0. KOMPILACJA

```
$ javac messanger/Server.java
$ javac messanger/Contact.java

   URUCHOMIENIE
$ java messanger.Server
$ java messanger.Client &
```

UWAGA! hasło musi zawierać:
- znaki specjalne
- cyfre
- litere mała
- litere dużą
testowane wyrażeniem regularnym

1. BAZA DANYCH - SQLite

Program (Server) używa bazy danych SQL.

Aby uzyskać dostęp do bazy danych SQLite z programu napisanego w JAVA należy
w Unix/Mac shell wykonać następującą komendę:
1.) używając BASHa

```
$ export CLASSPATH=$CLASSPATH:/java/classes:/Users/michzio/
Developer/ProgramowanieSieciowe/cw6/sqlite-jdbc-3.7.2.jar
```

Żeby zobaczyć zmienne środowiskowe CLASSPATH można wykonać

```
@ echo $CLASSPATH
```

Folder (pakiet) ./model zawiera pliki klas ORM (Mapowanie obiektowo relacyjne)
Message, Contact, ContactList oraz plik DataBase.

Message reprezentuje w formie obiektowej tablicę message bazy danych SQL
- metody do zapisywania wiadomości do bazy danych
- aktualizowanie wiadomości w bazie danych
- usuwanie wiadomości z bazy danych
- pobieranie wiadomości po message_id
- usuwanie wiadomości dla konkretnego odbiorcy wiadomości
- pobieranie wiadomości nieprzeczytanych (timestamp read_date == NULL) dla konkretnego użytkownika
- usuwanie wiadomości dla konkretnego nadawcy
- wybieranie zarchiwizowanych wiadomości w komunikacji pomiędzy dwoma konkretnymi użytkownikami

Tablica w bazie danych SQL:
message

```
*********************************************************
* message_id INTEGER PRIMARY KEY AUTOINCREMENT          *
* receiver_id INTEGER                                   *
```

```
* sender_id INTEGER                                          *
* message TEXT                                               *
* created_date TEXT “YYYY-MM-dd HH:MM:ss.SSS”                *
* read_date TEXT “YYYY-MM-dd HH:MM:ss.SSS”                   *
*************************************************************
```

Contact - to obiekt reprezentujący wpis do tablicy contact w bazie danych

```
*************************************************************
* contact_id INTEGER PRIMARY_KEY AUTOINCREMENT               *
* nickname VARCHAR(255)                                      *
* sha1password CHARACTER(40)                                 *
* first_name VARCHAR(255)                                    *
* last_name VARCHAR(255)                                     *
*************************************************************
```

Klasa Contact umożliwia:
- wstawianie nowego kontaktu (np. przy rejestracji)
- aktualizowanie danych kontaktu
- pobieranie kontaktu o konkretnym contact_id
- usuwanie kontaktu z bazy danych
- wbieranie kontaktu dla konkretnego nicka
- autoryzacje poprzez wybieranie rekordu dla nickname AND sha1password
- wybieranie kontaktów które pasują do konkretnego nickname, first_name, last_name z użyciem LIKE i val% -> wyszukiwania kontaktów

ContactList - klasa reprezentująca tablicę contact_list w SQL
- reprezentuje wpisy na liście kontaktów
- relacje pomiędzy kontaktami

```
*************************************************************
* list_owner_id INTEGER (foreign key)                        *
* contact_id INTEGER  (foreign key)                          *
* PRIMARY KEY(list_owner_id, contact_id)                     *
*************************************************************/
```

Obiekt ContactList umożliwia:
- wstawienie kontaktu na listę kontaktów
- usunięcie kontaktu z listy kontaktów
- wybranie listy kontaktów dla konkretnego list_owner_id
  z użyciem INNER JOINa z tablicą Contact
- usunięcie listy kontaktów dla konkretnego list_owner_id
- usunięcie wpisów z list kontaktów dla konkretnego contact_id
  np. gdy usuwanie konta

DataBase to obiekt który korzystając ze sterownika org.sqlite.JDBC umożliwia nawiazanie połaczenia z bazą danych SQL
- jeżeli w bazie danych nie ma tablic to tworzy stosowane tablice
  CREATE IF NOT EXISTS

KLIENT oraz SERVER znajdują się w pakiecie ./messanger

SERVER
- plik Server.java

1) tworzy obiekt DataBase()
2) pokier obiekt Connection i zapisuje go w zmiennej statycznej
3) tworzy ServerSocket
4) alokuje mapy (client_id => socket), (client_id => outputstream)
    oraz mapę (client_id, status), status np. widoczny, dostepny, zaraz wracam
5) w pętli akceptuje przychodzące połączenia ServerSocket.accept()
6) dla nowego Socket'u połaczenia tworzy obiekt Server (implementuje Runnable)
   i uruchamia nowy wątek.
7) każdy program Klienta to jedno gniazdo na serwerze
    (jeden wątek do obsługi klienta)
8) konstruktor pobiera strumienie ObjectOutpuStream i ObjectInputStream
7) funkcja run w pętli while(true) odbiera pokier przesyłana za pomoca Socketów
    są to obiekty Packet<T>

Packet<T> to obiekt przesyłanego pakietu przez sieć
T - typ opakowanego w pakiecie obiektu np. Message, Integer, String, Contact,
     List<Contact>, Boolean
- posiada metode command() okraślającą rodzaj przesyłanego rozakazu
  pozwala okreslić jaki typ obiektu T towarzyszy pakietowi
- object() pobiera towarzyszacy pakietowi obiekt

8) funkcja run() w switch(packet.command()) używając stałych zdefiniowanych
   w klasie Packet<T> rozpoznaje rozaj pakietu (polecenia) i w zależności od
   tego polecenia przekazuje pakiet do obsługi odpowiedniej metodzie obsługi
   np. signUpNewAccount(Packet<Contact> packet)
       handleIsNickOccupiedRequest(Packet<String> packet)


KLIENT

Główny plik programu to Client.java
- tworzenie okna programu korzystając z javax.swing.*
- klasa Client rozszerza JFrame i implementuje Listenery eventów
- posiada mape (receiver_id => okienko rozmowy)  oraz (contact_id => Contact)
- Client w kostrukorze alokuje  własnie te mapy oraz
  tworzy Socket (nawiazujący połączenie z serwerem)
- uzyskanie ObjectOutputStream i ObjectInputStream
  przez strumienie sieciowe obiektowe wysyłane są obiekty Packet<T>
  w które opakowane są obiekty typu T + komenda identyfikujaca jakie

typu jest T
  i jaki jest cel wysłanego pakietu
  Przesyłane obiekty muszą implementować interfejs Serializable i podlegaja
  serializacji i deserializacji.
  Np. w przypadku obiektu Message, Contact, użytko słówka transient dla
      atrybutu Connection żeby nie przesyłać obiektu połaczenia do SQL
- alerty w postaci popupów są tworzone z użyciem JOptionPane
- w konstruktorze pobierany jest z Preferences (lokalne persystowanie danych)
  client_id jeżeli go nie było to pojawia się okienko logowania, rejestracji
- po uwierzytelnieniu wczytywana jest lista kontaktów
  metoda loadContactList -> zapytanie CONTACT_LIST_REQUEST
  np. new Packet<Integer>( CONTACT_LIST_REQUEST, clientId);
  i taki pakiet jest wysyłany a w odpowiedzi dostajemy
  Packet< Map<Integer, Contact> > z komenda CONTACT_LIST_RESPONSE
- analogicznie odbywa się każda inna komunikacja
- po uwierzytelnieniu (zalogowaniu, rejestracji) uzyskaniu clientId i pobraniu
  listy kontaktów uruchamiany jest wątek gdzie w funkcji run()
  podobnie jak na serwerze w pętli odbierane są obiekty Packet<T>
- ogólnie mechaniz działa w ten sposób, że w wyniku eventu np klikniecie
  przycisku, wpisania nicku w wyszukiwarce, ENTER wysyłajace wiadomość
  tworzony jest Packet<T> z zapytaniem i potrzebnymi danymi do jego realizacji
  i wysyłany jest on przez oos.writeObject(packet) na serwer z metody obsługi
  tego zdarzenia lub metody pomocniczej wywołanej przez metode obsługi zdarzenia
- następnie serwer odpowiada po obsłużeniu zapytania przesłaniem pakietu
  Packet<T> z odpowiedzia która odczytuje wątek klienta używając ois.readObject()
  I następnei w zależności od typu Packet.command() w switch przekierowuje
  ten pakiet do obsługi do odpowiedniej funkcji pomocniczej.
- nowe okna w swing tworzymy poprzez kontrukcje obiektów rozszerzajacyh
  JFrame lub JDialog (w konstrukotrze jest tam setVisible(true).

- lista kontaktów to JList<Contact>
   gdzi jest customizowane renderowanie elementów listy
   obiekt ContactCellRenderer  gdzie Contact => JPanel z JLabelami
- okienka logowania, wyszukiwania kontatków to JDialog
- okienko rozmowy to JFrame z JList<Message> i customizowanym
   renderowaniem elementów listy MessageCellRenderer gdzie Message => JPanel
- funkcja obsługująca odbieranie wiadomości sprawza w HashMapie czy
   okienko dla kontaktu o danym Id jest juz otwarte, jeżeli nie to

tworzy
   takie okienko rozmowy wpw dodaje wiadomość do istniejacego okienka
- komunikacja pomięzy okienkami przy użyciu INTERFEJS OkienkoListener
  na okienku metoda addOkienkoListener()
  i później zdarzenia tego okienka są delegowane listener.eventHappaned()
  Client implementuje interfejsy OkienkoListener i metody eventHappened()

Pozostałe pliki:
AddContacDialog - okienko szukania znajomych (kontaktów)
ClientMessageFrame - okienko rozmowy
SignInDialog - okienko rejestracji/logowania
Packet - klasa z pakietem i stałymi komend przesyłanych przez sieć
AuthorizationCompletedListener - interferes komunikcja okienko rejestracji -> okno główne programu
ContactAddedListener -> interfejs komunikacja okienko wyszukiwania kontaktu ->
okno główne programu
Statu -> typ wyliczeniowy statusów dostępności: online, offline, invisible, idle
ArchiveDialog -> okienko z archiwum rozmów z danym kontaktem
ArchiveListener -> komunikacja okienko archiwum rozmów -> okno głowne programu