1. Chat klijent – komunikacija sa serverom

1.1 Cilj vježbe

Dodati komunikaciju sa Char serverom na ekranu za chat klijent.

1.2 Opis

1.2.1 Otvaranje konekcije prema serveru

Komunikacija sa serverom se ostvaruje putem TCP konekcije. U javi se takva konekcija ostvaruje sa Socket klasom:

Socket soc = new Socket(UserConfig.getHost(), UserConfig.getPort());

Parametri konstruktora su: naziv servera (localhost) i port na koji se treba spojiti (4444). Klasa Socket definira dva toka za komunikaciju sa serverom. OutputStream, preko metode getOutpuStream za slanje podataka prema serveru i InputStrema (metoda getInputStream) za čitanje podataka poslanih sa servera. Oba toka šalju/primaju bytove podataka. Pošto se trebaju slati i primati tekstualni podaci (i to liniju po liniju) bolje je koristiti klase iz hijerarhije Writer i Reader. Za slanje podataka koristit ćemo klasu PrinterWriter a za primanje BufferedReader jer obe klase rade sa karakterima i imaju metode koje šalju/primaju cijelu liniju podataka:

PrinterWriter se može direktno povezati sa OutputStream klasom:

PrintWriter pw = new PrintWriter(soc.getOutputStream());

dok BufferedReader prima samo klase iz hijerarhije Reader tako da se najprije treba InputStream povezati sa InputStreamReader:

BufferedReader br = new BufferedReader(new InputStreamReader(soc.getInputStream()));

Metoda koja radi spajanje na server:

```
	private void connect(){
		try {
			soc = new Socket(UserConfig.getHost(), UserConfig.getPort());
			pw = new PrintWriter(soc.getOutputStream());
			br = new BufferedReader(new
InputStreamReader(soc.getInputStream()));
			String response;
			try {
				response = br.readLine();
				if (txtChat.getText().length()>0)
					txtChat.append("\n");
				txtChat.append(response);
				textField.setText(null);
			} catch (IOException e) {
				log.error("Greška kod čitanja inicijalnog odgovora", e);
```

```
                        JOptionPane.showMessageDialog(txtChat, "Greška kod
čitanja inicijalnog odgovora", "Greška!", JOptionPane.ERROR_MESSAGE);
                }
            } catch (UnknownHostException e) {
                log.error("Nepoznati host", e);
                this.dispose();
            } catch (IOException e) {
                log.error("IO iznimka", e);
                this.dispose();
            }
        }
```

Metoda koja šalje i prima poruku od servera:

```
        private void send(){
            pw.println(textField.getText());
            if (pw.checkError())
            {
                JOptionPane.showMessageDialog(txtChat, "Greška kod slanja
poruke", "Greška!", JOptionPane.ERROR_MESSAGE);
            }
            String response;
            try {
                response = br.readLine();
                if (txtChat.getText().length()>0)
                    txtChat.append("\n");
                txtChat.append(response);
                textField.setText(null);
            } catch (IOException e) {
                log.error("Greška kod čitanja", e);
                JOptionPane.showMessageDialog(txtChat, "Greška kod čitanja
odgovora", "Greška!", JOptionPane.ERROR_MESSAGE);
            }
        }
```

Zadatak: Dodati ove dvije metode na ekran za chat klijent (promijeniti varijable da odgovaraju vašem programu) te dodati pozive ovih metoda na odgovarajuća mjesta. Potrebno je text koji se upiše u JtextField komponente poslati serveru a odgovor upisati u JTextArea komponentu.

U chat.properties datoteci (datoteka gdje se spremaju postavke aplikacije) treba postaviti ispravne vrijednosti za host i port (localhost, 4444).

Za testiranje ekrana najprije je potrebno pokrenuti SocketServer aplikaciju. Raspakirajte SocketServer.zip datoteku u neki direktorij. U Command promptu se trebate pozicionirati u taj direktorij i pokrenuti server:

java SocketServer

2. Chat klijent – logiranje

2.1 Cilj vježbe

Dodati logiranje najvažnijih događaja u aplikaciji i napraviti JAVADOC dokumentaciju za klasu koja sadržava postavke aplikacije.

2.2 Opis

2.2.1 Logiranje pomoću paketa Logback

Logback se bazira na tri elementa Logger, Appender i Layout/Encoder. Pomoću te tri komponente može se u kodu logirati po tipu i razini poruke te se može definirati kako te poruke izgledaju i gdje se zapisuju.

Logger je klasa koje služi za logiranje. Svaki Logger objekt ima svoj naziv i poziciju u hijerarhiji. To nam omogućava da se odrede različite postavke za različite Loggere. Svaki Logger objekt je pridružen LoggerContext-u koji je zadužen za kreiranje Logger-a i njihovim slaganjem u hijerarhijsku strukturu. Nazivi Logger-a su osjetljiva na veličinu slova i točka definira granicu hijerarhije. Npr. Logger koji ima naziv „com.foo“ je roditelj od Logger-a „com.foo.Bar“.

Logger rootLogger = LoggerFactory.getLogger(„naziv.loggera“);

```
package org.slf4j;
public interface Logger {

  // Osnovne metode za ispis poruka:
  public void trace(String message);
  public void debug(String message);
  public void info(String message);
  public void warn(String message);
  public void error(String message);
}
```

Pozivom odgovarajuće metode pokušava se ispisati poruka željene razine (daje se zahtjev za ispis). U samoj konfiguraciji Loggera može se definirati koje će se sve razine poruka ispisati (TRACE, DEBUG, INFO, WARN i ERROR). Konfiguracija se definira hijerarhijski. Ako je nešto definirano na korijenu hijerarhije te se postavke nasljeđuju na djecu:

Naziv Logger	Dodijeljena razina	Efektivna razina
root	DEBUG	DEBUG
X	ništa	DEBUG
X.Y	ništa	DEBUG
X.Y.Z	ništa	DEBUG

Naziv Logger	Dodijeljena razin	Efektivna razina
Root	DEBUG	DEBUG
X	INFO	INFO
X.Y	ništa	INFO
X.Y.Z	ERROR	ERROR

Konfigurirana efektivna razina definira koji će sve pozivi metoda za ispis poruka rezultirati da se ta poruka i ispiše na neki izlaz:

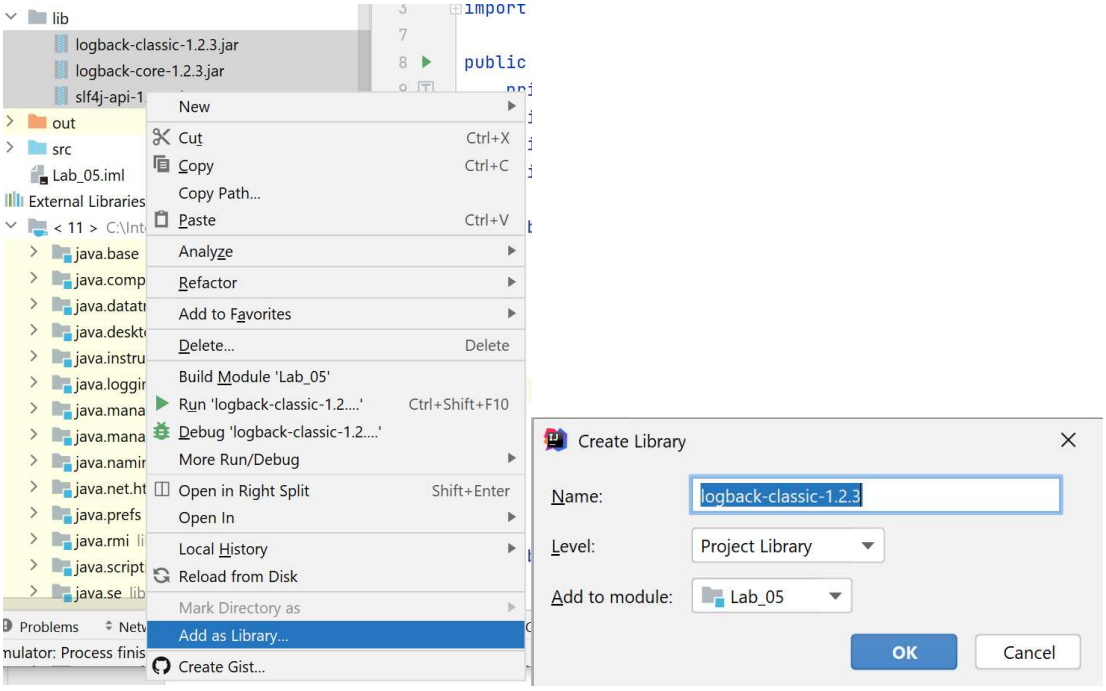
Razina zahtjeva *p*	Efektivna razina *q*					
	TRACE	**DEBUG**	**INFO**	**WARN**	**ERROR**	**OFF**
TRACE	**DA**	**NE**	**NE**	**NE**	**NE**	**NE**
DEBUG	**DA**	**DA**	**NE**	**NE**	**NE**	**NE**
INFO	**DA**	**DA**	**DA**	**NE**	**NE**	**NE**
WARN	**DA**	**DA**	**DA**	**DA**	**NE**	**NE**
ERROR	**DA**	**DA**	**DA**	**DA**	**DA**	**NE**

Appenders i Layouts/Encoders definiraju gdje se poruke ispisuju i u kojem formatu. Pomoću Appender-a se poruke mogu zapisati na konzolu, u datoteku, na socket server, bazu podataka, ... Na pojedini Logger (tj. naziv Logger-a) mogu biti vezana više Appender-a. Appender-i se isto tako konfiguriraju hijerarhijski ali se Appenderi nasljeđuju aditivno (ako nije drukčije definirano pomoću additivity zastavice):

Naziv Logger	Definirani Appender-i	Additivity zastavica	Izlazni Appender-i	Komentar
root	A1	Nije primjenjivo	A1	
x	A-x1, A-x2	true	A1, A-x1, A-x2	Appender-i do "x" i od root-a.
x.y	none	true	A1, A-x1, A-x2	Appender-i do "x" i od root-a.
x.y.z	A-xyz1	true	A1, A-x1, A-x2, A-xyz1	Appender-i do "x.y.z", "x" i od root-a.
security	A-sec	false	A-sec	Nema naslijeđenih appendera zbog zastavice addivity = `false`. Samo appender A-sec će se koristiti.
security.access	none	true	A-sec	Samo appender-i od "security" zbog additivity zastavce u "security" koja je postavljena u false

2.2.2 Korištenje i konfiguracija

Potrebno je u direktoriju projekta kreirati direktorij lib u koji ćemo smjestiti jar datoteke. Iz zip datoteke slf4j.zip potrebo je kopirati sve jar datoteke u kreirani lib direktorij. Da bi se klase u tim paketima mogle koristiti najprije se trebaju dodati na class path. U Intellij se to može postići tako da se u strukturi projekta izaberu sve jar datoteke i desnim klikom miša pokrene izbornik u kojem se treba izabrati Add as a library:



Nakon toga se mogu koristiti klase iz priloženih jar datoteka.

Logger koji se koristi u kodu je org.slf4j.Logger i kreira se putem org.slf4j.LoggerFactory-a. U svakoj klasi u kojoj želimo koristiti logiranje treba dodati:

```
private static final Logger log = LoggerFactory.getLogger(JFrameTest.class);
```

Ovakav poziv će kreirati Logger koji ima naziv isti kao i puni naziv (paket + naziv klase) klase koja je proslijeđena u metodu (hr.vsite.java.JFrameTest). Nakon toga korištenjem log varijable mogu se ispisivati poruke ili greške određene razine.

```
log.trace("Main Enter");
log.debug("Main Enter");
log.info("Main Enter");
log.warn("Main Enter");
log.error("Greška kod otvarnja datoteke", e); // e – varijabla iznimke u catch bloku;
```

Ako se žele logirati i varijable ili parametri to se može na slijedeći način. U String parametru metode se sa {} određuje mjesto na kojem se ispisuje pojedini parametar a nakon String parametra se navode jedan ili više parametara koje se ispisuju:

```
log.trace("Varijable: prva={}, druga={}, treca={}", prva, druga, treca);
```

Kako će poruke izgledati i gdje će se ispisivati je definirano konfiguracijskom datotekom koja se zove logback.xml i koja se treba nalaziti u src direktoriju projekta:

```
<?xml version="1.0" encoding="CP1250"?>
<!-- scan zastavice definira da li se u toku rada plikacije prate promjene konfiguracije
scanPeriod svakih koliko sekundi se provjerava da li je došlo do promjene datoteke -->
<configuration scan="true" scanPeriod="10 seconds">
  <!--definicija prvog appender-a. Ispisuje se na konzolu -->
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
```

```
    <encoder>
      <pattern>%d{dd.MM.yyyy. HH:mm:ss} %level [%thread] %logger{20} - %msg%n</pattern>
    </encoder>
  </appender>
  <!--definicija drugog appender-a. Ispisuje se u datoteku, novu svakog dana -->
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>chat.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>chat.log.%d{yyyy-MM-dd}</fileNamePattern>
      <!-- možemo i ograničiti veličinu svake datoteke (kada je datoteka veća
                                od limita (100MB) napravi se nova) -->
      <!--<timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
        <maxFileSize>100MB</maxFileSize>
      </timeBasedFileNamingAndTriggeringPolicy>-->
      <!-- možemo definirati i koliko maksimalno datoteka se može napraviti -->
      <!--<maxHistory>30</maxHistory>-->
    </rollingPolicy>
    <encoder>
                  <!--ispisuje se datum razina-poruke [naziv threda] naziv-loggera(max 20 zankova) - poruka-->
      <pattern>%d{dd.MM.yyyy. HH:mm:ss} %level [%thread] %logger{20} - %msg%n</pattern>
    </encoder>
    <Encoding>utf-8</Encoding>
  </appender>
  <!-- definiraju se postavke logger-a
         svaki logger čije ime počinje sa hr.vsite se ispisuje na appender-e STDOUT i FILE definirane gore
         pod level se može definirati od koje razine se zapisuju poruke; ALL – poruke svih razina

  -->
  <logger name="hr.vsite" additivity="false" level="ERROR">
    <appender-ref ref="FILE"/>
  </logger>
<logger name="hr.vsite.java.JFrameTest" additivity="false" level="ALL">
    <appender-ref ref="STDOUT"/>
</logger>

  <!--root logger ili default postavke -->
  <root level="WARN">
    <appender-ref ref="STDOUT"/>
  </root>
</configuration>
```

Zadatak: U vašem kodu trebate logirati sve ulaze (zajedno sa ulaznim parametrima) i izlaze iz vaših metoda a kod obrade greške treba logirati koja se je greška desila.
Više o konfiguraciji na http://logback.qos.ch/manual/configuration.html o appenderima na http://logback.qos.ch/manual/appenders.html a o formatu poruke na http://logback.qos.ch/manual/layouts.html