Проект по

“*Системи за Паралелна Обработка*”

На тема

“*Simple Chat Client-Server*”

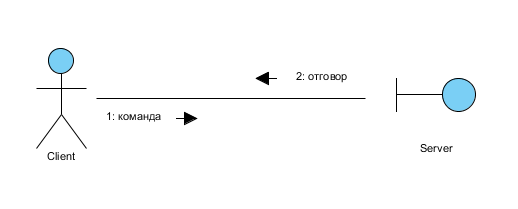
*Изготвен от  
Мартин Минчев Минчев  
Компютърни Науки – 3ти курс  
ФН: 80209*

# Описание на Задачата

Задачата е да се създаде клиент-сървър приложение за обмяна на малки текстови съобщения. Приложението трябва да се състои от две части, работещи напълно независимо една от друга – чат сървър и чат клиент. Те трябва да комуникират помежду си по специален, предварително дефиниран протокол.

## Протокол за Комуникация

Чат сървърът и чат клиентът трябва да разбират и да си комуникират чрез следния протокол.



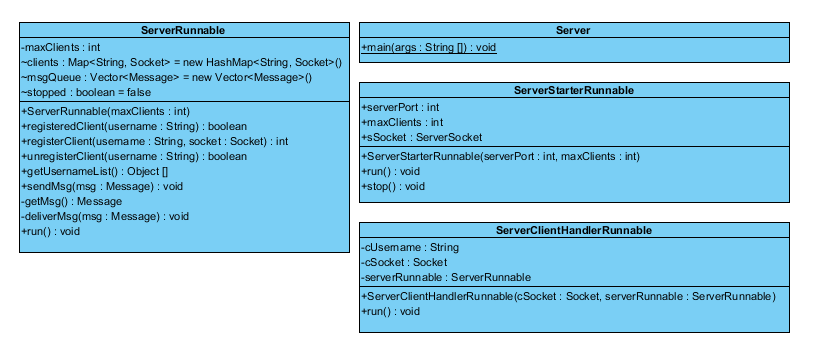
1. Действие: ***Регистриране на потребител***
   1. Команда: ***hello <username>\r\n***
      1. Отговор при успех: ***200 ok <username> successfully registerred\r\n***
      2. Отоговор при неуспех: ***100 err <username> already taken!\r\n***
2. Действие: ***Съобщение до потребител***
   1. Команда:**msg\_to <username> <single line message>\r\n**
      1. Отговор при успех: **200 ok message to <username> sent successfully.**
      2. Отоговор при неуспех: **100 err <username> does not exists!**
3. Действие: **Съобщение до всички потребители**
   1. Команда:**msg <single line message>\r\n**
      1. Отговор при успех: **200 ok message sent successfully.**
      2. Отоговор при неуспех: **100 err server error!**
4. Действие: **Списък на потребителите ползващи в момента сървъра (регистрирани)**
   1. Команда:**list\r\n**
      1. Отговор при успех: **200 ok <username1> ... <usernameN>\r\n**
      2. Отоговор при неуспех: **100 err server error!**
5. Действие: **Преустановяване на работа на потребител (напускане)**
   1. Команда:**quit\r\n**
      1. Отговор при успех: **200 ok goodbye! \r\n**
      2. Отоговор при неуспех: **100 err server error!**

# Реализация на Проекта

За реализацията на проекта беше използван езикът за програмиране Java. В основите на сървъра и клиента стоят многонишковото програмиране и сокет комуникацията. В реализацията им участват 8 класа, като всички те са обединени в един пакет, наречен javachat.

## Реализация на Сървъра

В реализацията на сървърa участват 4 основни класа. Три от тези класове, а именно ServerRunnable, ServerStarterRunnable и ServerClientHandlerRunnable имплементират Java интерфейса Runnable, което им позволява да могат да бъдат стартирани в отделни нишки на програмата. Класът Server реализира обикновено конзолно приложение.



### Стартиране на Сървъра

Класът Server притежава един единствен статичен main() метод , който играе ролята на начална точка за изпълнението на сървъра.

public static void main(String[] args) {

…

}

При стартиране на приложението, този метод може да приема параметри от командния ред чрез масива args. В случая е предвидено да приема 2 параметъра – порт, на който да работи сървъра и максимален брой клиенти, които сървъра може да обработва.

java –jar server.jar <ServerPort> <MaxClients>

В случай, че не бъдат зададени тези параметри от командния ред, те приемат стойности по подразбиране – 4444 и 5. След обработването на тези параметри, се създава и стартира в нова нишка обекта ssRunnable от класа ServerStarterRunnable.

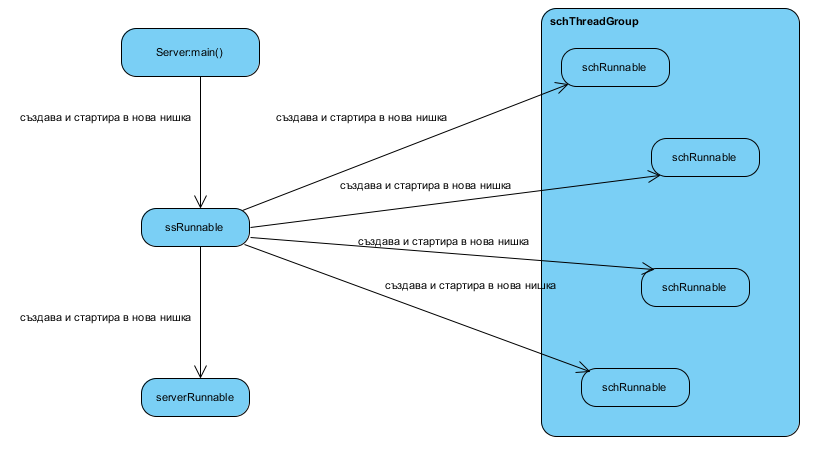
ServerStarterRunnable ssRunnable = **new** ServerStarterRunnable(serverPort, maxClients);

Thread ssThread = **new** Thread(ssRunnable);

ssThread.start();

След това main метода на Server продължава да работи, слушайки за потребителски команди от конзолата.

Веднага след започване на работата си , обекта ssRunnable създава и стартира нов обект наречен serverRunnable от класа ServerRunnable в нова нишка. Идеята на serverRunnable е да играе роля на общ склад за различни видове данни, както и да бъде главен отговорник за доставянето и разпределянето на текстовите съобщения межди клиентите на чат сървъра. След стартирането на serverRunnable в нова нишка, ssRunnable продължава своята работа като започва да слуша на определения порт за появата на нови клиенти. При всяка поява на нов клиент, той създава нов обект schRunnable от класа ServerClientHandlerRunnable и го стартира в нова нишка. Идеята на този нов обект е да поеме обработката на новия клиент. За целта при създаването на schRunnable, като параметър му се праща сокета на клиента, който трябва да обслужва. Всички нишки, предназначени да обслужват клиенти се стартират така, че да са от една ThreadGroup – schThreadGroup. Ето и диаграма, която илюстрира процеса на стартиране на сървъра.



### Изпълнение на обекта serverRunnable от класа ServerRunnable

Обектът serverRunnable играе синхронизираща роля при работата на сървъра. Той е достъпен от всички обекти, обслужващи клиенти. В него се пазят данни за това кои потребители са регистрирани в момента. През него минават всички текстови съобщения, като неговата отговорност е да ги достави правилно да адресатите. Нека да разгледаме какви атрибути и методи притежава той.

Информация за всички регистрирани в момента клиенти се пази в атрибута clients. С цел по-бързо търсене на клиенти е използвана структурата от данни – Map.

Map<String, Socket> clients = new HashMap<String, Socket>();

Всички съобщения който пристигат в serverRunnable и все още не се доставени се пазят в опашка.

Vector<Message> msgQueue = new Vector<Message>();

Този метод проверява дали даден клиент е регистриран в момента.

public synchronized boolean registeredClient(String username)

Този методи се използват за регистриране и отрегистриране на клиенти.

public synchronized int registerClient(String username, Socket socket)  
public synchronized boolean unregisterClient(String username)

Тези методи са свързани с изпращането и получаването на текстовите съобщения.

public synchronized void sendMsg(Message msg)

private synchronized Message getMsg()

private synchronized void deliverMsg(Message msg)

Всички методи, които разгледахме са дефинирани като synchronized. Това гарантира, че при евентуално състезание между нишки за достъп до тях, няма да възникнат проблеми.

Още със стартирането на обекта serverRunnable в нишката, той влиза в цикъл, който може да бъде прекъснат само ако бъде прекъсната нишката.

**while** (!Thread.*currentThread*().isInterrupted())

{

**try**

{

Message msg = getMsg();

deliverMsg(msg);

}

**catch** (InterruptedException ie)

{

//ie.printStackTrace();

Thread.*currentThread*().interrupt();

}

}

С getMsg() се извлича първото съобщение от опашката. В случай че в опашката няма съобщения в дадения момента, getMsg() блокира нишката чрез wait(). Тя се събужда, когато в опашката се добави ново съобщение, чрез методa sendMsg(), който използва notify().

deliverMsg() изпраща подаденото съобщение до неговия адресат или адресати.

### Изпълнение на обекта schRunnable от класа ServerClientHandlerRunnable

Както вече споменахме, задачата на schRunnable е да обслужва даден клиент. Веднага след стартирането на обекта се създават потоци през който да се обменят данни с клиента.

InputStream c\_in = cSocket.getInputStream();  
OutputStream c\_out = cSocket.getOutputStream();

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(c\_in));

PrintWriter out = **new** PrintWriter(c\_out);

След това изпълнението на обекта се блокира и се чака клиента да изпрати команда с която да се регистрира.

String line = in.readLine();

При получаването на командата, обекта се събужда и продължава своето изпълнение. Ако потребителското име, с което потребителя се представя не е заето, и сървърът не е достигнал максималния брой клиенти, които може да обработва, schRunnable регистрира потребителя в обекта serverRunnable.

**int** status = serverRunnable.registerClient(cUsername, cSocket);

След това изпълнението на обекта продължава в цикъл, в който се чака команда от клиента и след получаването и се отговаря по подходящ начин съобразно протокола за комуникация. Цикълът може да бъде прекъснат ако клиента изпрати команда „**quit\r\n**”, като в този случай към клиента се изпраща отговор съдържащ съобщение за довиждане. Излизането от цикъла води и до завършване на работата на нишката.

### Спиране на Сървъра

Спирането на сървъра започва, когато бъде изпратена команда „quit” от конзолата към main() метода на Server. Тогава се извиква метода stop() на обекта ssRunnable, след което се изчаква приключването на нишката, в която се изпълнява той.

**if** (ssThread.isAlive())

{

ssRunnable.stop();

ssThread.join();

}

Изпълнението на методът stop() върху ssRunnable причинява затварянето на сокета, по който обекта е блокиран. Това води до хвърлянето на изключение, което бива хванато по надолу в неговия код. Обработката на изключението се състои в изпълнението на interrupt() върху групата на нишките обслужващи клиенти и върху нишката, която изпълнява обекта serverRunnable.

schThreadGroup.interrupt();

**if** (serverThread.isAlive())

{

serverThread.interrupt();

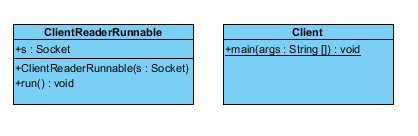
serverThread.join();

}

Това води до прекратяване работата на тези нишки. ssRunnable изчаква тяхното приключване и след това също приключва. Последен работата си завършва main() метода на Server.

## Реализация на Клиента

В реализацията на клиента участват 2 класа. Единият имплементира Java интерфейсът Runnable, а другият реализира обикновено конзолно приложение.



Изпълнението на клиента започва от main() метода на класа Client. Този метод приема 2 параметъра от командния ред – адрес и порт на сървъра, към който клиента трябва да се свърже.

java –jar client.jar <host> <port>

Ако такива не бъдат зададени, стойностите по подразбиране са – 127.0.0.1 и 4444. След обработването на подадените параметри, от потребителя се очаква да въведе потребителско име, с което да бъде регистриран на сървъра. Не се допуска потребителят да има за име символът - „\*”.

Отваря се сокет към сървъра и се създават потоци за получаване и пращане на данни от и към него.

Socket s = new Socket(clientAddress, clientPort);

OutputStream os = s.getOutputStream();

InputStream is = s.getInputStream();

PrintWriter out = new PrintWriter(new OutputStreamWriter(os));

BufferedReader in = new BufferedReader(new InputStreamReader(is));

Изпраща се команда за регистрация към сървъра и се чака отговор от него. При положителен отговор, тоест успешна регистрация, се създава обект crRunnable от класа ClientReaderRunnable, който се стартира в нова нишка.

След това main() метода продължава изпълнението си като влиза в цикъл, където чака потребителки команди от конзолата и ги изпраща към сървъра. През това време обекта crRunnable влиза в цикъл, където слуша за съобщения изпращани от сървъра към клиента и при получаване ги извежда в конзолата на потребителя.

Чрез командата “quit”, потребителя може да започне процедура по прекратяване на работата на клиента. Сървърът се уведомява за напускането на клиента, а той от своя страна връща съобщение за довиждане. Метода main() на класа Client излиза от цикъла, в който слуша за потребителски команди. Извиква interrupt() метода на нишката, в която се изпълнява обекта crRunnable, и изчаква нейното приключване. Обекта crRunnable излиза от цикъла, в който чака за съобщения от сървъра и завършва своята работа. Накрая main() метода на Client също приключва изпълнението си.

# “Тих” Решим на Работа на Сървъра

По подразбиране за всяко едно събитие, което се случва на сървъра, бива извеждано съобщение, което го описва в конзолата. За извеждане на съобщенията е използван статичния метод show(), на класа Alert. Този клас притежава, също така, и статичен булев атрибут visible, чрез който се контролира дали съобщенията подавани на метода show() да биват извеждани в конзолата или не. За да се постихне „тих” режим на работа на сървъра, на този атрибут трябва да бъде зададена стойност false.

public class Alert {

static public boolean visible = true;

static public void show(String msg)

{

if (visible)

{

System.out.println(msg);

System.out.flush();

}

}

}

Alert.show("Socket opened successfully at 127.0.0.1:" + serverPort);