 94: Prikazi(signal,"Ne moze da radi sa racunom koji je obradjen ili
storniran");return false;
                    }
             return false;
           }
boolean nadjiKlijentaiVratiGa(Klijent cl)
  { signal = DB.nadjiSlogiVratiGa(cl,new Usluge(cl));
    if (!stanjeOperacije(signal)) return false;
    return true;
  }
 private boolean Preduslov(Klijent cl1)
    {
```

```
                    if ((cl1.Obradjen == true) || (cl1.Ispisan == true))
                      {
                       signal = 94;
                       return false;
                }
        signal = 77;
              return true;
            }
 private boolean racunajUkupno(Klijent cl)
              { double Suma =0;
               try {
                        for(int i=0;i<cl.oklijent.length;i++)
                          { cl.oklijent[i].Vrednost = new Double
(cl.oklijent[i].CenaUsluge.doubleValue() * cl.oklijent[i].BrojKomada.intValue());
                            Suma = Suma +  cl.oklijent[i].Vrednost.doubleValue();
                           }
                    cl.Ukupno = new Double(Suma);
                         } catch(Exception e) { signal = 78; return false;}
          signal = 79;
                return true;
              }
private boolean azurirajKlijenta(Klijent cl1,Klijent cl)
      {
                    if (cl1.KlijentId.equals("NEMA"))
          return zapamtiKlijenta(cl);
         else
          return promeniKlijenta(cl);
      }
private boolean zapamtiKlijenta(Klijent cl)
   { signal = DB.pamtiSlog(cl);
           if (!stanjeOperacije(signal)) return false;
           return true;
   }

private boolean promeniKlijenta(Klijent cl)
   {  signal = DB.promeniSlog(cl);
            if (!stanjeOperacije(signal)) return false;
            return true;
   }

private boolean brisiStavkeKlijent(Klijent cl)
               { signal = DB.brisiSlogove(new Usluge(cl));
                 if (!stanjeOperacije(signal)) return false;
                 return true;
                    }
```

```
private boolean dodavanjeUsluge(Klijent cl)
                    {  for(int i=0;i<cl.oklijent.length;i++)
                    { signal = DB.pamtiSlog(cl.oklijent[i]);
                                if (!stanjeOperacije(signal)) return false;
                    }
                  return true;
      }
}
```

Pretrayi.java

```
 class Pretrazi extends OpstaSO
 {
public static int Pretrazi(Klijent cl)
   {          Pretrazi pr = new Pretrazi();
              return pr.opsteIzvrsenjeSO(cl);
   }

int izvrsenjeSO(Klijent cl)
    { signal = DB.nadjiSlogiVratiGa(cl,new Usluge(cl));
           if (!stanjeOperacije(signal)) return 0;
      return 1;
          }

 boolean stanjeOperacije(int signal)
             {
                    switch (signal)
                      { case 71: Prikazi(signal,"Uspesno je procitan slog iz baze (uspesno
pretrazivanje)");return true;
                        case 72: Prikazi(signal,"Neuspesno je procitan slog iz baze (neuspesno
pretrazivanje)");return false;
                      }
              return false;
             }
 }
```

OpstiDomenskiObjekti.java

```
import java.sql.*;


interface OpstiDomenskiObjekat
{
String vratiVrednostiAtributa();
String postaviVrednostiAtributa();
String vratiImeKlase();
String vratiUslovZaNadjiSlog();
String vratiUslovZaNadjiSlogove();
boolean Napuni(ResultSet RSslog, ResultSet RSstavkeSlogova);
String vratiAtributPretrazivanja();
boolean Napuni(ResultSet RSslog);

}
```

EkranskaForma.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.table.*;
import java.lang.reflect.*;


 class EkranskaForma extends JFrame {

public static void main(String args[])
 {
          EkranskaForma EF = new EkranskaForma();
       EF.show();
 }

public EkranskaForma ()
  { KreirajKomponenteEkranskeForme();
    PokreniMenadzeraRasporedaKomponeti();
    PostaviImeDokumenta();


    PostaviPoljeKlijentId();
    PostaviPoljeImeIPrezime();
    PostaviPoljeAdresa();
```

```
    PostaviPoljeUkupno();
    PostaviKBObradjen();
    PostaviKBIspisan();
    PostaviTabelu();

    PostaviLabeluKlijentId();
    PostaviLabeluImeIPrezime();
    PostaviLabeluAdresa();
    PostaviLabeluUkupno();
    PostaviDugmeUbaci();
    PostaviDugmeObradi();
    PostaviDugmeIspisi();
    PostaviDugmeRacunaj();

    Izvrsavanje();

    PostaviPoljePretrazivanje();

    pack();
    }

void KreirajKomponenteEkranskeForme()
  {
        LNazivDokumenta = new JLabel();
        KlijentId = new JFormattedTextField();
        ImeIPrezime = new JFormattedTextField();
    Adresa = new JFormattedTextField();
        Ukupno = new JFormattedTextField();
    Obradjen = new JCheckBox();
    Ispisan = new JCheckBox();
    TabSlogoviKlijenta = new JTable();

        LKlijentId = new JLabel();
        LImeIPrezime = new JLabel();
    LAdresa = new JLabel();
        LUkupno = new JLabel();

    Ubaci = new JButton();
        Racunaj = new JButton();
        Obradi = new JButton();
    Ispisi= new JButton();

    Pretrazivanje = new JFormattedTextField();
 }

void PokreniMenadzeraRasporedaKomponeti()
```

```
  { getContentPane().setLayout(new AbsoluteLayout());}

void PostaviImeDokumenta()
{  LNazivDokumenta.setFont(new Font("Times New Roman", 1, 24));
   LNazivDokumenta.setText("HEMIJSKO");
   getContentPane().add(LNazivDokumenta, new AbsoluteConstraints(20, 10, -1, -1));
}

    private JFormattedTextField KlijentId;
    private JFormattedTextField ImeIPrezime; ;
    private JFormattedTextField Adresa;
    private JFormattedTextField Ukupno;
    private JCheckBox Obradjen;
    private JCheckBox Ispisan;


void PostaviPoljeKlijentId()
 {
   KlijentId.setValue(new String("nema"));
   KlijentId.setEditable(false);
 getContentPane().add(KlijentId, new AbsoluteConstraints(20, 100, 50, -1));
}

void PostaviPoljeImeIPrezime()
 {
          ImeIPrezime.setValue(new String("nema"));
    getContentPane().add(ImeIPrezime, new AbsoluteConstraints(20, 150, 250, -1));
 }

void PostaviPoljeAdresa()
 {
          Adresa.setValue(new String("nema"));
          getContentPane().add(Adresa, new AbsoluteConstraints(20, 200, 250, -1));
 }

void PostaviPoljeUkupno()
  {
          Ukupno.setValue(new Double(0));
  Ukupno.setBackground(new Color(51, 51, 255));
    Ukupno.setForeground(new Color(255, 255, 51));
    Ukupno.setFont(new Font("Dialog", 3, 14));
          Ukupno.setEditable(false);
          getContentPane().add(Ukupno, new AbsoluteConstraints(480, 310, 120, -1));
  }

 void PostaviKBObradjen()
```

```
 {
  Obradjen.setText("Obradjen");
  Obradjen.setFocusable(false);
  Obradjen.setEnabled(false);
  getContentPane().add(Obradjen, new AbsoluteConstraints(390, 80, -1, -1));
}

void  PostaviKBIspisan()
  {
         Ispisan.setText("Ispisan");
         Ispisan.setFocusable(false);
   Ispisan.setEnabled(false);
  getContentPane().add(Ispisan, new AbsoluteConstraints(500, 80, -1, -1));
  }


   JTable TabSlogoviKlijenta;

   DefaultTableModel DTM;
   JComboBox NazivUsluge;

void PostaviTabelu()
{
 DTM =  new DefaultTableModel (Usluge.ZagSlogoviKlijenta(),1)
      {
          Class[] types = Usluge.vratiTipove();
          public Class getColumnClass(int columnIndex){return types [columnIndex];}
      };

TabSlogoviKlijenta.setModel(DTM);

TabSlogoviKlijenta.addKeyListener(new KeyAdapter()
  {
           public void keyPressed(KeyEvent evt)
         {int signal = KontrolerKI.pritisakTipke

(evt,KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

          PrikazPorukeUspesnosti(signal);
         }
    });

TabSlogoviKlijenta.addKeyListener(new KeyAdapter()
  {
           public void keyReleased(KeyEvent evt)
         {int signal = KontrolerKI.pustanjeTipke
```

```
(evt,KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

            PrikazPorukeUspesnosti(signal);
           }
      });

int vsb = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int hsb = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane skrol = new JScrollPane(TabSlogoviKlijenta,vsb,hsb);
getContentPane().add(skrol, new AbsoluteConstraints(20, 250, 440,150));

}

    private JLabel LNazivDokumenta;
    private JLabel LKlijentId;
    private JLabel LImeIPrezime;
    private JLabel LAdresa;
    private JLabel LUkupno;

void PostaviLabeluKlijentId()
{ LKlijentId.setText("KlijentId");
  getContentPane().add(LKlijentId, new AbsoluteConstraints(20, 80, -1, -1));
}

void  PostaviLabeluImeIPrezime()
 { LImeIPrezime.setText("ImeIPrezime");
   getContentPane().add(LImeIPrezime, new AbsoluteConstraints(20,130, -1, -1));
 }

void  PostaviLabeluAdresa()
 { LAdresa.setText("Adresa");
   getContentPane().add(LAdresa, new AbsoluteConstraints(20, 180, -1, -1));
 }

void  PostaviLabeluUkupno()
{ LUkupno.setText("Ukupno");
  getContentPane().add(LUkupno, new AbsoluteConstraints(480, 290, -1, -1));
}

    private JButton Ubaci;
    private JButton Racunaj;
    private JButton Obradi;
    private JButton Ispisi;
```

```
void PostaviDugmeUbaci()
 { Ubaci.setText("Ubaci");
   Ubaci.addActionListener(new ActionListener() {

   public void actionPerformed(ActionEvent evt)
         { int signal = KontrolerKI.SOUbaciNovog

(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

              PrikazPorukeUspesnosti(signal);
          }
      });

  getContentPane().add(Ubaci, new AbsoluteConstraints(170, 30, -1, -1));
}

void PostaviDugmeRacunaj()
 { Racunaj.setText("Racunaj");
   Racunaj.addActionListener(new ActionListener() {

   public void actionPerformed(ActionEvent evt)
         { int signal = KontrolerKI.SORacunaj

(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

           PrikazPorukeUspesnosti(signal);
          }
      });

  getContentPane().add(Racunaj, new AbsoluteConstraints(280, 30, -1, -1));
}

void PostaviDugmeObradi()
 {
   Obradi.setText("Obradi");
   Obradi.addActionListener(new ActionListener() {

     public void actionPerformed(ActionEvent evt)
           { int signal = KontrolerKI.SOObradi

(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

                PrikazPorukeUspesnosti(signal);
          }
         });
   getContentPane().add(Obradi, new AbsoluteConstraints(390, 30, -1, -1));
```

```
 }
void  PostaviDugmeIspisi()
{Ispisi.setText("Ispisi");
  Ispisi.addActionListener(new ActionListener() {

          public void actionPerformed(ActionEvent evt)
          { int signal = KontrolerKI.SOIspisi

(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogoviKlijenta,DTM);

            PrikazPorukeUspesnosti(signal);
           }
       });

  getContentPane().add(Ispisi, new AbsoluteConstraints(500, 30, -1, -1));
}


void Izvrsavanje()
{
          int signal =
KontrolerKI.SOIzvrsi(KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogo
viKlijenta,DTM);
     PrikazPorukeUspesnosti(signal);
}


private JFormattedTextField Pretrazivanje;

void PostaviPoljePretrazivanje()
 {
   Pretrazivanje.setValue(new String(""));
   Pretrazivanje.setBackground(new Color(51, 51, 255));
   Pretrazivanje.setForeground(new Color(255, 255, 51));
   Pretrazivanje.addActionListener
     (new ActionListener()
       {
              public void actionPerformed(ActionEvent evt)
        { int signal = KontrolerKI.SOPretrazi

(Pretrazivanje,KlijentId,ImeIPrezime,Adresa,Ukupno,Obradjen,Ispisan,TabSlogoviKlijen
ta,DTM);

          PrikazPorukeUspesnosti(signal);
        }
             }
```

```
            );

    getContentPane().add(Pretrazivanje, new AbsoluteConstraints(100, 100, 50, -1));
}



void PrikazPorukeUspesnosti(int signal)
{ Boolean prikazi = new Boolean(true);;
  DijalogBoksPrikazPoruke DBPP = new DijalogBoksPrikazPoruke(this);
  int signal1 = DBPP.PrikazPoruke(signal);
  if (signal1 == 1)
    { DBPP.show();
      pack();
    }
}

}

class DijalogBoksPrikazPoruke extends JDialog
{ public DijalogBoksPrikazPoruke(JFrame roditelj)
   { super(roditelj,"-Prikaz poruka o izvrsenim operacijama-", true);}

  int PrikazPoruke(int signal)
    { String poruka;
            Box b = Box.createVerticalBox();
      b.add(Box.createGlue());

     System.out.println("Signal na dijalogu:" + signal);
          switch (signal)
            {

                    case 21: poruka = new String("Unet novi red u kolonu");break;
                    case 22: poruka = new String("Red ne moze da se brise ako nije
selektovan");break;
                    case 23: poruka = new String("Pritisnuta je tipka koja se ne
obradjuje");return 0;
                    case 31: poruka = new String("Slog je uspesno zapamcen");break;
                    case 32: poruka = new String("Neuspesno pamcenje sloga u bazi");break;
                    case 33: poruka = new String("Slog ili slogovi uspesno obrisani u
bazi");break;
                    case 34: poruka = new String("Neuspesno brisanje sloga u bazi");break;
                    case 35: poruka = new String("Slog je uspesno promenjen");break;
                    case 36: poruka = new String("Neuspesna promena sloga u bazi");break;
                    case 37: poruka = new String("Nasao slog u bazi");break;
                    case 38: poruka = new String("Nema sloga u bazi");break;
```

```
                case 39: poruka = new String("Neuspesno pretrazivanje sloga u
bazi.");break;
                case 41: poruka = new String("Baza je uspesno otvorena");return 0;
                case 42: poruka = new String("Baza je neuspesno otvorena
(drajver).");break;
                case 43: poruka = new String("Baza je neuspesno otvorena
(konekcija).");break;
                case 44: poruka = new String("Baza je neuspesno otvorena
(zastita).");break;
                case 45: poruka = new String("Slog je uspesno kreiran i zapamcen u
bazi.");break;
                case 46: poruka = new String("Slog nije uspesno kreiran i zapamcen u
bazi.");break;
                case 47: poruka = new String("Uspesno povecan klijent id.");return 0;
            case 48: poruka = new String("Neuspesno povecan klijent id");break;

                case 51: poruka = new String("Uspesno je odradjen commit
transakcije.");break;
                case 52: poruka = new String("Neuspesno je odradjen commit
transakcije.");break;
                case 53: poruka = new String("Uspesno je odradjen rollback
transakcije.");break;
                case 54: poruka = new String("Neuspesno je odradjen rollback
transakcije.");break;
                case 55: poruka = new String("Uspesno dodeljen klijent id.");return 0;
                case 56: poruka = new String("Neuspesno dodeljen klijent id.");break;

                case 57: poruka = new String("Uspesno obradjen klijent.");break;
                case 58: poruka = new String("Neuspesno obradjen klijent.");break;


                case 61: poruka = new String("Uspesno zatvorena baza.");return 0;
                case 62: poruka = new String("Neuspesno zatvorena baza.");break;

                case 71: poruka = new String("Uspesno pretrazivanje slogova");return 0;
                case 72: poruka = new String("Neuspesno pretrazivanje slogova");break;

                case 75: poruka = new String("Uspesno procitan zadnji slog iz baze,
ukoliko postoji");return 0;
                case 76: poruka = new String("Neuspesno procitan zadnji slog iz
baze");return 0;


                case 77: poruka = new String("Zadovoljen preduslov SO");return 0;
```

```
                    case 78: poruka = new String("Nije dobro izracunata ukupna
cena");break;

                    case 79: poruka = new String("Uspesno izracunata ukupna cena");return
0;

                   case 81: poruka = new String("Uspesno povezivanje sa bazom i citanje
zadnjeg racuna ukoliko postoji.");break;
                    case 82: poruka = new String("Greska kod citanja zadnjeg
clanaaze");break;

                    case 94: poruka = new String("Pokusaj da se radi sa obradjenim ili
ispisanim clanom");break;
                    case 92: poruka = new String("Ne moze se obraditi klijent koji je
obradjen ili ispisan");break;
                    case 93: poruka = new String("Klijent je vec ispisan");break;


                    case 157: poruka = new String("Uspesno ispisan klijent");break;
                    case 158: poruka = new String("Neuspesno ispisan klijent");break;

                    case 171: poruka = new String("Uspesno vracen broj zadnjeg
sloga");break;
                    case 173: poruka = new String("Neuspesno vracen broj zadnjeg
sloga");break;



                    default: poruka = null;return 0;
            }
     b.add(new JLabel(poruka));
     getContentPane().add(b,"Center");
     setSize(450,70);
     return 1;
    }

}
```

Literatura:

Skripta za predmet: Pojektovanje programa, dr.Siniša Vlajić