Računarske mreže, Ispit - JAN1

18.01.2021.

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!** Vreme za rad: **3h**.

1. **Čitamo poeziju (15p) (za studente koji nisu radili projekat)**
   Napisati program koji na standardni izlaz ispisuje informacije o pesmama.

   - Sa standardnog ulaza se unosi reč. U direktorijumu `pesme` unutar direktorijuma `tests` na Desktop-u se nalaze datoteke koje sadrže pesme (naziv datoteke je naslov pesme, a svaka linija je jedan stih).
     Na standardni izlaz, za svaku datoteku, ispisati naziv pesme, ispisati najduži stih i koliko se puta reč pojavljuje u toj datoteci. (6p)
   - Za svaku datoteku pokrenuti zasebnu nit koja će obrađivati i ispisivati informacije. (3p)
   - Postarati se da se ispisi svake niti na standardni izlaz ne prepliću. (4p)
   - Postarati se da program ispravno obrađuje specijalnim slučajeve (npr. ako datoteka ne postoji na datoj putanji) i ispravno zatvoriti sve korišćene resurse u slučaju izuzetka. (2p)

```
Spisak datoteka:
    \home\ispit\Desktop\tests\pesme\Dolap.txt
    \home\ispit\Desktop\tests\pesme\DomovinaSeBraniLepotom.txt
    \home\ispit\Desktop\tests\pesme\MozdaSpava.txt
    \home\ispit\Desktop\tests\pesme\LjubavnaPesma.txt
    \home\ispit\Desktop\tests\pesme\AlSuToGrdneMuke.txt
    \home\ispit\Desktop\tests\pesme\PoslednjaPesma.txt
Ulaz:   kao
Izlaz:
    Dolap
    Pusti snovi! Napred, vrance, nemoj stati,
    4
    DomovinaSeBraniLepotom
    Sestrinom suzom majcinom brigom
    0
    MozdaSpava
    Ja sad nemam svoju dragu, i njen ne znam glas.
    3
    LjubavnaPesma
    I sa mnom ljube, ceznu, strepe, zude!
    0
    AlSuToGrdneMuke
    kad bi mogle da se pruze!
    0
    PoslednjaPesma
    Njen je plast suncani mozda tkivo lazi;
    2
```

---

*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!

2. **TCP Sockets (25p/18p)**
Napraviti Java klijent-server TCP igru iks-oks.

- Kreirati Java aplikaciju koja ima ulogu iks-oks igrača - klijenta. Klijenti se povezuju na lokalni server na portu 12345 i periodično izvršavaju naredne operacije sve dok server ne signalizira kraj igre: (3p)
  - ispis trenutnog stanja igre tj. iks-oks table (proizvoljno implementirati)
  - ucitavanje poteza od strane igraca (potez proizvoljno implementirati)
  - slanje poteza serveru (nije potrebno implementirati validaciju poteza na klijentskoj strani)
- Kreirati Java aplikaciju koristeći TCP Sockets API koja ima ulogu iks-oks servera. Server nakon pokretanja najpre čeka da se dva klijenta povežu a zatim pokreće igru. Nakon završetka igre, server ponovo čeka na igrače i nakon toga započinje novu igru. (4p)
- Implementirati iks-oks igru na strani servera. Naizmenično čitati poteze igrača, ažurirati stanje igre i poslati ažurirano stanje klijentima. Kad se igra završi, poslati igračima informaciju da se igra završila i prekinuti vezu sa igračima. (9p)
- U slučaju da igrač pošalje nevalidan potez serveru, server odgovara nazad porukom `Nevalidan potez`, i očekuje ponovo potez od istog igrača. (2p)
- Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (2p)

```
(klijent 1)          |
                     |   |      (klijent 2)
                     |   |
    ---              |   |           ---
    ---              |   |           -X-
    ---              |   |           ---
                     |   |
    > 5              |   |           > 3
    --O              |   |           --O
    -X-              |   |           -XX
    ---              |   |           ---
                     |   |
    > 6              |   |           > 4
    --O              |   |           --O
    OXX              |   |           OXX
    ---              |   |           -X-
                     |   |
    > 2              |   |           > 1
    OXO              |   |           OXO
    OXX              |   |           OXX
    ---              |   |           -X-
                     |   |
    > 8              |   |
    OXO              |   |
    OXX              |   |
    -X-              |   |
```

3. **UDP Sockets (20p/12p)**
Napraviti UDP server koji računa površine krugova čije poluprečnike šalju klijenti.

- Napraviti Java klasu koja ima ulogu UDP klijenta koristeći Java Datagram API. Klijent šalje datagram serveru na portu 31415 u kome se nalazi jedan realan broj učitan sa standardnog ulaza. Primiti i ispisati odgovor servera na standardni izlaz. (9p/5p)
- Napraviti Java klasu koja ima ulogu lokalnog UDP servera koji osluškuje na portu 31415, koristeći Java `UDP` API. Nakon primanja datagrama, server odgovara klijentu realnim brojem koji predstavlja površinu kruga čiji je poluprečnik jednak broju koji je klijent poslao. Ukoliko server primi negativan broj od klijenta, klijentu odgovoriti porukom `Neispravan poluprecnik`. (10p/6p)
- Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (1p)

Računarske mreže, Ispit - JAN2

04.02.2021.

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!** Vreme za rad: **3h**.

1. **Log-file parser (15p) (za studente koji nisu radili projekat)**
   Napraviti Java aplikaciju koja selektivno parsira log fajl.

   - Napraviti Java aplikaciju koja prima putanju do regularnog fajla koji predstavlja log fajl i čita ga koristeći URL klasu i FILE protokol. Ispisati sadržaj fajla na standardni izlaz. (3p)
   - U fajlu se nalaze linije u sledećem formatu:
     `[<DATUM_VREME>][<IP_ADRESA>][<URL_DO_RESURSA_NA_SERVERU>]` npr.
     `[12.12.2010 17:41:00][123.123.123.123][http://test.com/path/secret.txt]`
     `[12.12.2010 17:51:03][2607:f0d0:1002:51::4][https://test.com/path/secret.txt]`
     `[15.12.2010 17:52:11][123.23.2.33][ftp://test.com/secret.txt]`
     `[15.12.2010 17:52:21][123.23.2.33][FILE:///home/test/secret.txt]`
     Filtrirati pročitani sadržaj tako da se na standardni izlaz (umesto ispisivanja čivavog fajla) ispišu samo one linije u kojima je zahtev za resurs stigao preko HTTP ili HTTPS protokola. (3p)
   - Format ispisa linije na standardni izlaz promeniti na:
     `v<VERZIJA_IP_ADRESE>:<KORIŠĆENI_PROTOKOL>:<PUTANJA_DO_RESURSA>`
     (zameniti tagove odgovarajućim informacijama) npr.
     `v4:http:/path/secret.txt` (2p)
   - Ispisati samo one linije u kojima je vrednost niske koja predstavlja datum i vreme hronološki pre trenutnog datuma i vremena. (4p)
   - Preskočiti linije u kojima je unutar URL-a naveden nepodrazumevani port za odgovarajući protokol. (2p)
   - Postarati se da u slučaju izuzetka aplikacija ispravno zatvori korišćene resurse i tokove podataka. (1p)

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!

2. **UDP Sockets (20p/12p)**
Napraviti osnovu za klijent-server Java aplikaciju koristeći *Java UDP API* koja vrši odgovarajuće kodiranje niske.

- Napraviti Java aplikaciju koja ima ulogu UDP klijenta. Poslati datagram lokalnom serveru na portu 12345 koji sadrži nisku (koja može da sadrži beline). Niska se učitava sa standardnog ulaza na klijentskoj strani. Primiti datagram koji predstavlja odgovor servera i rezultat ispisati na standardni izlaz. (8p/4p)
- Napraviti Java aplikaciju koja ima ulogu servera koji sluša na portu 12345. Kada dobije nisku od klijenta, tu nisku transformiše na sledeći način:
  - Ukoliko je karakter veliko slovo, postaje malo slovo i udvostručuje se.
  - Ukoliko je karakter malo slovo, postaje veliko slovo.
  - Ukoliko je karakter broj, postaje karakter **'.'** i udvostručuje se.

  Primer:
  niska: `mReze Ispit 2`
  transformisana niska: `MrrEZE iiSPIT ..`
  Poslati datagram pošiljaocu u kome se nalazi transformisana niska. (10p/7p)
- Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (2p/1p)

3. **Non-Blocking IO - Loto (25p/18p)**
Napraviti klient-server Java aplikaciju koristeći *TCP Sockets/Channels API* preko koje se računa broj pogodaka u igri Loto.

- Napisati Java klasu koja ima ulogu **blokirajućeg** TCP klijenta koristeći Java Channels API. Klijent formira konekciju sa lokanim serverom na portu 12345 i zatim šalje serveru jednu Loto kombinaciju, to jest 7 celih brojeva iz intervala [1, 39], učitanu sa standardnog ulaza. (8p/6p)
- Napisati Java klasu koja ima ulogu lokalnog **neblokirajućeg** TCP servera, koji osluškuje na portu 12345, koristeći Java Channels API. Server slučajnim izborom generiše jednu Loto kombinaciju. Nakon konekcije, klijent šalje serveru učitanu Loto kombinaciju. Brojevi unutar kombinacije se ne smeju ponavljati. Kao odgovor, server šalje klijentu informaciju o broju pogodaka u serverski generisanoj kombinaciji. (15p/11p)
- Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (2p/1p)

Računarske mreže, Ispit - JUN1 2021

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. **URL Scanner (15p) (za studente koji nisu radili projekat)**
   Napisati Java aplikaciju koja obrađuje URL-ove. U direktorijumu `urls`, unutar direktorijuma `tests` na Desktop-u, nalaze se datoteke koje sadrže spisak URL-ova (po jedan u svakoj liniji).

   - Za svaku datoteku pokrenuti zasebnu nit koja će obrađivati i ispisivati statistiku za tu datoteku. (2p)
   - Za svaku pročitanu liniju datoteke kreirati novi URL objekat koristeći `URL` klasu. Preskočiti sve linije koje ne predstavljaju validan URL. (1p)
   - Za svaki validni URL ispisati protokol, `authority` i putanju iz URL-a koristeći URL klasu. Izlaz formatirati na sledeći način:
     `<KORIŠĆENI_PROTOKOL> <AUTHORITY> <PUTANJA_DO_RESURSA>`
     Primer:
     ulaz: `http://www.matf.bg.ac.rs:3030/dir1/dir2/test.txt`
     izlaz: `http www.matf.bg.ac.rs:3030 /dir1/dir2/test.txt` (4p)
   - Postarati se da se ispisi svake niti na standardni izlaz ne prepliću. (3p)
   - Ukoliko se unese IP adresa unutar URL-a, dodatno uz informacije iznad ispisati i informaciju o verziji IP adrese koja je uneta i ako je to IPv4 adresa ispisati njene bajtove, u formatu:
     `(v<VERZIJA_IP_ADRESE>) <KORIŠĆENI_PROTOKOL> <PUTANJA_DO_RESURSA> [<BAJTOVI_ADRESE>]`
     Primer:
     ulaz: `http:///123.123.123.123:80/dir1/dir2/test.txt`
     izlaz: `(v4) http /dir1/dir2/test.txt [123 123 123 123]`
     ulaz: `sftp://2001:0db8:85a3:::8a2e:0370:7334/dir1/dir2/test.txt`
     izlaz: `(v6) sftp /dir1/dir2/test.txt` (4p)
   - Postarati se da program ispravno obradjuje specijalne slučjeve i ispravno zatvoriti sve korišćene resurse u slučaju izuzetka. (1p)

---

*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!

2. **UDP Sockets (20p/12p)**
Napraviti osnovu za klijent-server Java aplikaciju koristeći `Datagram API` koja šifruje poruku koristeći Morzeov kôd. Morzeova azbuka se sastoji od dva simbola: `.` (*dit*) i `-` (*dah*). Svaki karakter se kodira odgovarajućim nizom ovih simbola, a čitave poruke se kodiraju tako što se dodaje pauza nakon svakog karaktera, kao i duža pauza nakon svake reči.

- Napraviti Java aplikaciju koja ima ulogu UDP klijenta. Poslati datagram lokalnom serveru na portu 23456 koja sadrži nisku (koja može da sadrži beline). Niska se učitava sa standardnog ulaza na klijentskoj strani. Primiti datagram koji predstavlja odgovor servera i rezultat ispisati na standardni izlaz. (8p/4p)
- Napraviti Java aplikaciju koja ima ulogu servera koji sluša na portu 23456. Kada dobije nisku od klijenta tu nisku transformiše Morzeovom azbukom u šifrovanu poruku koristeći datoteku *morse.txt*. Posle svakog karaktera dodati razmak koji predstavlja pauzu, a nakon svake reči dodati tri razmaka koji predstavljaju trostruku pauzu. Na kraju poruke dodati signal za kraj: `.-.-.-` (10p/7p)
- Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (2p/1p)

```
Klijent:   Morze
Server:    -- --- .-. --.. .   .-.-.-

Klijent:   Orao Pao
Server:    --- .-. .- ---   .--. .- ---   .-.-.-
```

3. **Non-Blocking IO (25p/18p)**
Napraviti klient-server Java aplikaciju koristeći `TCP Sockets/Channels API` koja se može iskoristiti kao osnova za kreiranje kartaških igara.

- Napisati Java klasu koja ima ulogu **blokirajućeg** TCP klijenta koristeći `Java Channels API`. Klijent formira konekciju sa lokanim serverom na portu 12345 i zatim šalje serveru broj učitan sa standardnog ulaza koji predstavlja broj karti koje se izvlače iz serverskog špila. (4p/3p)
- Napisati Java klasu koja ima ulogu lokalnog **neblokirajućeg** TCP servera, koji osluškuje na portu 12345, koristeći `Java Channels API`. Karte predstaviti na *proizvoljan* način, sa narednim ograničenjima:
  - Svaka karta se sastoji od vrednosti i znaka
  - Vrednost karte može biti od 2 do 10 za karte bez slike dok žandar ima vrednost 11, kraljica 12, kralj 13 i kec 14
  - Znak može biti `pik`, `herc`, `tref` ili `karo`

  (2p/2p)
- Server slučajnim izborom generiše jean špil karata prilikom pokretanja. Kreirati standardni špil od 52 karte i promešati ga prilikom kreiranja servera. Nakon toga ispisati špil na standardni izlaz, svaku kartu u zasebnom redu, u formatu `vrednost.znak`, na primer `4.Pik` ili `14.Karo`. (3p/2p)
- Nakon konekcije, klijent šalje serveru broj karti koje želi da izvuče iz špila. Server izvlači toliko karti sa vrha špila šalje ih klijentu (proizvoljno implementirati slanje). Jednom izvučene karte se ne vraćaju u špil. Klijent ispisuje primljene karte na standardni izlaz. (10p/8p)
- Ukoliko je broj koji je klijent poslao veći od trenutnog broja karata u špilu ili manji od 1, poslati tu informaciju korisniku na proizvoljan način i ispisati odgovarajuću poruku na klijentskoj strani. (4p/2p)
- Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (2p/1p)

Računarske mreže, Ispit - JUN2 2021

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. **Threads/URL(17p) (za studente koji nisu radili projekat)**
   U direktorijumu **urls**, unutar direktorijuma **tests** na Desktopu, nalaze se datoteke koje sadrže spisak URL-ova (po jedan u svakoj liniji) i brojeva.

   - Za svaku datoteku pokrenuti zasebnu nit koja će je obrađivati i ispisivati statistiku za tu datoteku. (4p)
   - Za svaku pročitanu liniju datoteke kreirati novi URL objekat koristeći URL klasu. Preskočiti sve linije koje ne predstavljaju validan URL. (2p)
   - Za validni URL ispisati protokol i putanju iz URL-a koristeći URL klasu. Izlaz formatirati na sledeći način: `PUTANJA_DO_RESURSA KORIŠĆENI_PROTOKOL` (5p)

     ```
     ulaz: 3 http://www.matf.bg.ac.rs:3030/dir1/dir2/test.txt
     izlaz: /dir1/dir2/test.txt http
     ```

   - Postarati se da se ispisi niti na standardni izlaz ne prepliću. (3p)
   - Početak svake linije u datotekama je broj koji označava dužinu linije obrade (u daljem tekstu n). Ukoliko je ime hosta unutar URL-a navedeno putem IP adrese, dodatno uz informacije iznad ispisati linije obrade (matrica $n \times n$ poput matrice u primeru) i informaciju o verziji IP adrese, kao u primeru ispod. (3p)

     ```
     ulaz: 6 http://123.123.123.123:80/dir1/dir2/test.txt
     izlaz:
     >=====
     =>====
     ==>===
     ===>==
     ====>=
     =====>
     (v4) /dir1/dir2/test.txt http
     ulaz: 4 sftp://2001:0db8:85a3:::8a2e:0370:7334/dir1/dir2/test.txt
     izlaz:
     >===
     =>==
     ==>=
     ===>
     (v6) /dir1/dir2/test.txt sftp
     ```

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!

2. **Kviz (28p/20p)**
Napraviti TCP klijent-server aplikaciju preko koje se korisnici takmiče u kvizu. Server ima ulogu sudije u kvizu i vodi evidenciju o poenima takmičara (klijenata).

- Napraviti Java klasu koja ima ulogu lokalnog TCP servera (koristeći *Java Sockets API*) koji osluškuje na portu 12321. Pri pokretanju, server sa standardnog ulaza učitava nisku koja predstavlja putanju do direktorijuma u kome se nalaze fajlovi sa pitanjima za kviz. Naziv fajla predstavlja naziv oblasti iz koje su pitanja u njemu, a svaki fajl ima format kao u primeru ispod. (4p/3p)
- Napraviti Java klasu koja ima ulogu TCP klijenta (koristeći *Java Sockets API*). Klijent formira konekciju sa lokanim serverom na portu 12321. Nakon prihvatanja svakog novog klijenta, server kreira novu nit koja će preuzeti dalju komunikaciju sa njim. Nakon uspostavljanja konekcije, klijent šalje serveru nisku koja predstavlja ime tog klijenta, učitanu sa standardnog ulaza, a zatim od servera dobija nazive oblasti iz kojih može da se takmiči. (6p/5p)
- Pošto klijent odabere oblast iz koje želi da dobija pitanja, započinje kviz. Server šalje klijentu pitanje po pitanje iz odabrane oblasti i čeka 5 sekundi da pročita odgovor od klijenta. (2p/2p)
- U slučaju da takmičar (klijent):
  (a) zakasni da pošalje odgovor — server mu ne menja rezultat i odgovara porukom:
  `Niste stigli da odgovorite na vreme.`
  (b) pogrešno odgovori — server umanjuje rezultat tog klijenta i odgovara porukom:
  `Netačan odgovor. Izgubili ste 1 poen.`
  (c) tačno odgovori — server mu dodaje na trenutni rezultat broj poena koje nosi to pitanje i obaveštava ga o tome porukom:
  `Tačan odgovor. Osvojili ste <broj poena>.`
  (d) pošalje kao odgovor opciju: `Ne znam` — server mu ne menja rezultat i odgovara porukom:
  `Niste znali tačan odgovor.`

  (7p/5p)
- Server sve vreme vodi evidenciju o najbolja 3 rezultata iz svake oblasti. (3p/2p)
- Nakon potrošenih svih pitanja u fajlu, server šalje klijentu poruku `Kviz je zavrsen!`. Dodatno, ukoliko je rezultat među najbolja 3 rezultata do sada iz oblasti, šalje mu i poruku `Medju najbolja tri ste rezultata iz ove oblasti do sada!` i ažurira evidenciju o najboljim takmičarima. (3p/2p)
- Ukoliko klijent odustane pre kraja kviza i ne odgovori na sva pitanja, smatrati da je odustao i njegove poene ne računati za ukupan plasman. Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (2p/1p)

```
ulaz: /home/ispit/Desktop/Kviz/Geografija.txt
Sadrzaj fajla:
1.Koji je glavni grad Holandije? Amsterdam 3
2.Koje je najvece po povrsini ostrvo na svetu? Grenland 5
3.Koja je najsevernija prestonica na kopnu Evrope? Helsinki 7
4.Koja je najduza reka na svetu? Nil 5
```

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/10p)**
Implementirati podršku za URL-ove koji koriste `quiz` protokol. Opis protokola je dat u prethodnom zadatku.

- 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/3p)
- Omogućiti slanje upita pomoću parametra `oblast` iz URL-a, npr. za upit , kompletan URL bi bio:

  ```
  quiz://localhost:1337?oblast=Geografija
  ```

  Server šalje nazad pitanje koja klijent ispisuje kao u primeru ispod. (5p/3p)
- Ukoliko port nije naveden unutar URL-a, iskoristiti predefinisani podrazumevani port isti kao u prethodnom zadatku. (1p/1p)
- Predefinisati `getInputStream()` metod da vraća ulazni tok do odgovora od strane servera ukoliko je konekcija ostvarena, a `null` ako nije. (1p/1p)
- Postarati se da je moguće bezbedno koristiti implementirani handler u višenitnom okruženju. (1p/1p)
- Napisati jednostavan test - kreirati URL, otvoriti konekciju do resursa i ispisati sve podatke koje server pošalje. (2p/1p)

```
URL:   quiz://localhost
izlaz: Nije uneta oblast.

URL:   quiz://localhost:12345?oblast=Geografija
izlaz: Koji je glavni grad Holandije?

URL:   quiz://localhost:7337?oblast=Geografija
izlaz: Neuspela konekcija.

URL:   quiz://localhost?oblast=x
izlaz: Nepostojeca oblast.
```

Računarske mreže, Ispit - SEP1 2021

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. Dekompresovati arhivu na Desktop i ubaciti svoje podatke u ime pomenutog direktorijuma.
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. **Selektivno kopiranje fajla (15p) (za studente koji nisu radili projekat)**

   - Napraviti Java aplikaciju koja koristeći odgovarajuće ulazne i izlazne tokove kopira sadržaj tekstualnog fajla sa imenom koje se unosi preko standardnog ulaza u fajl `timestamps.txt`. Postarati se da se u slučaju izuzetka prikaže odgovarajuća poruka (različita za različite tipove izuzetaka). (3p)
   - Prekopirati samo one niske koje predstavljaju validne vremenske niske u formatu `DD-MM-YYYY`. Pretpostaviti da svi meseci imaju najviše 31 dan i da je godina veća od 2000 (npr. `02-12-2015`). (6p)
   - Koristiti baferisanje ulaznog i izlaznog toka zarad smanjenja broja IO operacija. (2p)
   - Niske ispisati u fajl tako da po jedna niska bude u svakoj liniji. (2p)
   - Podesiti kodne strane za oba fajla na UTF-8. (1p)
   - Postarati se da se u slučaju izuzetka garantuje da su zatvoreni svi korišćeni resursi. (1p)

2. **TCP Sockets - Aerodromi (20p/12p)**
   Napraviti osnovu za TCP klijent-server Java aplikaciju koja pruža informacije o odlaznim letovima sa aerodroma.

   - U direktorijumu `aerodromi`, unutar direktorijuma `tests` na Desktop-u, nalaze se tekstualni fajlovi koji sadrže informacije o odlaznim letovima sa nekog aerodroma za taj dan. Ime fajla predstavlja ime grada u kome se aerodrom nalazi, a svaka linija fajla sadrži informacije o jednom leto i oblika je:
     `<JEDINSTVENA_ŠIFRA_LETA> <GRAD_SLETANJA> <VREME_POLETANJA> <VREME_SLETANJA>`.
     Na serverskoj strani keširati ove podatke kako se ne bi čitali iznova za svakog klijenta. (4p/3p)
   - Napraviti Java klasu koja ima ulogu lokalnog TCP servera koji osluškuje na portu 12345. Svakom novom klijentu server šalje spisak svih gradova za čije aerodrome ima informacije o letovima. (4p/2p)
   - Napraviti Java klasu koja ima ulogu lokalnog TCP klijenta. Nakon uspostavljanja konekcije sa serverom na portu 12345, klijent čeka spisak gradova od servera i ispisuje ih na standardni izlaz. Nakon toga, klijent dodatno šalje serveru ime grada, uneto sa standardnog ulaza, sa čijeg aerodroma želi da dobije informacije o odlaznim letovima, a zatim ponovo čeka odgovor od servera koji ispisuje na standardni izlaz i završava sa radom. (7p/4p)
   - U zavisnosti od imena grada, server šalje keširane informacije iz odgovarajućeg fajla. (4p/2p)
   - Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (1p)

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. **Non-Blocking IO (25p/18p)**
   Napraviti klient-server Java aplikaciju koristeći `TCP Sockets/Channels API`.

   - Napisati Java klasu koja ima ulogu **blokirajućeg** TCP klijenta koristeći `Java Channels API`. Klijent formira konekciju sa lokalnim serverom na portu 12345 i zatim šalje serveru 4 bajta jedan za drugim, učitani sa standardnog ulaza. (3p/2p)
   - Napisati Java klasu koja ima ulogu lokalnog **neblokirajućeg** TCP servera, koji osluškuje na portu 12345, koristeći `Java Channels API`. Server čuva skriveni ceo broj. Prilikom obrađivanja klijenta server prihvata bajtove od viših ka nižim i kreira ceo broj (tipa `int`). Na primer, ukoliko klijent šalje redom bajtove `0xA0`, `0xB0`, `0xC0` i `0xD0`, server kreira broj `0xA0B0C0D0`. Nakon uspešnog prihvatanja bajtova, server klijentu vraća novi broj nastao primenom operacije XOR na prihvaćene bajtove i skrivenog broja. (6p/4p)
   - Server slučajnim izborom generiše skriveni ceo broj prilikom pokretanja i ispisuje ga na standardni izlaz. Broj mora biti veći od 999 i prost. (10p/8p)
   - Klijent ispisuje odgovor od servera na standardni izlaz. Klijent može više puta da učitava i šalje bajtove serveru dok ne prekine vezu. (3p/2p)
   - Ukoliko klijent serveru pošalje broj bez ijednog bita postavljenog na `1`, server klijentu vraća `-1` (1p/1p)
   - Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (2p/1p)

Računarske mreže, Ispit - SEP1 2021

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 IntelliJ projekat u formatu `rm_rok_Ime_Prezime_mXGGXXX`.
Dekompresovati arhivu na Desktop i ubaciti svoje podatke u ime pomenutog direktorijuma.
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. **TCP Sockets (25p/18p)**
   Napraviti osnovu za TCP klijent-server Java aplikaciju koja pomaže u održavanju server farme. Održavanje server farme podrazumeva obilazak i obradu svakog računara. Takodje, u slučaju kvara, neophodno je ispraviti odgovarajući računar. Pošto je broj računara u farmi veliki, potrebno je puno tehničara i sistem za praćenje trenutnog stanja farme. Sistem bi trebalo da čuva status za svaki od računara u mrežnoj topologiji farme i da omogući tehničarima da udaljeno promene statuse računara. Odlučeno je da se za ove potrebe kreira Java serverska aplikacija koja će omogućiti tehničarima da se povežu i postavljaju upite sistemu — kakvo je trenutno stanje farme, markiraj računar `X` kao pokvaren, itd. Topologija serverske farme je takva da se može predstaviti matricom veličine `NxN` (izgledno je da će se veličina menjati u budućnosti) i da se svaki računar može "adresirati"parom brojeva koji predstavljaju vrstu i kolonu u matrici, redom.

   - Napraviti Java klasu koja ima ulogu lokalnog TCP servera koji osluškuje na portu 13370. Pre pokretanja servera, zatražiti od korisnika da unese broj `N` koji predstavlja veličinu matrice server farme. Nakon toga, server osluškuje na pomenutom portu i svakom novom klijentu inicijalno šalje stanje serverske farme — trenutno stanje matrice farme. Ispisati matricu kao u primeru ispod — sa `o` su označeni ispravni, a sa `x` neispravni računari. Prilikom pokretanja servera pretpostaviti da su svi računari ispravni. Server svakog klijenta treba da obradi u zasebnoj niti. (4p/2p)
   - Napraviti Java klasu koja ima ulogu lokalnog TCP klijenta. Nakon uspostavljanja konekcije sa serverom na portu 12345, klijent čeka na trenutno stanje farme od servera i ispisuje ga na standardni izlaz. Nakon toga, klijent šalje serveru upite koje tehničar unosi sa standardnog ulaza, sve dok se ne unese upit `exit`, nakon čega klijent prestaje s radom. (3p/2p)
   - Nakon slanja stanja farme klijentu, server čeka na upite klijenta. Upiti mogu biti: (13/11p)
     (a) `list` — odgovor servera treba da bude trenutno stanje farme
     (b) `mark X Y` — markira server sa koordinatama `(X, Y)` kao neispravan i šalje klijentu ažurirano stanje farme
     (c) `repair X Y` — uklanja marker neispravnosti serveru sa koordinatama `(X, Y)` i šalje klijentu ažurirano stanje farme
     (d) `exit` — server prestaje da obradjuje klijenta i raskida vezu (bez slanja odgovora klijentu)
     (ukoliko koordinate nisu ispravne, server ne vrši nikakve modifikacije)
   - Sinhronizovati pristup modelu farme na serverskoj strani. (3p/2p)
   - Postarati se da su svi resursi ispravno zatvoreni u slučaju izuzetka. (2p/1p)

```
$ java Client.java
+|01234
++-----
0|ooooo
1|ooooo
2|ooooo
3|ooooo
4|ooooo
```

```
(upit) mark 3 3
+|01234
++-----
0|ooooo
1|ooooo
2|ooxoo // markirano polje (3,3)
3|oooox // drugi tehnicar markirao
4|ooooo
```

```
(upit) repair 3 3
+|01234
++-----
0|oooxo
1|ooooo
2|ooooo // ispravljeno
3|oooox
4|ooooo
```

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!

2. **Protocol handlers (20p/12p)**
Implementirati podršku za URL-ove koji koriste `farm` protokol. Opis protokola je dat u prethodnom zadatku, pretpostaviti da je server pokrenut na istom portu kao i u prethodnom zadatku.

   - 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/3p)
   - Omogućiti slanje upita pomoću parametra `oblast` iz URL-a, npr. za upit `repair 3 4`, kompletan URL bi bio:

     ```
     farm://localhost:13370?q=repair&x=3&y=4
     ```

     Server šalje nazad odgovor u primeru iznad (jedno stanje farme prilikom ostvarivanja konekcije, tj. pre upita, i jedno stanje farme nakon upita). (5p/3p)
   - Ukoliko port nije naveden unutar URL-a, iskoristiti predefinisani podrazumevani port isti kao u prethodnom zadatku. (1p/1p)
   - Predefinisati `getInputStream()` metod da vraća ulazni tok do odgovora od strane servera ukoliko je konekcija ostvarena, a `null` ako nije. (1p/1p)
   - Postarati se da je moguće bezbedno koristiti implementirani handler u višenitnom okruženju. (1p/1p)
   - Napisati jednostavan test - kreirati URL, otvoriti konekciju do resursa i ispisati sve podatke koje server pošalje. (2p/1p)

3. **Matrice (15p) (za studente koji nisu radili projekat)**
Napisati program koji na standardni izlaz ispisuje informacije o matricama.

   - U direktorijumu `fajlovi` unutar direktorijuma `tests` na Desktop-u se nalaze datoteke koje sadrže matrice (prva dva broja odgovaraju dimenzijama matrice nakon čega slede elementi matrice). Na standardni izlaz, za svaku datoteku sa ekstenzijom *.txt*, ispisati naziv datoteke i da li je matrica gornje trougaona ili ne (ispisati *da* ili *ne*). Matrica je gornje trougaona ako su elementi ispod dijagonale jednaki 0. (6p)
   - Za svaku datoteku pokrenuti zasebnu nit koja će obrađivati i ispisivati informacije. (3p)
   - Postarati se da se ispisi svake niti na standardni izlaz ne prepliću. (4p)
   - Postarati se da program ispravno obrađuje specijalnim slučajeve (npr. ako datoteka ne postoji na datoj putanji) i ispravno zatvoriti sve korišćene resurse u slučaju izuzetka. (2p)