Računarske mreže, Ispit - JUN1 2023

Pročitati sve zadatke **pažljivo** pre rada - sve što nije navedeno ne mora da se implementira!
Na `Desktop`-u se nalazi zip arhiva. Unutar arhive se nalazi direktorijum u formatu `rm_rok_Ime_Prezime_mXGGXXX` u kome se nalazi validan IntelliJ projekat. Izvući direktorijum iz arhive na Desktop i ubaciti svoje podatke u ime. Otvoriti IntelliJ IDEA, izabrati opciju `Open project` (ne `Import project`!) i otvoriti pomenuti direktorijum. Sve kodove ostaviti unutar već kreiranih Java fajlova. **Kodovi koji se ne prevode se neće pregledati. Nepoštovanje formata ulaza/izlaza nosi kaznu od -10% poena na zadatku!**

1. **UserAgents (20p)**
   Implementirati program koji pomaže pri određivanju prioriteta davanja podrške različitim veb pregledačima za potrebe jedne internet prodavnice. Sa standardnog ulaza se unosi apsolutna putanja do direktorijuma u kojem se skladište log datoteke `nginx` servera te internet prodavnice. Zatim se unosi i broj n koji predstavlja broj niti kojima treba pretražiti sadržaj logova.

   - Implementirati klasu koja sadrži metodu `getLogFiles()`, kojom se pronalaze putanje do svih log datoteka u zadatom direktorijumu. Log datoteke uvek imaju `.log` ekstenziju i mogu se nalaziti u zasebnim poddirektorijumima datog direktorijuma (pretragu vršiti rekurzivno). Kao rezultat rada metode vratiti kolekciju putanja svih pronađenih log datoteka. (4p)
   - Obrada jedne log datoteke podrazumeva pronalaženje `user_agent` niske i odgovarajućeg broja pojavljivanja te niske (tj. frekvencije). Rezultat jedne obrade smestiti u zasebnu datoteku. Na primer, za log datoteku `x.log` napraviti rezultujuću datoteku `x.log.freq` u istom direktorijumu, sa sadržajem narednog oblika:

     ```
     user_agent_1    frequency_1
     user_agent_2    frequency_2
                 . . .
     user_agent_M    frequency_M
     ```

     Svaki log se čuva u zasebnom redu neke log datoteke i sledećeg je formata:

     ```
     addr - user [time] "request" status bytes_sent "referer" "user_agent"
     ```

     Pri čitanju i pisanju obavezno koristiti baferisane tokove. (5p)
   - Korišćenjem n niti obraditi prosleđenu kolekciju putanja ka log datotekama. Prilikom obrade koristiti jednu nit po datoteci i voditi računa da ne dođe do trke sa resursima. (6p)
   - Implementirati klasu koja treba da objedini sve frekvencije iz prethodno kreiranih `*.freq` datoteka u novu datoteku sa nazivom `user_agents.txt` koja je (po redovima posmatrano) leksikografski sortirana. Rezultujuću datoteku smestiti u direktorijum koji je zadat na početku. Koristiti baferisano čitanje i ispis. (5p)

   Primere ulaza i izlaza možete naći u direktorijumu `1-test-primeri`. Na putanji `1-test-primeri/1-in/` nalazi se ulazni direktorijum za prvi test primer. Poddirektorijum `1-test-primeri/1-out/` sadrži rezultujuće datoteke.

Napomena: pretpostaviti da u zadatom direktorijumu i njegovim poddirektorijumima neće biti `user_agents.txt` niti `*.freq` datoteka.

Okrenite stranu!

2. **TCP Sockets (20p)**

- Napraviti Java aplikaciju koja ima ulogu klijenta i koja kao prvi argument komandne linije uzima nisku koja predstavlja `tipKlijenta`, gde `tipKlijenta` može biti proizvoljna niska. Povezati se na lokalni server na portu 8623 koristeći **Socket** klasu. Odmah nakon povezivanja, serveru poslati poruku *hello tipKlijenta*. Sve poruke sa servera ispisati na standardni izlaz. Ostali zahtevi se unose sa standardnog ulaza i oni mogu biti: *list*, *scan* ili *bye*. Na primer, kada korisnik na standardni ulaz unese komandu *list*, poruka *list* se šalje serveru, a odgovor servera se ispisuje na standardni izlaz. Poruke o greškama (neuspešna konekcija, nepoznata komanda i sl) ispisati na standardni izlaz za greške. (3p)
- Klijent, u zasebnoj niti, treba da osluškuje poruke od servera. Kada server pošalje poruku *ping*, on treba na nju da odgovori porukom *pong*. Ova interakcija se ne prikazuje korisniku, tj. ni *ping* ni *pong* se ne ispisuju na standardni izlaz (za potrebe debagovanja, možete ispisivati poruke na standardni izlaz za greške). (3p)
- Napraviti Java aplikaciju koja ima ulogu servera. Pokrenuti lokalni server na portu 8623. Server za svakog primljenog klijenta pokreće zasebnu nit u kojoj će se taj klijent obraditi. Nakon uspostavljanja konekcije, server na standardni izlaz ispisuje poruku koja sadrži IP adresu i port klijenta. (2p)
- Server treba da održava strukturu koja mu omogućava da prati broj klijenata svakog tipa koji trenutno imaju otvorenu konekciju. Ova struktura treba da omogući efikasan odgovor na upit ,,koliko klijenata tipa `foo` je konektovano?" (videti komandu *list* ispod). (3p)
- Server obrađuje poruke na sledeći način:
  - *hello* ***`tipKlijenta`***: evidentira da se povezao servis tipa `tipKlijenta` i odgovara sa *hello* ***`tipKlijenta`***. Klijenti koji ne pošalju *hello* poruku se mogu ignorisati za potrebe svih ostalih komandi. (0.5p)
  - *list*: odgovara porukom formata *tipServisa: broj_instanci* za svaki tip servisa, odvojeno zapetama. Recimo, ako se na server povežu tri klijenta, dva pošalju *hello backend* i jedan *hello frontend* poruku, odgovor na komandu *list* treba da bude *backend: 2, frontend: 1.* (1p)
  - *scan*: server pokreće proveru koji klijenti su i dalje živi, tj. može im se dostaviti poruka, tako što svima pošalje poruku *ping*. Ako dođe do greške pri slanju poruke, zatvoriti konekciju. *pong* poruke možete ignorisati na serveru. Klijentu koji je poslao *scan* komandu, odgovoriti sa *Scan started.* (2p)
  - *bye*: zatvara konekciju. (0.5p)
- Sve greške na serveru i nepoznate komande ispisati na standardni izlaz za greške. (1p)
- Imati u vidu da može postojati mnogo klijenata i da svi oni mogu slati poruke istovremeno i u proizvoljnom poretku između inicijalne *hello* i finalne *bye* poruke, dok god imaju otvorenu konekciju. Obezbediti da ne dolazi do trka za resurse (*race condition*-a). Takođe, obezbediti da se svi resursi u svakom slučaju pravilno zatvaraju. (4p)

U narednom primeru, data je interakcija servera i nekoliko klijenata:

Klijent 1

```
00:00> hello service
hello service
00:01> list
service: 1
00:05> list
service: 2, proxy: 1
//00:07: prima ping,
    odgovara sa pong
00:10> bye
```

Klijent 2

```
00:02> hello proxy
hello proxy
00:03> list
service: 1, proxy: 1
00:06> scan
Scan started.
//00:07: prima ping,
    odgovara sa pong
00:08> bye
```

Klijent 3

```
00:04> hello service
hello service
00:05> list
service: 2, proxy: 1
//00:07: prima ping,
    odgovara sa pong
00:09> list
service: 2
00:10> bye
```

Napomena: Ohrabrujemo studente da koriste ***`netcat`*** *kako bi testirali delimične implementacije i otkrili greške pre vremena. Takodje, ukoliko se npr. preskoči implementacija servera, može se mock-ovati server putem* ***`netcat`****-a.*

Okrenite stranu!

3. **Protocol handlers (15p)**
Pretpostaviti da je dostupan server koji sa klijentima komunicira koristeći protokol **chess**. Opis protokola:

   - Podrazumevani port za protokol je 1996 (TCP).
   - Server održava bazu podataka o šahistima i odgovara na sledeće komande (koje klijent šalje u tekstualnom formatu nakon ostvarivanja konekcije):
     - *sel id* (`id` je tipa `int`)
     - *ins naziv* (`naziv` je tipa `String`) (podrazumevana vrednost za elo je 1300)
     - *upd id elo* (`id` i `elo` su tipa `int`)
     - U slučaju da komanda ne odgovara nijednom od slučajeva opisanim iznad, server vraća nisku `Nevalidan upit.`.
   - Nakon obrade jedne komande (linije teksta koju je klijent poslao) i slanja odgovora klijentu, server prekida vezu.

   Implementirati podršku za URL-ove koji koriste `chess` protokol.

   - Prilikom otvaranja konekcije, formirati vezu koristeći `Socket API`. Povezati se na server i port na osnovu URL-a i otvoriti ulazni tok do odgovora od strane servera. (5p)
   - Omogućiti slanje upita pomoću putanje i parametara URL-a, primer (paziti na enkodiranje karaktera unutar URL-a):

     ```
     chess://localhost:1996/sel?id=2
     chess://localhost:1996/ins?name=Marko+Markovic
     chess://localhost:1996/upd?id=1&elo=900
     ```

     Server šalje nazad rezultat koji klijent ispisuje kao u primeru ispod. (5p)
   - Ukoliko port nije naveden unutar URL-a, iskoristiti podrazumevani port za protokol. (1p)
   - Predefinisati `getInputStream()` metod da vraća ulazni tok do odgovora od strane servera ukoliko je konekcija ostvarena, a `null` ako nije. (1p)
   - Postarati se da je moguće bezbedno koristiti implementirani handler u višenitnom okruženju. (1p)
   - Napisati jednostavan test - kreirati URL, otvoriti konekciju do resursa i ispisati sve podatke koje server pošalje. (2p)

   Napomena: Za simulaciju rada servera je moguće koristiti ***`netcat`****.*

   Primer rada:

```
URL:   chess://localhost
izlaz: Nevalidan upit.

URL:   chess://localhost:12345
izlaz: Povezivanje sa serverom nije uspelo.

URL:   chess://localhost:1996/sel?id=1
izlaz: Fabiano Caruana: 1300

URL:   chess://localhost/ins?name=Marko+Markovic
izlaz: ins je uspesno izvrsen

URL:   chess://localhost/upd?id=1&elo=900
izlaz: upd je uspesno izvrsen

URL:   chess://localhost/upd?elo=900
izlaz: Nevalidan upit.
```