1. domaća zadaća; OPRPP2

Ako još niste, napravite novi radni prostor (engl. *workspace*) u okolini Eclipse. U njemu potom napravite prazan Maven projekt:

- u vršnom direktoriju radnog prostora napravite direktorij `hw01-0000000000` (zamijenite nule Vašim JMBAG-om),
- u tom novom direktoriju oformite Mavenov projekt `oprpp2.jmbag0000000000:hw01-0000000000` (zamijenite nule Vašim JMBAG-om),
- u projekt dodajte ovisnost prema biblioteci `junit` pa
- importajte projekt u Eclipse.

Sada možete nastaviti s rješavanjem zadataka.

Zadatak

U okviru ove zadaće razvit ćete klijenta i poslužitelja za jednostavnu uslugu razgovora („čavrljanja”?) koja za prijenos svojih poruka na prijenosnom sloju koristi protokol UDP.

Klijent i poslužitelj mogu razmjenjivati 5 vrsta poruka:

- `HELLO` (pridružen kod je 1)
- `ACK` (pridružen kod je 2)
- `BYE` (pridružen kod je 3)
- `OUTMSG` (pridružen kod je 4)
- `INMSG` (pridružen kod je 5)

Svaka poruka u memoriji započinje jednim oktetom čija je vrijednost kod poruke (prema prethodnom popisu). Nakon toga slijedi 8 okteta long vrijednost koja predstavlja redni broj paketa. Redni brojevi kreću od nule. Ostatak poruke ovisi o tipu poruke. U nastavku će se koristiti sljedeća sintaksa:

- *byte(nešto)* – ako se predana vrijednost šalje kao jedan oktet, prema `DataOutputStream#writeByte`
- *long(nešto)* – ako se predana vrijednost šalje kao 8 okteta, prema `DataOutputStream#writeLong`
- *str(nešto)* – ako se predana vrijednost šalje kao UTF-8 kodirani string uz prethodno slanje informacije o broju okteta koji čine taj sadržaj, prema `DataOutputStream#writeUTF`

Koristeći razred `DataOutputStream` lagano je u memoriji izgraditi paket određenog formata; primjerice:

```
ByteArrayOutputStream bos = new ByteArrayOutputStream();
DataOutputStream dos = new DataOutputStream(bos);
dos.writeByte(42);
dos.writeLong(75L);
dos.writeUTF("Ovo je string");
dos.close();
byte[] buf = dos.toByteArray();
```

Kako je UDP protokol koji nudi nepouzdanu uslugu prijenosa, naš će aplikacijski protokol biti ostvaren tako da se svaka poruka numerira, te je druga strana dužna poslati paket-potvrdu s istim rednim brojem kako bismo znali da je naš izvorni paket primljen. Ako ne dobijemo potvrdu unutar 5 sekundi, radimo retransmisiju i ponovno čekamo na potvrdu, i to ponavljamo 10 puta. Ako u 10 pokušaja ne uspijemo poslati poruku i dobiti potvrdu, smatramo da je komunikacija nemoguća (i tada bismo, primjerice, ugasili klijenta,

ili na poslužitelju uklonili podatke o vezi s klijentom). Primjetite da prilikom komuniciranja klijent vlastitim rednim brojevima numerira poruke koje on šalje poslužitelju (i koje poslužitelj potvrđuje tim brojevima), a poslužitelj koristi vlastite redne brojeve kojima numerira poruke koje šalje klijentima (i koje onda klijent numerira tim rednim brojevima). Dodatno, poslužitelj za svakog klijenta s kojim priča koristi nezavisni slijed rednih brojeva (drugim riječima, prva poruka koju poslužitelj šalje prema svakom novom klijentu nosit će redni broj 1).

Kako je zamišljen protokol koji će ponuditi uslugu čavrljanja? Pretpostavka je da je poslužitelj pokrenut.

Po pokretanju klijenta, on poslužitelju šalje HELLO koji je numeriran rednim brojem 0, u sebi sadrži ime korisnika te jedan slučajno generirani ključ tipa long. Po primitku ove poruke, poslužitelj za klijenta odabire jedan jedinstveni identifikator tipa long (zvat ćemo ga UID, kao User ID); ovu i druge potrebne informacije poslužitelj dodaje u svoje podatkovne strukture, a klijentu vraća ACK numeriran istim brojem koji je pisao u primljenom HELLO-paketu, te šalje dodijeljeni UID:

Prijava na poslužitelj
Klijent → Poslužitelju: byte(HELLO),long(number),str(Pero Perić),long(randkey)
Poslužitelj → Klijentu: byte(ACK),long(number),long(UID)

Kako je moguće da se ACK izgubi na putu prema klijentu pa da klijent radi retransmisiju HELLO-paketa koji je poslužitelj već prethodno dobio, po primitku HELLO-paketa poslužitelj treba pogledati je li već dogovorio komunikaciju s korisnikom koji je s iste IP-adrese i porta te koji je poslao isti ključ (randkey), pa ako je, onda će iz svojih podataka pročitati koji mu je dodijelio UID i samo poslati odgovarajući ACK (dakle neće dodjeljivati novi UID). Ako nije, onda generira novi UID i sve relevantno pamti u svojim podatkovnim strukturama.

UID-ove je potrebno generirati na sljedeći način: po pokretanju programa odabrati jednu slučajnu long-vrijednost i potom svaki puta kada je potreban UID, uvećati taj broj za 1 i njega koristiti kao UID. Ovo će također (s visokom vjerojatnošću) osigurati da svaki puta kada restartate poslužitelj aktivni UID-ovi klijenata postanu nevažeći.

Kada se klijent gasi, mora se odjaviti s poslužitelja. To radi slanjem paketa BYE, na koji će poslužitelj odgovoriti s ACK (i kod sebe obrisati informacije koje je čuvao o toj vezi s klijentom).

Odjava s poslužitelja
Klijent → Poslužitelju: byte(BYE),long(number),long(UID)
Poslužitelj → Klijentu: byte(ACK),long(number),long(UID)

Ako se klijent odmah nakon prijave želi odjaviti, BYE-paket bio bi numeriran rednim brojem 1. Ako je klijent nakon prijave poslužitelju poslao 5 poruka (koje će potrošiti redne brojeve 1, 2, 3, 4 i 5), redni broj BYE-poruke bio bi 6.

Kada korisnik kroz grafičko sučelje utipka neki tekst (npr. „Hello world!”), klijent stvara poruku OUTMSG u kojoj poslužitelju šalje taj tekst. Poslužitelj će potom za svakog spojenog korisnika (uključivo i ovog koji je poslao taj tekst) stvoriti po jedan paket tipa INMSG i njega poslati svakom od tih korisnika. Dodatno, korisniku koji mu je poslao OUTMSG potvrdit će primitak tog paketa ACK-paketom:

Slanje poruke u chat
Klijent → Poslužitelju: byte(OUTMSG),long(number),long(UID),str(message text)
Poslužitelj → Klijentu: byte(ACK),long(number),long(UID)

te dodatno poslužitelj svakom od klijenata generira i šalje INMSG, kako je prikazano u nastavku.

pa tu poruku onda poslužitelj distribuira svakom od spojenih klijenata
Poslužitelj → Klijentu: byte(INMSG),long(number),str(Ime Prezime),str(message text)
Klijent → Poslužitelju: byte(ACK),long(number),long(UID)

U INMSG „Ime Prezime” su podatci korisnika koji je tekst dostavio poslužitelju porukom OUTMSG, i koji se sada distribuira prema svim klijentima.

Trebate napraviti dva naredbeno-retčana programa: jedan koji predstavlja klijenta s GUI-jem te drugog koji predstavlja poslužitelja. Prilikom pokretanja poslužitelja potrebno je specificirati kroz argument naredbenog retka port na kojem poslužitelj treba slušati. Prilikom pokretanja klijenta potrebno je specificirati kroz argumente naredbenog retka ip-adresu poslužitelja, port poslužitelja te ime korisnika. Evo u nastavku primjera pokretanja jednog poslužitelja i dva klijenta:

```
java -cp štoVećTreba oprpp2.hw01.server.Main 6000
java -cp štoVećTreba oprpp2.hw01.client.Main 127.0.0.1 6000 "Ante Antić"
java -cp štoVećTreba oprpp2.hw01.client.Main 127.0.0.1 6000 "Pero Perić"
```

Za potrebe razvoja na Ferka sam u repozitorij uploadao JAR-arhivu s prototipnom implementacijom obje strane, pa dok razvijate Vašeg klijenta, možete pokrenuti poslužitelj koji sam uploadao direktno iz JAR-arhive (i obrnuto).

Ako uploadanom poslužitelju još dodate pri pokretanju argument -lr, dobit ćete ponešto ispisa (da možete pratiti je li dobio išta); ako mu date argument -d, poslužitelj će se povremeno praviti da nije dobio paket od Vas (u 33% slučajeva) što Vam može poslužiti za testiranje retransmisija.

Okvirni prijedlog implementacije poslužitelja

Za svakog spojenog klijenta održavajte podatkovnu strukturu koja, između ostaloga ima i referencu na red poruka koje treba poslati, te red primljenih potvrda za tog klijenta.

Poslužitelj u glavnoj dretvi sjedi i čeka na pristupnoj točki. Čim nešto primi:

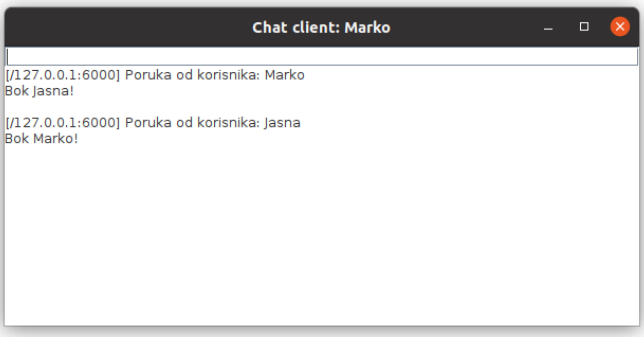
- ako je to HELLO-paket, provjeri što se događa, po potrebi alocira nove podatke za novog klijenta, pošalje ACK tom klijentu i stvori novu dretvu koja će se baviti tim klijentom
- ako je to ACK-paket, pogleda za kojeg je klijenta i gurne ga u njegov red pristiglih potvrda
- ako je to BYE-paket, odradi ga
- ako je to OUTMSG-paket, pošalje potvrdu, za svakog postojećeg klijenta stvori po jedan INMSG s pristiglim tekstom i ubaci ga tom klijentu u red poruka koje treba poslati
- ako je nešto peto, ignorira

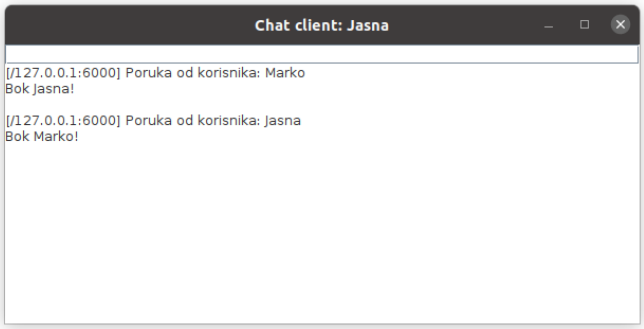
Na poslužitelju, kako je opisano u prvoj točki, svaki će klijent imati jednu dretvu koja se njime bavi, i ona može raditi sljedeće:

```
ponavljaj {
  m = izvadi poruku iz reda poruka za slanje
  retransmisije = 0
  dok( retransmisije < 10 ) {
    retransmisije++;
    kroz spristupnu točku pošalji paket s korektnim rednim brojem
    čekaj do 5 sekundi da se u redu primljenih potvrda pojavi korektna potvrda
      ako se to dogodi, break; inače continue;
  }
}
```

Prilikom implementacija klijenta i poslužitelja predlažem da sve vrste poruka modelirate odgovarajućim razredima; u redove šaljete objekte koji su primjerci tih razreda, te pripremite pomoćne metode koje u skladu s definiranim formatom poruku (objekt) prevode u odgovarajuće polje okteta koje onda možete slati UDP-datagramom, odnosno na temelju polja okteta rekonstruiraju odgovarajući objekt.

Evo i primjera GUI-ja:





Važno. Smijete se konzultirati s Vašim kolegama i razmijenjivati ideje od trenutka kada ste pročitali uputu pa sve do trenutka kada krenete rješavati zadatake. Od trenutka kada napišete prvu liniju koda rješenja daljnja komunikacija je zabranjena. Naravno, u slučaju bilo kakvih nejasnoća, mene uvijek možete kontaktirati.

Dokumentirajte Vaš kod. Projekt i sav izvorni kod mora biti pohranjen uporabom kodne stranice UTF-8.

Kada ste završili sa zadaćom, zapakirajte projekt u ZIP-arhivu imena `hw02-0000000000.zip` (zamijenite nule Vašim stvarnim JMBAG-om). U ZIP-arhivu **ne pakirate** datoteke koje nastaju prevođenjem (tipa: direktorij `target`). Arhivu na Ferka uploadajte **prije** isteka roka. Ako ne stignete predati rješenje do tog roka, na zasebnom mjestu u Ferku bit će omogućena zakašnjela predaja koja će se prihvaćati još najviše 72 sata nakon izvornog roka (uvjeti su specificirani u početnoj prezentaciji).

Nemojte zaboraviti zaključati Vaš upload jer ga inače nećemo prihvatiti.