درس برنامه‌سازی پیشرفته

تمرین سوم بخش دوم

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم‌سال دوم ۰۰ـ۹۹

استاد:
دکتر محمدامین فضلی

مبحث:
ترد، شبکه، جنریک

مهلت ارسال:
۳۱ خرداد ـ ساعت ۲۳:۵۹:۵۹

مسئول تمرین‌ها:
پرهام صارمی

مسئول تمرین سوم:
پرهام چاوشیان

طراحان تمرین سوم بخش دوم:
محمدطه جهانی‌نژاد، سروش جهان‌زاد، میلاد سعادت

ویراستار فنی:
فاطمه عسگری

به موارد زیر توجه کنید:

* هم‌فکری و هم‌کاری در پاسخ به تمرینات اشکالی ندارد و حتی توصیه نیز می‌شود؛ ولی پاسخ ارسالی شما باید حتما توسط خود شما نوشته شده‌باشد. در صورت هم‌فکری در مورد یک سوال، نام فرد دیگر را به‌صورت کامنت در ابتدای کد خود بنویسید.
* شما می‌توانید تمامی سوالات و ابهامات خود را در سایت کوئرا در بخش مشخص‌شده برای این تمرین بپرسید .
* مهلت ارسال تمرین تا ساعت **۲۳:۵۹:۵۹** روز **۳۱** خرداد **۱۴۰۰** است.
* به سیاست‌های تأخیر در تمارین که قبلا اعلام شده است، توجه داشته باشید.
* تست‌های این تمرین در اختیار شما قرار می‌گیرد و شما موظفید تمامی آن‌ها را پاس کنید. هم‌چنین در هنگام تحویل حضوری تمرین، کد شما به وسیله تعدادی تست مشابه سنجیده می‌شود.

۱ عباس بوعذار

علی دایی پس از اینکه نتوانست حضار جلسه و خبرنگاران را متوجه مفهوم حرفش بکند، تصمیم گرفت از تعدادی شکل و نمودار برای توضیح دقیق‌تر حرفش استفاده کند. او متوجه شد که رسم شکل و نمودار به طور دستی کار طاقت‌فرسایی است بنابراین تصمیم گرفت برنامه‌ای بنویسد که کار او را ساده‌تر کند اما هنگامی که شروع به کد زدن کرد متوجه شد که برای نوشتن این برنامه به مفهومی نیاز دارد که "نه ح داره، نه میم داره، نه دال داره، یدونه ر داره" که این مفهوم همان جنریک است. اما قسمت تلخ ماجرا اینجاست که آشنایی ایشان با جنریک به همان جمله ختم می‌شود بنابراین در نوشتن برنامه مدنظرش به مشکل خورده است. آقای دایی که نمی‌خواهد از موضعش کوتاه بیاید و می‌خواهد همه متوجه عمق مطلب بشوند به سراغ شما می‌آید و از شما می‌خواهد که به او کمک کنید. در این سوال شما باید یک صفحه نمودار را به شکل جنریک پیاده‌سازی کنید.به طور دقیق نام و تعریف داده‌ساختاری که قرار است پیاده کنید به شکل زیر است:

```
public class Board<T extends Drawable> {}
```

که Drawable یک interface به شکل زیر است:

```
public interface Drawable {
    double getPerimeter();
    double getSurface();
    int getSide() throws SideNotDefinedException;
}
```

تابع getPerimeter() : محیط شکل را برمی‌گرداند.

تابع getSurface() : مساحت محصور شده توسط شکل را باز می‌گرداند.

تابع getSide() : در صورتی که شکل منحنی نباشد تعداد اضلاع آن را برمی‌گرداند و اگر شکل منحنی باشد یک استثنا پرتاب می‌شود.

تعریف عملیات‌هایی که این داده‌ساختار باید انجام بدهد به همراه تعریف توابعی که هر عملیات را انجام می‌دهند در ادامه آورده شده است:

- یک شکل به صفحه نمودار اضافه می‌کند.

```
public void addNewShape(T shape)
```

- مجموع محیط تمام شکل‌ها را بازمی‌گرداند.

```
public double allPerimeter()
```

- مجموع مساحت تمام اشکال را بازمی‌گرداند.

```
public double allSurface()
```

- مجموع تعداد اضلاع تمام اشکالی را که ضلع برای آن‌ها تعریف می‌شود را بازمی‌گرداند.

```
public double allSide()
```

- مجموع تعداد اضلاع تمام شکل‌ها را بازمی‌گرداند.اگر شکلی وجود داشته باشد که برایش ضلع تعریف نشود یک استثنا از جنس "SideNotDefinedException" پرتاب می‌کند.

```
public double allSideException() throws SideNotDefinedException
```

- شکلی که کمترین مساحت دارد را باز می‌گرداند.

```
public T minimumSurface()
```

- اشکالی را که محیط آن‌ها بیشتر از x را است را بر اساس مساحتشان مرتب کرده و به عنوان خروجی باز می‌گرداند.

```
public ArrayList<T> sortedList(double x)
```

۲ تردپول

همانطور که می‌دانید، اجرای کارهای همزمان در جاوا با مفهومی به نام **ترد** انجام می‌شود. مشکلی که در تردها وجود دارد این است که، ساختن و نابود کردن یک ترد از سیستم حافظه و زمان نسبتا زیادی می‌برد، به همین علت هنگامی که می‌خواهیم تعدادی کار را داخل یک صف قرار دهیم و آن‌ها را به ترتیب به کمک تعداد مشخصی ترد اجرا کنیم، از سرویسی به نام ThreadPool استفاده می‌کنیم. برای حل این سوال، شما باید این سرویس را در جاوا پیاده سازی کنید. برای اینکه بیشتر با این مفهوم آشنا شوید، به این لینک مراجعه کنید.

بانکداری هوشمند

در این سوال شما باید سامانه ATM (یا خودپرداز) یک بانک را طراحی کنید. برای شروع، پروژه اولیه را که به پست مربوط به این تمرین پیوست شده است دانلود کنید. ساختار فایل های این پروژه به صورت زیر هستند:

```
src
├── Bank
│   ├── ATM.java
│   ├── Bank.java
│   ├── Card.java
│   └── Handler.java
├── Exceptions
│   ├── InvalidCardNoException.java
│   ├── InvalidCashAmountException.java
│   ├── NoCardInsertedException.java
│   ├── NoFreeAtmException.java
│   ├── NotEnoughBalanceException.java
│   └── WrongPasswordException.java
├── Results
│   ├── AccountCreatedResult.java
│   ├── BalanceMovedResult.java
│   ├── BalanceReturnedResult.java
│   ├── CardInsertedResult.java
│   ├── CardRemovedResult.java
│   ├── CashDepositedResult.java
│   ├── PasswordChangedResult.java
│   └── Result.java
└── Tasks
    ├── ChangePasswordTask.java
    ├── CreateAccountTask.java
    ├── DepositCashTask.java
    ├── GetBalanceTask.java
    ├── InsertCardTask.java
    ├── MoveBalanceTask.java
    ├── RemoveCardTask.java
    └── Task.java
```

روند کلی کار به این صورت است که در ابتدای اجرای برنامه یک instance از کلاس بانک ایجاد می‌شود. این بانک تعداد مشخصی خودپرداز دارد که هنگام ساختن بانک مشخص می‌شود. هر خودپرداز در هرلحظه می‌تواند حداکثر به یک کاربر خدمات ارائه کند، اما خودپردازها می‌توانند به صورت همزمان با هم فعالیت کنند؛ یعنی اگر پنج خودپرداز داشته باشیم، می‌توانیم هم‌زمان به پنج کاربر خدمات ارائه کنیم. کارهایی که هر خودپرداز می‌تواند انجام دهد، به عنوان یک دسته کلاس در پکیج Tasks تعریف شده‌اند و جواب‌هایی هم که هر Task برمی‌گرداند در پکیج Results تعریف شده. توضیحات بیشتر در مورد کلاس‌ها را در ادامه می‌بینید.

چند نکته

۱. در این سوال ما هیچ ورودی یا خروجی نداریم! (فکر کنم خبر خوبی براتون باشه :D)

۲. هرکدام از کلاس‌های Task یک تابع به اسم run دارند که هیچ ورودی دریافت نمی‌کند و تعدادی عملیات انجام می‌دهد. اگر انجام این عملیات با خطا مواجه شود، متناسب با ارور پیش آمده، یک Exception پرتاب می‌کند. پرت کردن این استثنا نباید منجر به کشته شدن (اصطلاحا Kill شدن) ترد شود. در نتیجه هندل کردن و return کردن جواب Task یا استثناء پرت شده به عهده شماست.

۳. جوابی که آپلود می‌کنید نباید تابع main داشته باشد. برای تست کردن کد خود می‌توانید یک تابع main بنویسید اما موقع آپلود باید این تابع را حذف کنید.

کلاس Bank

همانطور که بالاتر بیان شد، در کل فرایند اجرای برنامه فقط یک آبجکت از این کلاس ایجاد می‌شود. کانستراکتور این کلاس تعداد خودپردازهای این بانک را در ورودی دریافت می‌کند و به همان تعداد ATM ایجاد می‌کند و تا قبل از کال شدن تابع runATM، این کلاس کار دیگری انجام نمی‌دهد. برای اینکه خودپردازهای داخل این بانک قابلیت اجرای همزمان داشته باشند، باید موقع ساخته شدن این کلاس، یک ThreadPool با اندازه مناسب ایجاد کنید و تسک‌هایی که در ادامه داده می‌شوند را به این کلاس بدهید تا هرکدام در تردی جداگانه اجرا شوند. توابع داخل کلاس Bank به شرح زیر هستند:

- Bank(int ATMCount) : کانستراکتور کلاس که تعداد خودپردازهای این بانک را ورودی دریافت می‌کند. تضمین می‌شود که در تست کیس‌ها این عدد، یک عدد مثبت است.
- runATM(ArrayList<Task> tasks, Handler handler) : این تابع تسک‌ها را دریافت می‌کند، اگر یک ATM خالی (تردی که مشغول اجرای دستوری نباشد) وجود داشت، دستورات را توسط آن ترد اجرا می‌کند. در غیر این صورت، دستور وارد صف تردپول می‌شود و تا زمان اجرا شدن در آن صف باقی می‌ماند. پس از پایان یافتن کار ترد، باید تابع ()done از آبجکت Handler داده شده کال شود؛ با صدا کردن این تابع کد شما اعلام می‌کند که نتیجه اجرای این Task آماده می‌باشد.

نکته مهم: این تابع نباید به صورت blocking پیاده سازی شود، یعنی برنامه تا قبل از اجرای کامل ترد و آماده شدن نتیجه، در این تابع باقی نمی‌ماند.

کلاس ATM

این کلاس معادل مدل خودپرداز می‌باشد. همواره یک instance از این کلاس باید به Task در حال اجرا داده شود.

کلاس Card

این کلاس نشان دهنده یک کارت بانکی می‌باشد. هر کارت یک شماره کارت، یک پسورد و یک موجودی دارد.

پکیج Tasks

کلاس های داخل پکیج Tasks به صورت کامل پیاده سازی شده‌اند و شما نباید در آنها تغییری ایجاد کنید. هر کدام از این کلاس‌ها یک کانستراکتور دارند که فیلدهای مورد نیاز را دریافت می‌کند. یک تابع run هم در این کلاس‌ها تعریف شده که هیچ ورودی دریافت نمی‌کند و خروجی آن یک آبجکت از کلاس Result می‌باشد. اگر در حین پردازش و اجرای این تابع خطایی بوجود بیاید، یک Exception توسط این کلاس پرتاب می‌شود. شما باید داخل تردی که تعریف می‌کنید، تابع run تسک های داده شده را کال کنید، اکسپشن احتمالی پرتاب شده را catch کنید و یا جواب ریترن شده را بگیرید و داخل یک ArrayList بریزید و در نهایت ArrayList تولید شده را ریترن کنید. توجه کنید که این ArrayList باید یک ArrayList<Object> و متشکل از Result ها و Exception های تولید شده باشد.

کاری که شما باید انجام دهید

ابتدا باید پروژه را به طور کامل دانلود کنید. وظیفه اصلی که بر عهده شماست، پیاده کردن کلاس ThreadPool می‌باشد. این کلاس باید در کانستراکتور خود یک عدد به عنوان ورودی دریافت کند، و به همان تعداد ترد بسازد. همچنین باید تابعی داشته باشد که یک مجموعه دستور دریافت کند و آن‌ها را داخل یک صف قرار دهد و به محض اینکه کار یک ترد به پایان رسید، دستور بعدی را از صف برداشته و اجرا کند.

نکته: ترد ها باید همزمان اجرا شوند، اما ترتیب خروجی‌ها باید به همان ترتیب ورودی‌ها باشد. همچنین صفی که تعریف می‌کنید باید سینکرونایزد (Synchronized) باشد تا وقتی که همزمان چند ترد از آن داده می‌خوانند، مشکلی بوجود نیاید.

آنچه باید آپلود کنید

تکمیل شده‌ی همه فایل‌هایی که دانلود کرده‌اید + کلاس‌هایی که ایجاد کرده‌اید را آپلود کنید. دقت کنید که تابع main نباید در برنامه وجود داشته باشد.

راهنمایی

صرفا جهت روشن‌تر شدن اینکه چگونه باید کلاس ThreadPool را پیاده سازی کنید: می‌توانید یک property از جنس Queue در این کلاس تعریف کنید که مجموعه تسک‌های داده شده را نگهداری کند:

```
Queue<ArrayList<Task>> queue = new Queue<>();
```

موقع ساخته شدن هر ترد هم، می‌توانید در تابع run که برای آن ترد تعریف می‌کنید، یک حلقه بی‌نهایت قرار دهید که تا زمانی که دیتا در این صف وجود دارد، اطلاعات را بخواند و آن‌ها را اجرا کند:

```
new Thread(new Runnable() {
    @Override
  public void run() {
        while (true) {
            //fetch tasks and execute...
  }
    }
}).start();
```

۳ پیام‌رسان محرمانه

مامور انگلیش در آخرین ماموریت خود به قلعه‌ای در دوردست‌ها رفته است. از آنجا که این ماموریت بسیار محرمانه است، او به پیام‌رسان‌های رایج اعتماد نمی‌کند و می‌خواهد که پیام‌رسانی مخصوص داشته باشد. به همین دلیل او از شما می‌خواهد که با دانش سرشاری که در درس برنامه‌نویسی پیشرفته به دست آورده‌اید، پیام‌رسانی نوین و مناسب نیازهای او طراحی کنید.

نمونه (instance) های مختلف این برنامه پس از اجرا به صورت یک به یک (peer-to-peer) در ارتباط خواهند بود و سرور جداگانه‌ای وجود نخواهد داشت. دقت کنید که تمامی این ارتباطات بین نمونه‌های مختلف هستند و باید بر بستر شبکه و با استفاده از برنامه‌نویسی سوکتی انجام شوند.

توجه: بخش‌هایی که بین دو علامت **\+** و **+** قرار دارند، امتیازی هستند.

هر دستور با یک کلیدواژه‌ی اصلی و تعدادی تگ مشخص می‌شود. **\+**تگ‌ها با ـــ شروع می‌شوند و می‌توانند به هر ترتیبی بیایند.**+** بعضی تگ‌ها ممکن است آرگومان دریافت کنند که در این صورت مقدار آن آرگومان بلافاصله بعد از نام آن تگ نوشته خواهد شد. در تمام مراحل اگر عملیات مدنظر با موفقیت انجام شود، عبارت زیر نمایش داده می‌شود.

```
success
```

در غیر این صورت پیغام گفته شده در همان بخش باید نمایش داده شود و در صورتی که در بخشی پیامی برای خطا مشخص نشده بود می‌توانید پیام زیر را نمایش دهید.

```
an error occurred
```

ورود به برنامه

با شروع به کار برنامه، پیش از هر کار نیاز داریم که یک کاربر بسازیم و نام خود را مشخص کنیم تا در پیام‌ها نام فرستنده معلوم باشد. هر کاربر یک نام کاربری و یک رمز عبور دارد. برای انجام کارهای مربوط به کاربر، از دستور userconfig استفاده می‌کنیم.

ساخت کاربر

ساختن کاربر با تگ create انجام می‌شود و دستور به صورت زیر خواهد بود.

```
userconfig --create --username <username> --password <password>
```

اگر کاربری با نام کاربری گفته شده از پیش وجود داشت یا شامل کاراکترهایی غیر از حروف الفبا و اعداد و ـ و _ بود باید پیغام زیر نمایش داده شود.

```
this username is unavailable
```

ورود کاربر

برای ورود به عنوان یک کاربر، از تگ login استفاده می‌کنیم و دستور به صورت زیر خواهد بود.

```
userconfig --login --username <username> --password <password>
```

اگر کاربری با این نام وجود نداشت پیغام زیر نمایش داده می‌شود.

```
user not found
```

اگر رمز عبور نادرست بود پیغام زیر نمایش داده خواهد شد.

```
incorrect password
```

توجه: تا زمانی که کاربر وارد نشده باشد نمی‌تواند از دیگر دستورات که در ادامه می‌آیند استفاده کند و در صورت تلاش برای این کار، عبارت زیر نمایش داده می‌شود.

```
you must login to access this feature
```

ـ دریافت پیام

آغار فرآیند دریافت

در هنگام آغاز کار، برنامه یک شماره درگاه (port) دریافت می‌کند که عملیات listen با آن انجام خواهد شد و پیام‌های دریافتی به این درگاه خواهند آمد. برای انجام کارهای مربوط به این درگاه، از دستور portconfig استفاده می‌کنیم. این درگاه به صورت زیر و با تگ listen مشخص می‌شود.

```
portconfig --listen --port <port-number>
```

اگر پیش از اجرای این دستور یک درگاه مشخص شده بود، پیغام زیر باید چاپ شود.

```
the port is already set
```

تغییر درگاه

اگر این دستور با تگ rebind نوشته شود، باید درگاه قبلی بسته شود و درگاه جدید روی شماره‌ی داده شده باز شود. در این صورت دستور به شکل زیر خواهد بود.

```
portconfig --listen --port <port-number> --rebind
```

بستن درگاه

بستن درگاه با تگ close انجام می‌شود و یک دستور نمونه برای آن به شکل زیر است.

```
portconfig --close --port <port-number>
```

در صورتی که درگاه مشخص شده از پیش باز باشد انجام دستور موفقیت‌آمیز خواهد بود و در غیر این صورت، پیام زیر چاپ می‌شود:

```
the port you specified was not open
```

پس از اینکه این درگاه مشخص شد، برنامه آماده است تا پیام‌های افراد دیگر را که با نمونه‌های دیگر این برنامه فرستاده می‌شوند دریافت کند.

ـ ۱+ ذخیره آدرس مخاطبان

برای سادگی بیشتر در زمان ارسال پیام، می خواهیم آدرس (host , port) مربوط به کاربرانی که با آن‌ها تعامل داریم را ذخیره کنیم تا با داشتن نام کاربری به تنهایی نیز بتوانیم به آن‌ها پیام ارسال کنیم. فرآیند ذخیره‌ی این آدرس‌ها به دو صورت خودکار و دستی قابل انجام است.

ذخیره خودکار آدرس

هنگامی که پیامی را دریافت می‌کنیم، باید برنامه میزبان و درگاه فرستنده را تشخیص دهد و آن‌ها را با نام کاربری او ذخیره کند. پس از آن نام کاربری او در زمان ارسال پیام از این سمت، نمایش‌دهنده‌ی این آدرس خواهد بود.

ذخیره دستی آدرس

می‌توانیم با دستور contactconfig با این آدرس‌ها کار کنیم. برای مربوط کردن یک نام کاربری به عنوان مخاطب با یک آدرس از تگ link استفاده می‌کنیم.

```
contactconfig --link --username <username> --host <host> --port <port-number>
```

توجه: در هر دو حالت بدون توجه به آدرس قبلی مربوط به آن نام کاربری، آدرس تغییر خواهد کرد و به‌روزرسانی خواهد شد. به عبارت دیگر، این آدرس بازنویسی (overwrite) می‌شود.

نکته: دقت کنید که این آدرس ممکن است به مرور زمان تغییر کند و اگر هنگام فرستادن پیام این آدرس پیدا نشد، همانگونه که جلوتر هم به آن اشاره می‌شود باید پیام زیر نمایش داده شود.

```
no contact with such username was found
```

۱+

ـ ارسال پیام

برای ارسال پیام، باید آدرس میزبان (host) مقصد و شماره درگاه (port) آن را داشته باشیم. ارسال پیام در دو حالت متمرکز و غیرمتمرکز ممکن است که در ادامه با آن‌ها آشنا می‌شویم. پیام ارسالی از طریق شبکه باید به فرمت زیر باشد؛ یک رشته که در آن نام کاربری فرستنده در ابتدا قرار می‌گیرد و پس از آن بعد از یک علامت «خط جدید» پیام اصلی می‌آید.

```
<username> -> <message>
```

حالت غیرمتمرکز

در این حالت با دستور send به صورت زیر پیام به مقصد مورد نظر ارسال می‌شود.

```
send --message "<message>" --port <port-number> --host <host>
```

+۱ ارسال پیام به یک مخاطب نیز با تگ username ممکن است.

```
send --message "<message>" --username <username>
```

اگر مخاطب با نام کاربری گفته شده تعریف نشده بود پیام زیر چاپ شود.

```
no contact with such username was found
```

+۱
اگر پیغام به دلیل دیگری با موفقیت ارسال نشد پیام زیر نمایش داده شود.

```
could not send message
```

حالت متمرکز

در صورتی که بخواهیم چند پیام را به یک مقصد بفرستیم، می‌توانیم به جای تکرار تگ‌های port و host در دستور send روی مقصدمان «تمرکز» کنیم. توجه کنید که این تمرکز می‌تواند روی یک میزبان خاص یا یک میزبان و درگاه خاص باشد. با این کار تا زمانی که این تمرکز را متوقف کنیم، پیام‌هایی که در دستور ارسالشان درگاه و میزبان مشخص نشوند از مقصد موردنظر در تمرکز استفاده خواهند کرد. با دستور focus و تگ start می‌توانیم

به دو صورت زیر این کار را انجام دهیم.

تمرکز روی یک میزبان

```
focus --start --host <host>
```

پس از انجام این دستور، باید بتوانیم دستوری مانند دستور زیر را با موفقیت انجام دهیم.

```
send --port <port-number> --message "<message>"
```

تمرکز روی یک درگاه پس از تمرکز روی میزبان

اگر از پیش روی یک میزبان تمرکز کرده باشیم می‌توانیم تمرکز را به یک درگاه در همان میزبان محدود کنیم.

```
focus --port <port-number>
```

در صورتی که تمرکز روی هیچ میزبانی وجود نداشت پیغام زیر باید چاپ شود.

```
you must focus on a host before using this command
```

تمرکز روی یک میزبان و درگاه (یک برنامه هدف)

```
focus --start --host <host> --port <port-number>
```

پس از اجرای این دستور، باید بتوانیم دستوری مانند دستور زیر را با موفقیت انجام دهیم.

```
fsend --message "<message>"
```

۱+ تمرکز روی یک نام کاربری

با استفاده از تگ username می‌توانیم روی آدرس مربوط به یک نام کاربری تمرکز کنیم.

```
focus --start --username <username>
```

اگر مخاطب با نام کاربری گفته شده تعریف نشده بود پیام زیر چاپ شود.

```
no contact with such username was found
```

۱+
پایان حالت متمرکز

با اجرای دستور focus به همراه تگ stop می‌توانیم از این حالت خارج شویم.

```
focus --stop
```

توجه: دقت کنید که اگر حالت متمرکز در جریان باشد و دوباره با تگ start بخواهیم روی یک هدف جدید تمرکز کنیم، هدف قبلی از حالت تمرکز خارج شده و فراموش می‌شود و هدف جدید جایگزین آن خواهد شد.

ـ نمایش داده‌ها

در نهایت، باید بتوانیم اطلاعاتی از جمله پیام‌های دریافت شده را ببینیم. برای این کارها از دستور show استفاده می‌کنیم. در هر بخش، هر مورد در خط جدیدی نمایش داده می‌شود. در هر کدام از بخش‌ها اگر هیچ موردی برای نمایش وجود نداشت عبارت زیر نمایش داده می‌شود.

```
no item is available
```

۱+ نمایش مخاطبان

با تگ contacts می‌توانیم نام کاربری و آدرس مخاطبان ثبت شده را مشاهده کنیم.

```
show --contacts
```

در خروجی این دستور، هر مورد باید به صورت زیر باشد.

```
<username> -> <host>:<port-number>
```

نمایش یک مخاطب خاص

با تگ contact می‌توانیم آدرس ذخیره شده مربوط به یک نام کاربری را مشاهده کنیم.

```
show --contact <username>
```

خروجی این دستور در صورت وجود مخاطب به فرمت زیر خواهد بود.

```
<host>:<port-number>
```

در غیر این صورت پیام زیر چاپ می‌شود.

```
no contact with such username was found
```

+۱

نمایش فرستندگان

با تگ senders می‌توانیم نام کاربری کسانی که تاکنون از آن‌ها پیام دریافت کرده‌ایم را مشاهده کنیم.

```
show --senders
```

نمایش پیام‌ها

برای دیدن پیام‌ها، از تگ messages استفاده می‌کنیم.

```
show --messages
```

این دستور تمام پیام‌های دریافت شده را به ترتیب دریافت و به همان فرمت دریافت شده نمایش می‌دهد.

نمایش تعداد

می‌توانیم تگ count را در ترکیب با هر کدام از دو تگ senders و messages استفاده کنیم که به ترتیب نتیجه‌ی نمایش داده شده برابر تعداد فرستندگان مختلف و تعداد تمام پیام‌های دریافت شده خواهد بود.

```
show --count --senders

show --count --messages
```

۱+ نمایش داده‌های مربوط به یک فرستنده خاص

می‌توانیم با تگ from به پیام‌های مربوط به یک فرستنده‌ی خاص دسترسی پیدا کنیم. این تگ به دو صورت زیر قابل استفاده خواهد بود.

پیام‌های یک فرستنده خاص

```
show --messages --from <username>
```

این دستور تمام پیام‌های دریافت شده از فرستنده با نام کاربری مدنظر را به ترتیب دریافت چاپ می‌کند. دقت کنید که در این روش، تنها پیام اصلی باید چاپ شود و بخش اول رشته‌ی دریافت شده که شامل نام کاربری است نباید در خروجی بیاید.

تعداد پیام‌های یک فرستنده خاص

```
show --count --messages --from <username>
```

در این حالت، تعداد پیام‌هایی که فرستنده با این نام کاربری ارسال کرده نمایش داده می‌شود.

۱+

یادداشت: پیشنهاد می‌شود که این دستورات را با استفاده از annotation پیاده‌سازی کنید. البته این کار برای این تمرین اجباری نیست.