Univerzitet u Beogradu

Fakultet organizacionih nauka

**Projektni rad iz predmeta Računarske mreže i telekomunikacije**

Mentor:

Bojan Jovanović

Studenti:

Anđela Babić 47/14

Marko Antić 42/14

Dane Nikolić 44/14

Beograd, 2017.

Sadržaj

[Verbalni opis 3](#_Toc491652132)

[Analiza modela aplikacije 4](#_Toc491652133)

[1. Uspostavljanje komunikacije 4](#_Toc491652134)

[2. Igranje 4](#_Toc491652135)

[3. Obaveštavanje o ishodu igre 4](#_Toc491652136)

[Dijagram konteksta 5](#_Toc491652137)

[Prvi nivo dekompozicije 6](#_Toc491652138)

[Drugi nivo dekompozicije 7](#_Toc491652139)

[1. Uspostavljanje komunikacije 7](#_Toc491652140)

[2. Igranje 8](#_Toc491652141)

[3. Obaveštavanje o ishodu igre 9](#_Toc491652142)

[Rečnik podataka 10](#_Toc491652143)

[PMOV 12](#_Toc491652144)

[Specifičnosti implementacije 13](#_Toc491652145)

[Povezivanje na bazu i rad sa podacima 13](#_Toc491652146)

[Igraci Table Model 15](#_Toc491652147)

[Timer 17](#_Toc491652148)

[Deserijalizacija 18](#_Toc491652149)

[Odvijanje komunikacije između klijenta i servera na konkretnom primeru 19](#_Toc491652150)

# Verbalni opis

Multiplejer aplikacija ***Upoznaj Evropu*** omogućava klijentima da se međusobno takmiče u znanju o Evropi. Zasniva se na igri *Ko zna zna* iz popularnog kviza *Slagalica*.

Tok igre:

Osoba koja pristupa aplikaciji unosi korisničko ime. Ukoliko postoji neki drugi korisnik sa tim korisničkim imenom, osobi se ispisuje poruka da je ime zauzeto i da pokuša ponovo. Nakon što se korisnik uspešno ulogovao, ispisuje mu se lista online korisnika. Ukoliko trenutno nema online korisnika, igrač može da osvežava prozor sve dok se ne pojavi neki korisnik, a ako ima online korisnika onda može da pošalje zahtev za igru bilo kome od njih. Dok je korisnik online, u svakom trenutku može da mu stigne zahtev za igru od strane drugog korisnika, nakon čega je on u mogućnosti da prihvati ili odbije taj zahtev. Ako korisnik odbije zahtev, igrač koji je poslao zahtev dobija poruku o tome. U suprotnom počinje igra. Igračima koji su započeli igru više se ne može slati zahtev.

Igra funkcioniše tako što oba igrača istovremeno dobijaju po jedno isto pitanje i imaju 10 sekundi da odgovore na njega.

Prilikom odgovaranja na pitanja mogući su sledeći slučajevi:

1. Ukoliko su oba igrača odgovorila na pitanje pre isteka vremena, vreme se zaustavlja a igračima se prikazuje tačan odgovor (obojen u zeleno) i odgovor protivnika. Igrač koji je tačno odgovorio osvaja 10 poena, dok se za netačan odgovor dobija -5 poena.
2. Ukoliko jedan igrač nije odgovorio na pitanje do isteka vremena, igrač ne dobija negativne poene. Drugi igrač mora da sačeka da vreme za odgovor istekne, nakon čega se obojici prikazuje tačan odgovor.
3. Ukoliko neko od igrača napusti igru, drugi igrač nastavlja igru regularno, samo što nakon svakog pitanja mora da sačeka da prođe vreme.

U svakom trenutku na ekranu se prikazuje ostvareni broj poena kao i broj poena protivnika. Igra se sastoji od 10 pitanja koja se odnose na znanje o Evropi.

Nakon 10 pitanja igračima se saopštava ko je pobedio i pobednik se upisuje na rang listu koja se prikazuje na ekranu. Ukoliko oba igrača imaju jednak broj poena obojica se upisuju na rang listu.

# Analiza modela aplikacije

Detaljnom analizom prethodno definisanih zahteva koje bi aplikacija Upoznaj Evropu trebala da zadovoljava, zaključili smo da se čitav proces aplikacije može podeliti na 3 podprocesa. To su uspostavljanje komunikacije, igranje i obaveštavanje igrača o ishodu igre.

## Uspostavljanje komunikacije

S obzirom na to da je ovaj kviz mrežna aplikacija, postoje dve strane koje učestvuju u komunikaciji - klijent koji traži uslugu i server koji pruža uslugu. Proces započinje tako što klijent šalje serveru zahtev kako bi se identifikovao, na šta server odgovara prihvatanjem zahteva, a zatim klijent unosi svoje korisničko ime, koje server može da odbije u slučaju da se u skladištu klijenata nalazi klijent sa istim imenom. U tom slučaju, server obaveštava klijenta da je potrebno ponovo uneti korisničko ime. U slučaju potvrde zahteva, server skladišti tog klijenta. Nakon toga, započinje proces povezivanja igrača tako što server šalje klijentu listu dostupnih korisnika koju pročita iz skladišta, a zatim klijent bira protivnika sa kojim će se takmičiti u kvizu. Svoj izbor prosleđuje serveru koji isti šalje do odabranog klijenta. Odabrani klijent šalje serveru povratnu informaciju. U slučaju odbijanja zahteva server obaveštava klijenta da je njegov izabranik odbio zahtev. Pri pozitivnom ishodu, server šalje protivnicima obaveštenje o početku igre.

## Igranje

Nakon što izvršimo povezivanje klijenta i obavestimo ih o početku igre, potrebno je da pošaljemo pitanje klijentima. Taj posao će obavljati server, na čijoj će se strani nalaziti baza pitanja u vidu tekstualne datoteke. Dakle server uzima jedno nasumično izabrano pitanje, i šalje pitanje klijentu. Kada klijentu stigne pitanje, prikazujemo ga, pokrećemo tajmer, kako bi ograničili vreme odgovora na pitanje. Ako odgovori na pitanje, klijent šalje serveru svoj odgovor ili praznu poruku. Server skladišti taj odgovor, kako bi na kraju mogao da izvrši obradu odgovora. Prilikom obrade server upoređuje odgovor klijenta sa tačnim odgovorom za dato pitanje i ažurira rezultat, koji se odnosi na broj poena klijenta u partiji. I na samom kraju server obaveštava klijenta o tačnom odgovoru.

## Obaveštavanje o ishodu igre

Na kraju je potrebno obavestiti igrače o ishodu igre, odnosno o osvojenom broju poena u kvizu. Za prikazivanje rezultata igraču, potrebno je pročitati broj poena iz skladišta Rezultat, nakon čega se klijentu šalje obaveštenje o ostvarenom broju poena. Na osnovu broja poena koje su osvojili učesnici kviza, potrebno je identifikovati pobednika. U slučaju da oba klijenta imaju jednak broj poena, i jedan i drugi se smatraju pobednicima. Pobednik se upisuje u skladište Pobednik. Nakon toga potrebno je upisati pobednika/pobednike na rang listu. Rang lista je baza podataka koja se nalazi na lokalnom računaru. U rang listi se čuvaju podaci o svim pobednicima I njihovim pozicijama. Igračima se potom prikazuje rang lista u vidu tabele, koja čita podatke sa sladišta Rang Lista. Time se igra završava.

Nakon detaljne analize modela aplikacije, može se kostruisati dijagram konteksta, SSA, rečnik podataka I PMOV

# Dijagram konteksta



# Prvi nivo dekompozicije



# Drugi nivo dekompozicije

## Uspostavljanje komunikacije



Skladište *Klijent* je tekstualna datoteka koja čuva podatke o klijentima koji su se povezali na server.

## Igranje



Skladište *Pitanje* je tekstualna datoteka, koja se sastoji od samog pitanja, ponđenih odgovora za to pitanje i tačnog odgovora. Skladište *Odgovor* je tekstualna datoteka gde se čuva istorija odgovora, sastoji se od odgovora klijenta (tekst\_odgovora) i broja poena za taj odgovor (0 ukoliko je odgovor tekst odgovora prazna poruka, 10 ukoliko je u pitanju tačan odgovor i -5 u slučaju netačnog odgovora), jedan odgovor se vezuje za konkretno pitanje i konkretnog klijenta. Skladište Rezultat je tekstualna datoteka u kojoj se pamti broj poena koje su osvojili svi učesnici kviza.

## 3. Obaveštavanje o ishodu igre



Skladište *Klijent* je tekstualna datoteka koja čuva podatke o klijentima koji su se povezali na server. Skladište *Pobednik* je tekstualna datoteka u kojoj se pamte svi pobednici kviza. Skladište *Rang Lista* je baza podataka na lokalnom računaru u kojoj se nalaze svi igrači kviza koji su pobedili, kao i njihov osvojeni broj poena. Skladište *Rezultat* je tekstualna datoteka u kojoj se pamti broj poena koje su osvojili svi učesnici kviza.

# Rečnik podataka

|  |  |  |
| --- | --- | --- |
| Polje | Tip | Ograničenje |
| **Klijent**<ID\_Klijent,Korisnicko\_ime> | | |
| ID\_Klijent | INTEGER | NOT NULL |
| Korisnicko\_ime | Char(30) |  |
| **Pitanje**<Rbr\_pitanja,Tekst\_pitanja,Ponuđeni\_odgovori,Tačan\_odgovor> | | |
| Rbr\_pitanja | INTEGER | NOT NULL |
| Tekst\_pitanja | Char(300) |  |
| Ponuđeni\_odgovori | Char(100) |  |
| Tačan\_odgovor | Char(25) |  |
| **Odgovor**<Tekst\_odgovora, Broj\_poena\_za\_odgovor> | | |
| Tekst\_odgovora | Char(30) |  |
| Broj\_poena\_za\_odgovor | INTEGER |  |
| **Rezultat**<ID\_Rezultat, Broj\_poena, *ID\_Klijent* > | | |
| ID\_Rezultat | INTEGER | NOT NULL |
| Broj\_poena | INTEGER |  |
| ID\_Klijent | INTEGER |  |
| **Pobednik**<ID\_Pobednika, Korisnicko\_ime, *ID\_Rezultat* > | | |
| ID\_Pobednika | INTEGER | NOT NULL |
| Korisnicko\_ime | Char(50) |  |
| *ID\_Rezultat* | INTEGER |  |
| **Rang\_lista**<Pozicija, *ID\_Pobednika*> | | |
| Pozicija | INTEGER | NOT NULL |
| ID\_Pobednika | INTEGER | NOT NULL |

# PMOV



# Specifičnosti implementacije

## Povezivanje na bazu i rad sa podacima

Pristup bazi i učitavanje i manipulisanje podacima u Java programima odvija se u nekoliko koraka:

- učitavanje JDBC drajvera

- uspostavljanje veze sa bazom

- kreiranje naredbe

- izvršavanje SQL upita

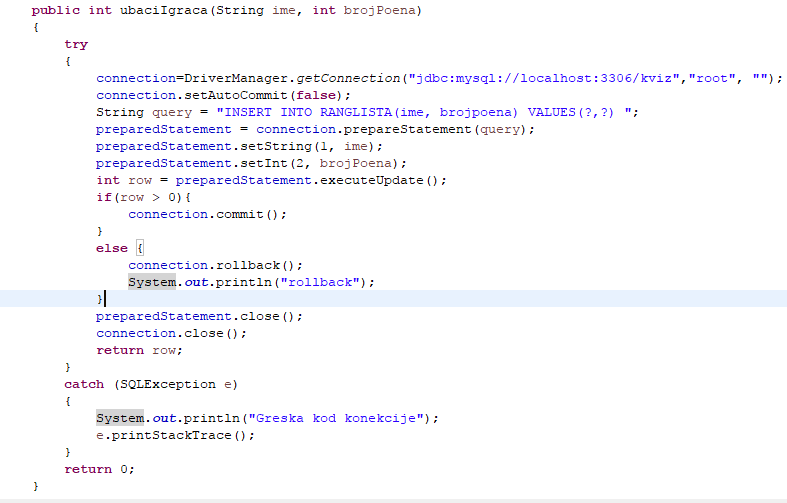
- manipulisanje dobijenim podacima

Za potrebe aplikacije Upoznaj Evropu, potrebno je napraviti bazu podataka u koju će se smeštati podaci o igračima pobednicima. Takodje prilikom prikazivanja rang liste, potrebno je čitati podatke iz baze. Dakle, ono što je potrebno implementirati je:

1. Ubacivanje igrača u bazu podataka

2. Učitavanje igrača iz baze radi prikazivanja rang liste

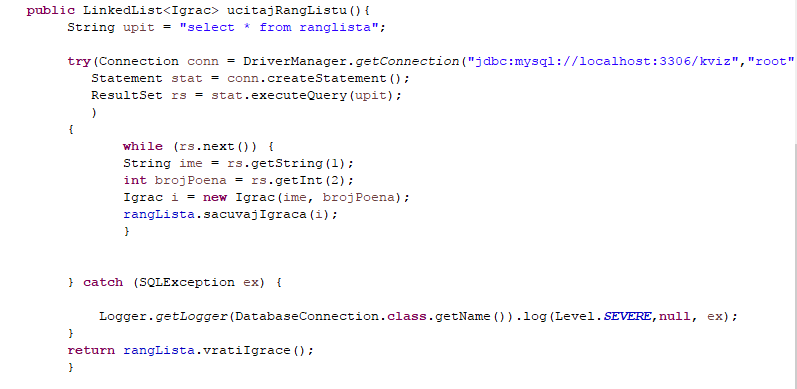
Iz tog razloga, za implementaciju su korišćene dve metode, čije je kratko objašnjenje dato u nastavku.



Slika 1 – metoda za ubacivanje pobednika u bazu

Nakon uspostavljanja veze sa bazom - *DriverManager.getConnection*, potrebno je kreirati naredbu u SQL-u koja će ubaciti igrača u bazu. Za to je korišćena klasa PreparedStatement koja omogućuje prosleđivanje parametara već pripremljenom SQL upitu. Parametri koje prosledjujemo upitu su upravo parametri naše funkcije ubaciIgrača - ime i broj poena.

Za izvršavanje upita korišćena je funkcija executeUpdate(), koja kao rezultat vraća broj redova koji je ažuriran. Ukoliko je rezultat veći od 0 (row>0), promenu je potrebno potvrditi u bazi naredbom connection.commit();.



Slika 2 – učitavanje igrača iz baze

Za razliku od prethodne metode, za učitavanje igrača iz baze korišćena je klasa Statement, koja se koristi za izvršenje statičnog upita i vraćanje rezultata. Igrači se čitaju iz baze jedan po jedan. Nakon svakog učitavanja igrača pozivom funkcije sacuvajIgraca, igrač će se ubaciti u listu. Nakon toga se ResultSet kursor pomera na narednog igrača naredbom rs.next(). Nakon što je kursor stigao do kraja i svi igrači iz baze su prebačeni u našu listu, lista se može prikazati igračima preko tabele. Implementaciju tabele možete naći u poglavlju -.

## Igraci Table Model

Za potrebe prikazivanja konačne rang liste implementirali smo klasu **IgraciTableModel** kako bismo napravili tabelu u kojoj će se prikazivati najbolji rezultati korisnika. Ova klasa nasleđuje apstrakltnu klasu AbstractTableModel koja se nalazi u paketu javax.swing.table. ATM je klasa koja implementira interfejs TableModel, tačnije daje implementaciju za većinu metoda ovog interfejsa. Prilikom kreiranja našeg modela – IgraciTableModel bilo je neophodno implementirati sledeće metode:

public int getRowCount();

public int getColumnCount();

public Object getValueAt(int row, int column);

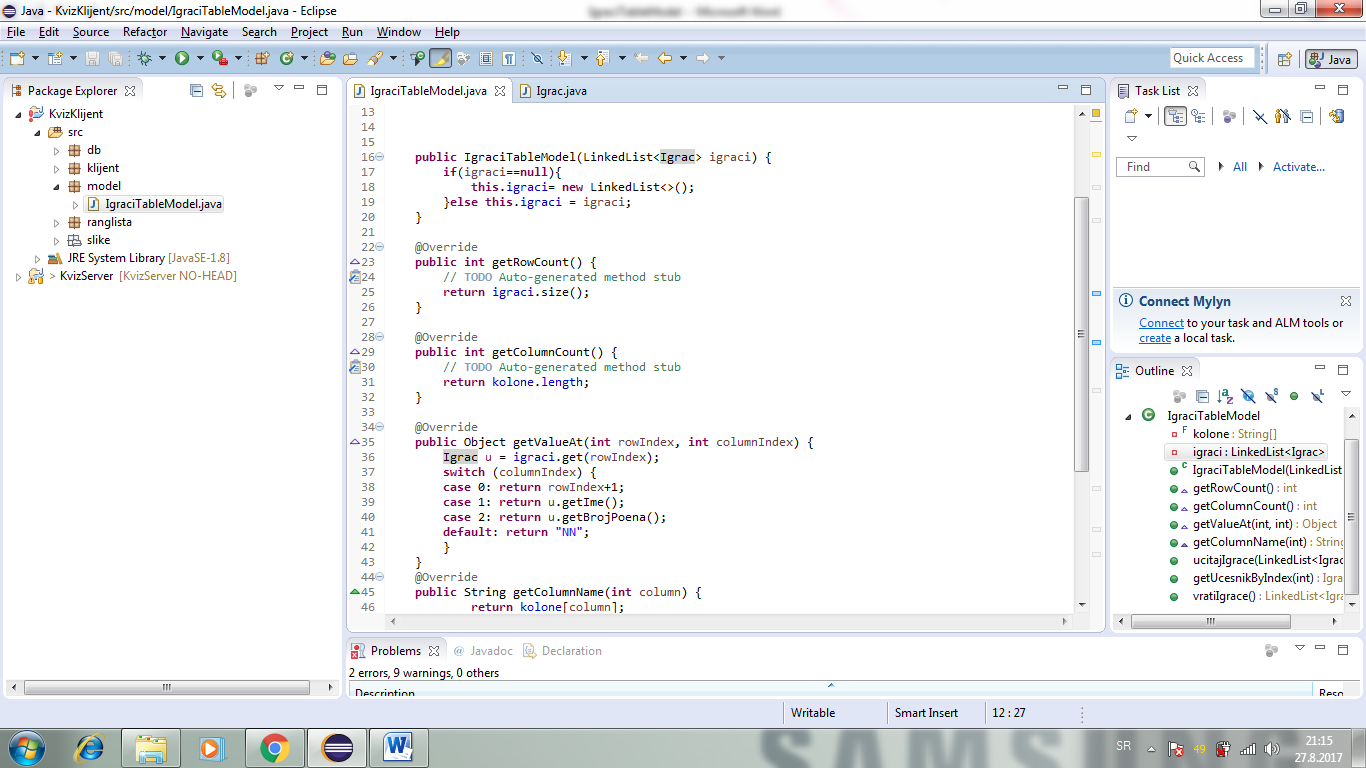
public String getColumnName(int column);

public void ucitajIgrace(LinkedList<Igrac> ucesnici);

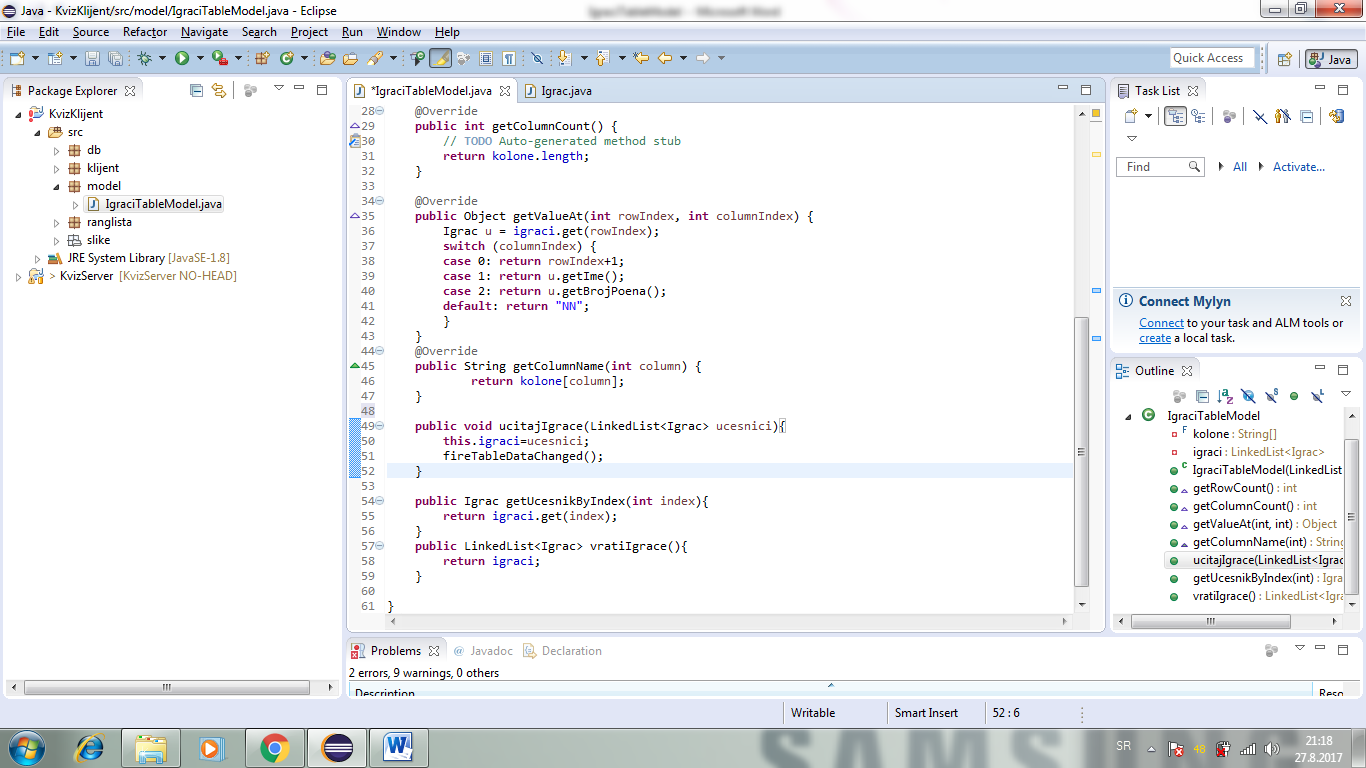
public Igrac getUcesnikByIndex(int index);

public LinkedList<Igrac> vratiIgrace();.

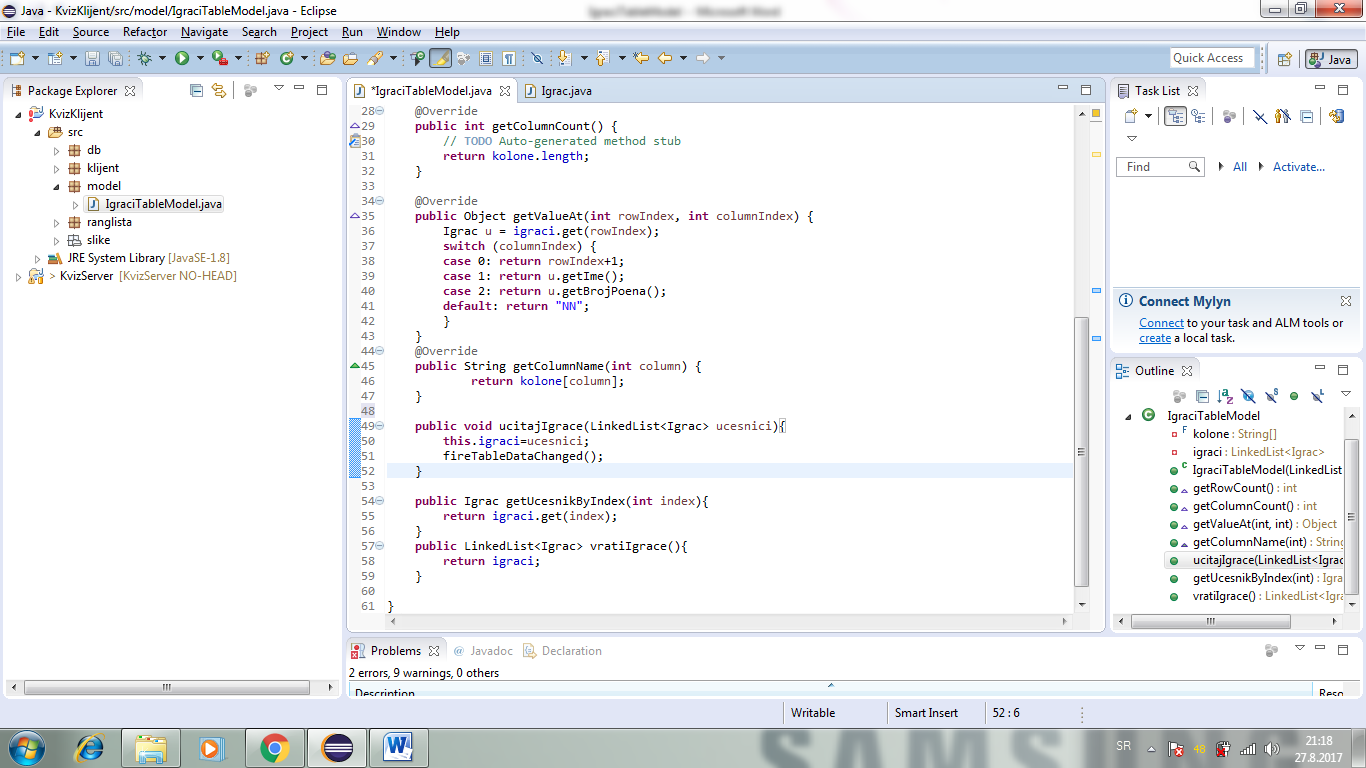
Metoda *getRowCount*() vraća broj redova ove tabele. Ova tabela ima onoliko redova koliko ima igrača u listi. Metoda *getColumnCount*() vraća broj kolona tabele, a to su atributi svakog igrača : Rang, Ime i Poeni. Metoda *getValueAt*() vraća konkretnu vrednost iz ćelije tabele čiji su red i kolona prosleđeni kao parametri ove metode.



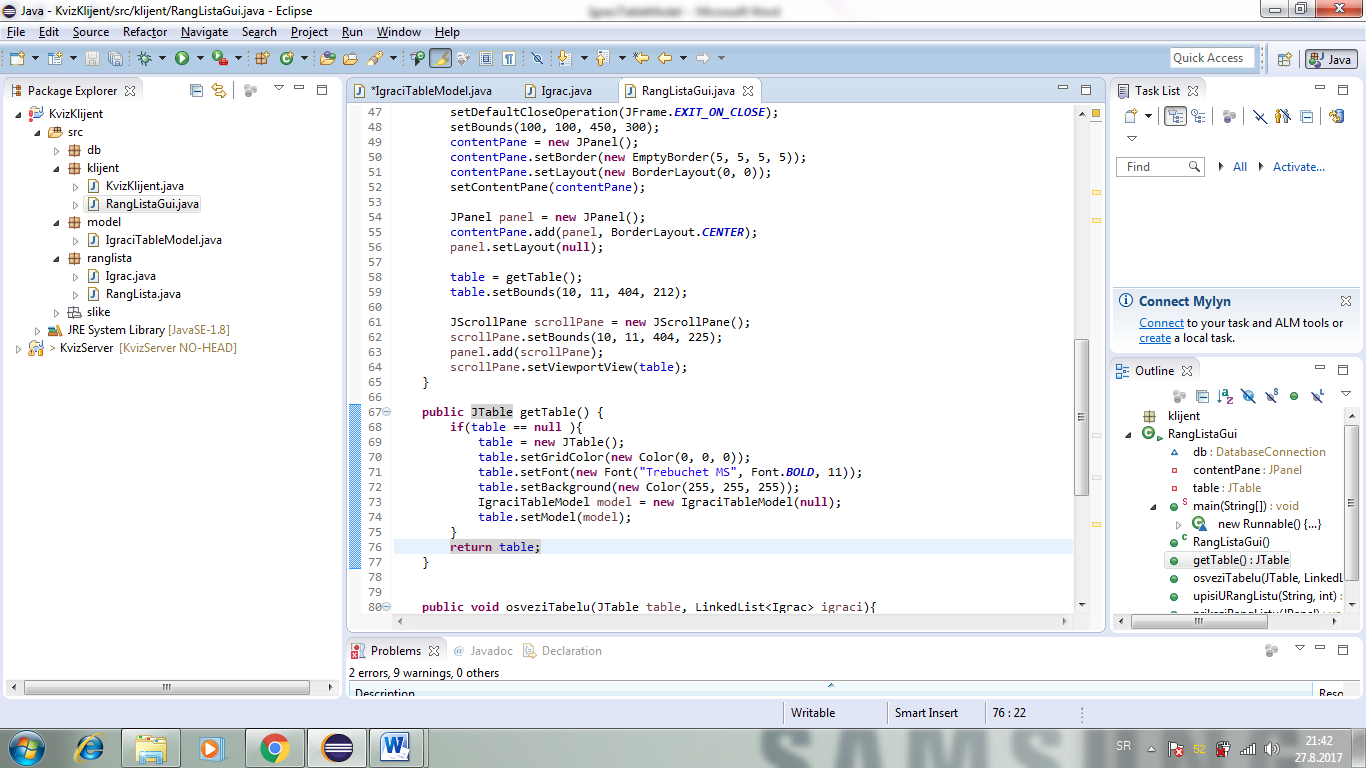
Metoda *fireTableDataChanged()* menja vrednsoti ćelija u redovima tabele kao i broj redova u tabeli, dok se struktura same table ne menja. U metodi *ucitajIgrace(LinkedList<Igrac> ucesnici)* se ažurira tabela pomoću ove metode nakon što se proslede igrači koji treba da se upišu na listu.



Metoda *getColumnName()* vraća naziv kolone čiji se broj prosleđuje kao parametar.



Prilikom pravljenja tabele u GUI-u pravimo novu instancu klase IgraciTableModel i modelu tabele koju smo prethodno napravili prosleđujemo ovaj model.



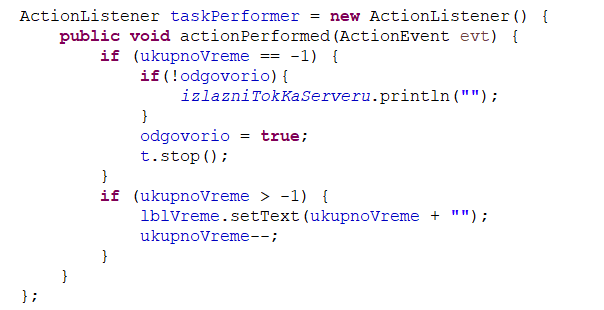
## Timer

Kao što smo već napomenuli u analizi modela, za projektovanje igrice ovog tipa bio nam je neophodan timer. U tu svrhu smo koristili swing timer. Svaki put kada od servera stigne naredno pitanje, pokreće se timer na sledeći način:

Description: C:\Users\Andjela Babic\Desktop\6 semestar\Baze\Vezbe\jovana\Capture1.PNG

gde je t objekat tipa Timer, koji incijalizujemo, a potom i statujemo. Prvi parameter (1000) se odnosi na kašnjenje i označava brzinu u milisekundama, dok je drugi parameter (taskPerformer) određuje osluškivača događaja, koji će da se pokretati svaki put nakon isteka predviđenog vremena.

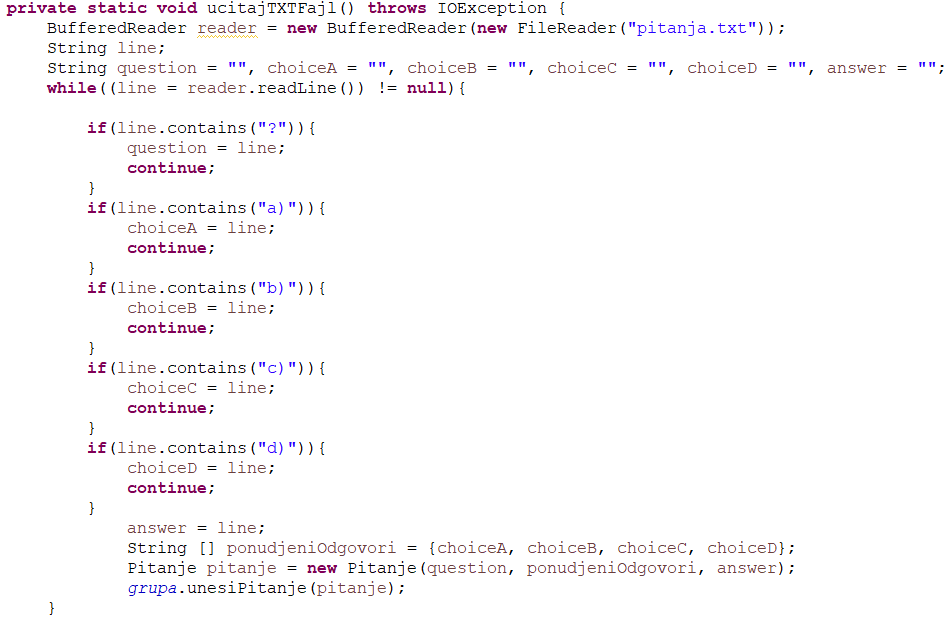
Linijom koda *t.start()* “startujemo” tajmer, kada je tajmer pokrenut, sačekaće prvo da prođe jedna sekunda, pre nego što okine prvi događaj (ActionEvent), nakon toga će nastaviti da okida događaj svaki put kada istekne definisano vreme (u našem slučaju 1 sekund), sve dok tajmer nije stopiran.



Promenljiva *ukupnoVreme* je tipa int, koja na početku, a i nakon svakog pitanja, ima vrednost 10. Kada pokrenemo tajmer *ukupnoVreme* će biti veće -1, tako da je zadovoljen drugi uslov, ispisujemo ga na ekranu i umanjujemo za 1, ovo se ponavlja svake sekunde, sve dok *ukupnoVreme* ne dođe do nule. Tada je vreme da stopiramo tajmer, jer je isteklo predviđeno vreme za odgovor. Pošto server čeka da dobije neki odgovor od klijenta, ukoliko nakon 10 sekundi, serveru nije stigao nikakav odgovor (vrednost promeljive odgovorio, koja je tipa boolean je true, ukoliko je korisnik odgovorio na pitanje), šaljemo praznu poruku. I stopiramo tajmer.

## Deserijalizacija

Pitanja koja smo koristili u kvizu se nalaze u tekstualnoj datoteci “pitanja.txt”, kako bismo mogli da radimo sa njima neophodno je da izvršimo njihovo učitavanje. Klasa BufferedReader obezbeđuje privremenu memoriju koja povećava efikasnost učitavanja, njoj se pri inicijalizaciji kao tip ulaznog toka proseđuje FileReader koji je namenjen za čitanje toka karaktera.

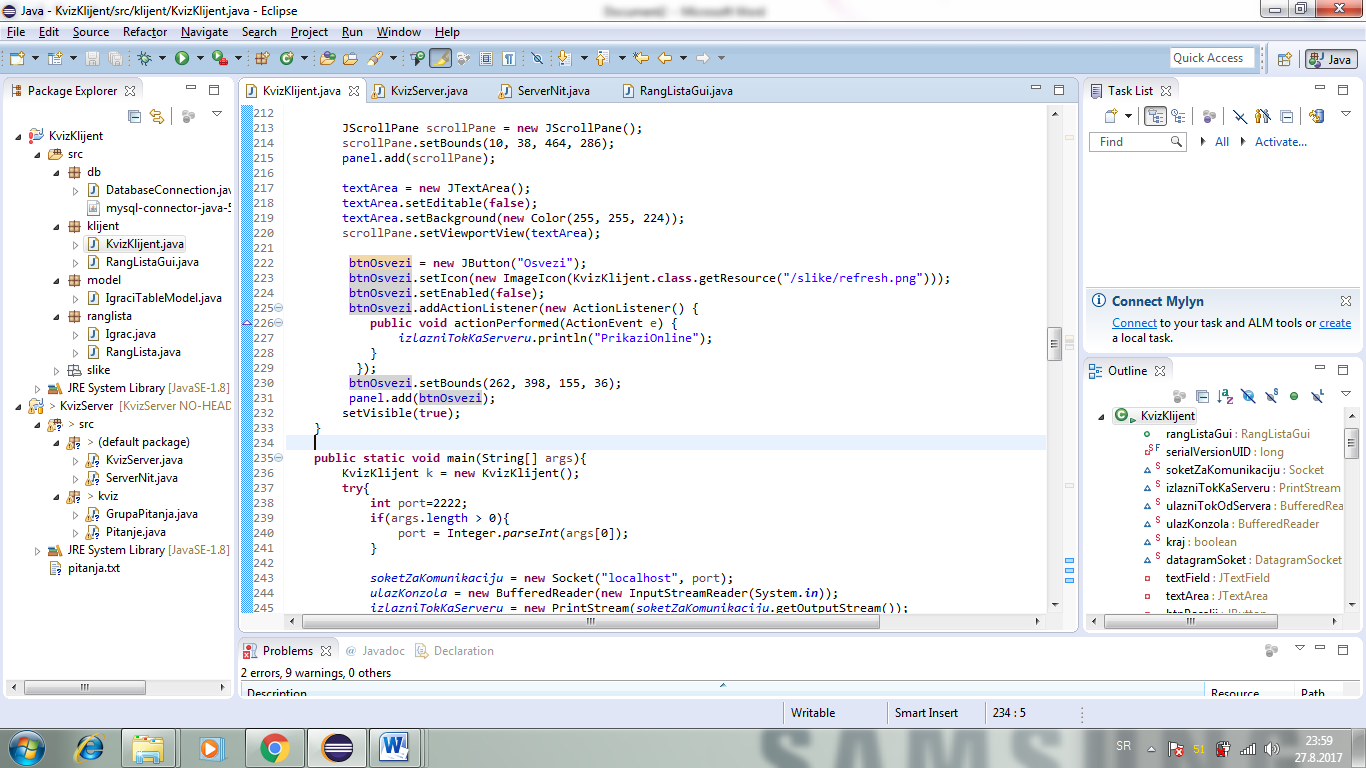


Tekstualna datoteka sa pitanjema je struktuirana tako da se u jednom redu nalazi pitanje, potom ponuđeni odgovori (po jedno u svakom narednom redu) i na kraju tačan odgovor.

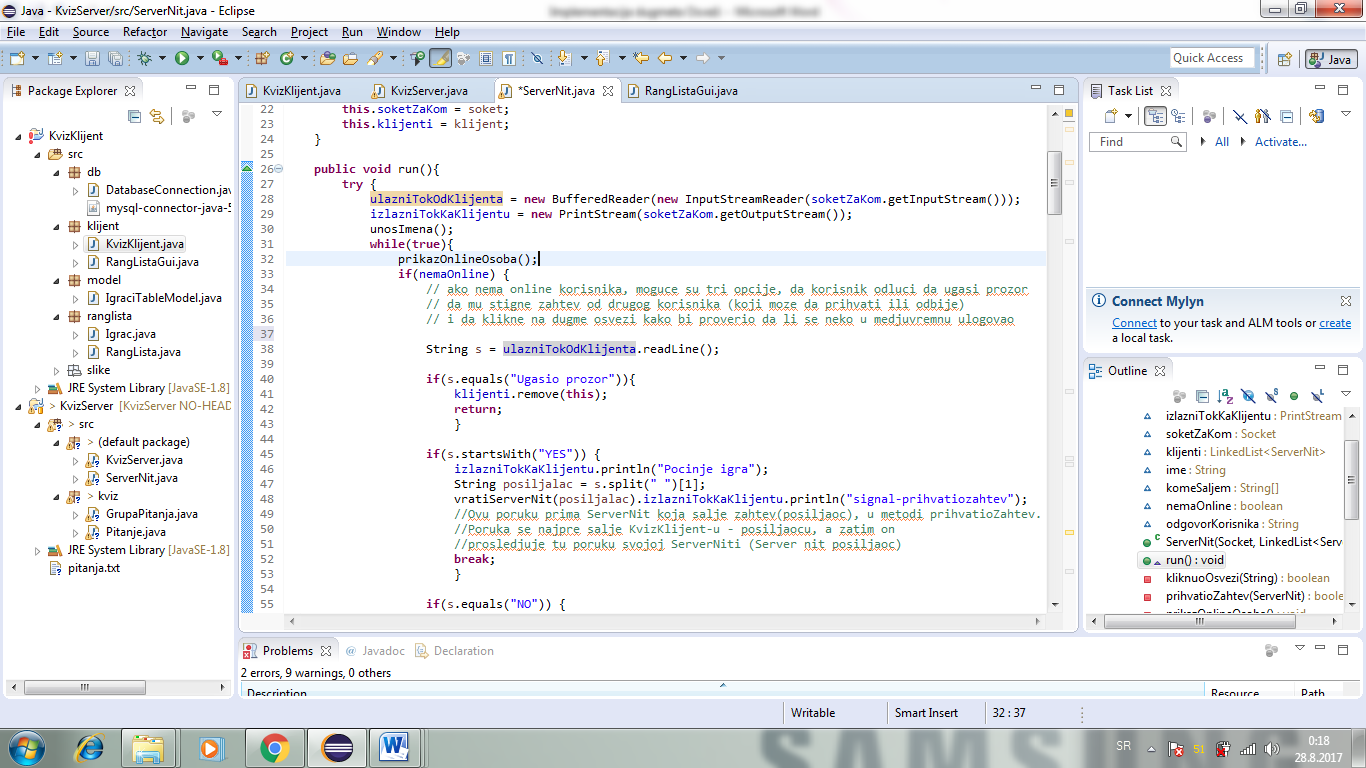
Kada pozovemo *reader.readLine()* čitamo jedan red tekstualne datoteke. Kada memorišemo pitanje i ponuđene odgovore, pravimo objekat tipa pitanje, koje potom ubacujemo u listu. Ukoliko se dođe do kraja datoteke, metoda vraća null i tada završavamo čitanje.

## Odvijanje komunikacije između klijenta i servera na konkretnom primeru

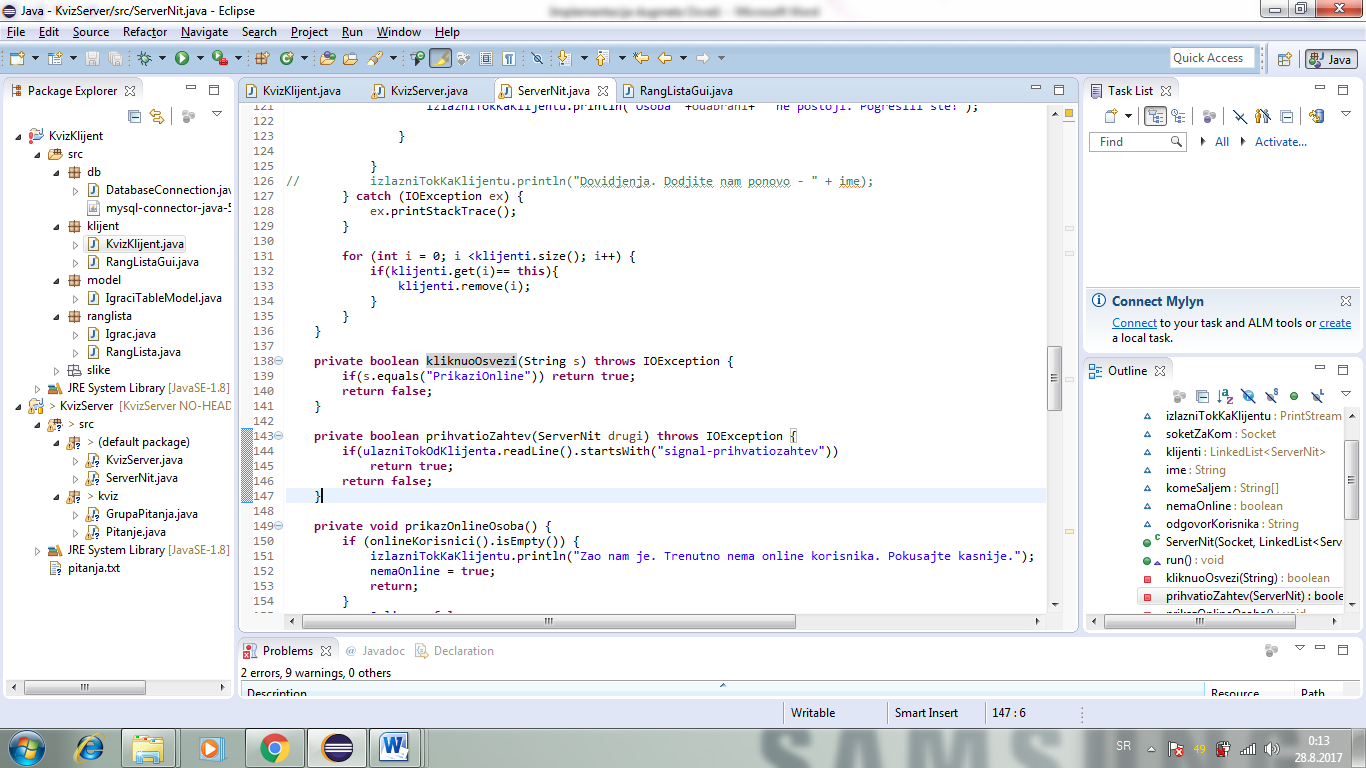
Button *Osveži* služi da bi klijent mogao da proveri da li se neki novi korisnik prijavio. Ovo dugme je u GUI-u implementirano preko klase JButton. Klikom na dugme Osveži pokreće se metoda actionPerformed i njome se šalje serveru poruka da klijent želi da vidi listu dostupnih korisnika. Ova poruka se šalje preko toka PrintStream koji služi da serveru ispiše datu poruku.



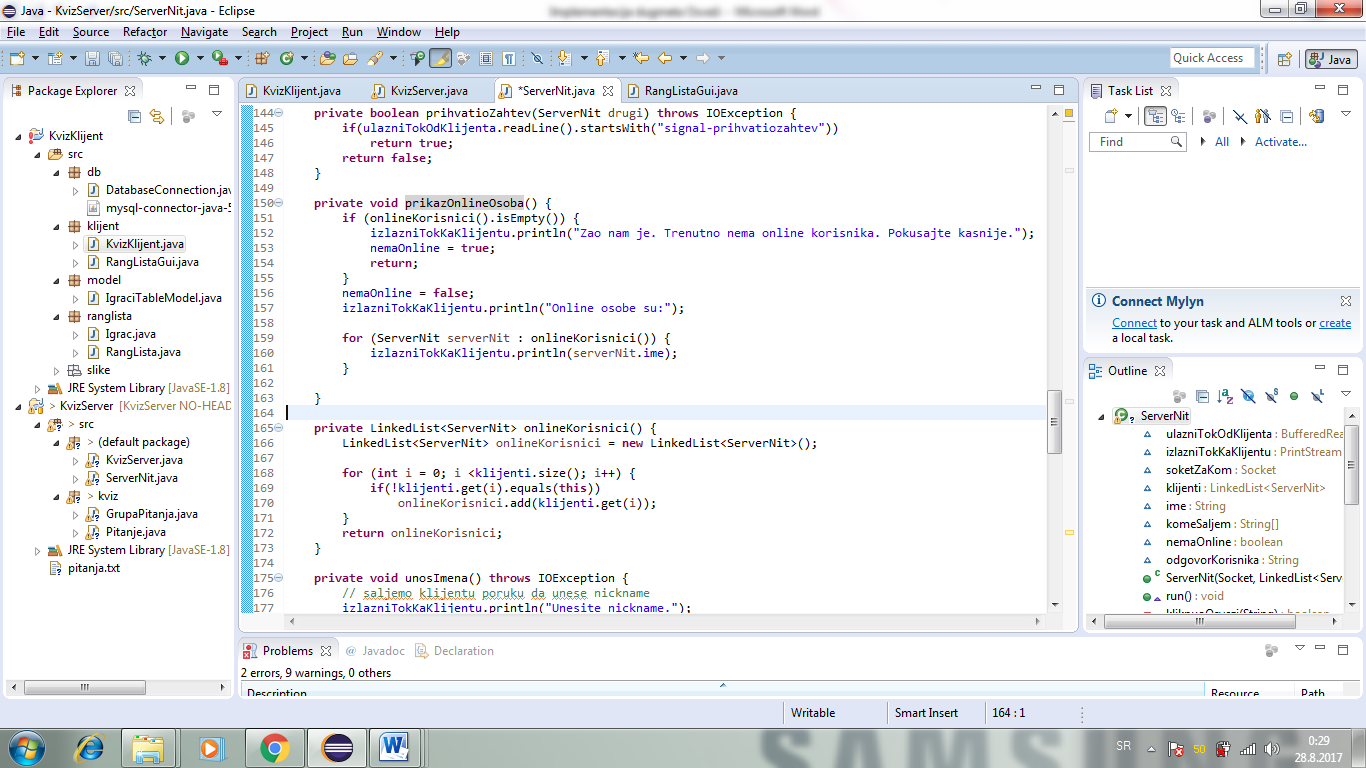
Server konstantno čeka poruke od klijenta (u while(true) petlji) koje pakuje u promenljivu *s* koja je tipa String. *UlazniTokOdKlijenta* je implementiran preko klase BufferedReader koja se koristi u slučajevima kada podatke iz toka treba prikupiti u celinu pre dalje obrade.



S obzirom da server od klijenta prima različite poruke, na serverskoj strani implementirana je metoda *kliknuoOsvezi(s)* koja vraća true u slučaju da je korisnik, klikom na dugme *Osveži*, poslao signal serveru da želi da vidi listu onlajn korisnika.



Kada server pročita ovu poruku, poziva se metoda *PrikazOnlineOsoba*() koja u slučaju da nema dostupnih korisnika, ponovo obaveštava korisnika o tome preko izlaznog toka ka klijentu koji je implementiran preko pomenute klase PrintStream. U slučaju da se neko u međuvremenu ulogovao , lista dostupnih korisnika neće više biti prazna i korisniku se preko istog izlaznog toka ispisuju imena dostupnih korisnika.



Po sličnom principu funkcioniše i ostala komunikacija koja se obavlja između klijenta i servera.