|  |  |
| --- | --- |
| **Programowanie obiektowe Java**  ***PROJEKT*** | |
| **Politechnika Świętokrzyska** | |
| ***Temat:*** | System do zarządzania zespołem |
| ***Autor:*** | Jakub Francuz |
| ***Grupa:*** | 2ID12B |

**1. Cel projektu:**

Celem projektu było stworzenie aplikacji graficznej do komunikacji między członkami zespołu ‘Team Manager’ z wykorzystaniem klient-serwer (localhost) oraz biblioteki graficznej Swing. Całość została napisana w języku Java wraz z praktykami programowania obiektowego.

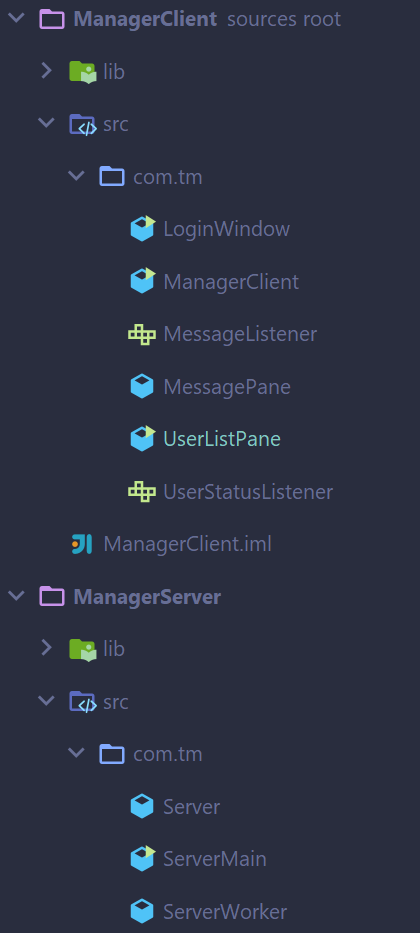
Program przygotowany jest aby działać zarówno jako CLI (command line interface) jak i GUI (graphic user interface). Powodem jest paradygmat zastosowany przy projektowaniu aplikacji – ‘dziel i zwyciężaj’. Najpierw okodowane zostały podstawowe funkcjonalności i prototypowane za pomocą linii poleceń. CLI zostało dla możliwości dalszego rozwijania aplikacji, a to po pierwsze pomogłoby w testowaniu kolejnych funkcjonalności, a także umożliwiło implementację tych bardziej złożonych ( z natury CLI zapewnia dużo większą swobodę i efektywność interakcji z programem).

Postępy projektowe miały być dokumentowane za pomocą commitów na platformie w chmurze wspierającej system rozproszonej kontroli wersji GIT. Postawiłem tutaj na bardziej popularny serwis Github. Treści commitów (jak i komentarzy w kodzie) są w języku angielskim. Umożliwia to bardziej elastyczne zaznajomienie z terminami z IT, gdzie próżno szukać polskich semantycznych odpowiedników. Link do repozytorium to:

<https://github.com/jqbFrnzs/javaPJ>

Jako środowisko programistyczne obrałem Intellij IDEA, które ma na rynku najbardziej przychylne opinie co do intuicyjności przy pracy nad projektami w języku JAVA.

**2. Struktura projektu:**



Aplikacja podzielona jest na 2 moduły: *ManagerClient* i *ManagerServer*.

Oba moduły należą do paczek com.tm ( tm – TeamManager )

Jak nazwa wskazuje, jeden zawiera komponenty dla strony klienta, drugi dla serwera aplikacji.

ManagerServer:

Klasa *ServerMain* – główny ‘entry point’ dla części serwerowej aplikacji

Klasa *ServerWorker* – implementacje funkcjonalności i logiki dla serwera

Klasa *Server* – umożliwia śledzenie i zarządzanie połączeniami z serwerem

Klasa *Constants* – zawiera stałą: numer portu, z której korzystają 3 klasy

ManagerClient:

Klasa *ManagerClient* – podstawowa klasa wejściowa dla części klienta

Klasa *LoginWindow* – zajmuje się GUI okna logowania

Klasa *UserListPane* – przedstawia graficznie listę użytkowników

Klasa *MessagePane* – okno do pisania wiadomości i komunikacji end-to-end

Interfejs *MessageListener* – metoda abstrakcyjna dla MessagePane

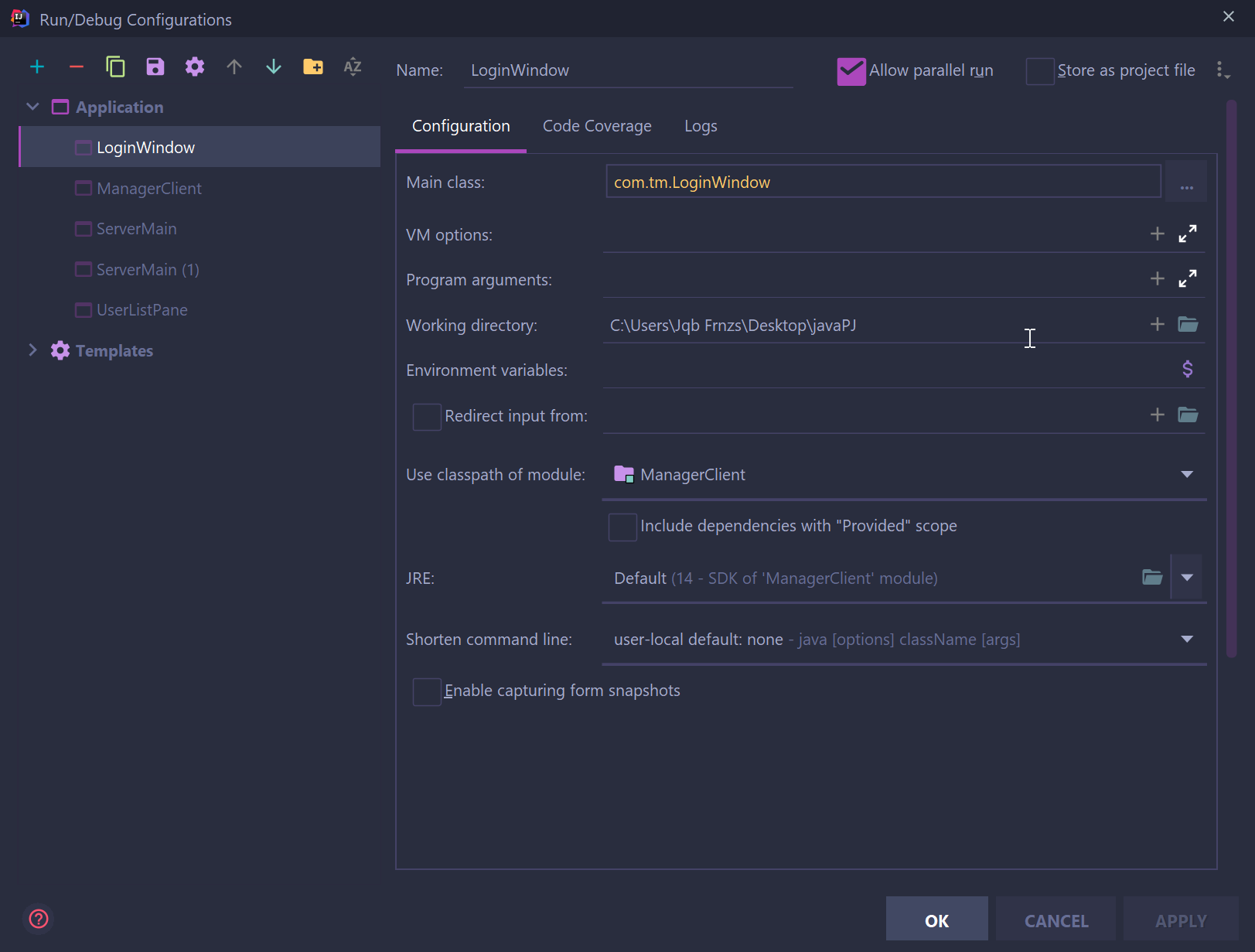
Interfejs *UserStatusListener* – metody abstrakcyjne dla UserListPane

**3. Działanie programu:**

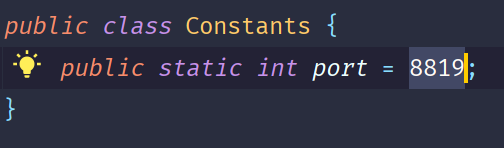
**1.** Przed uruchomieniem program trzeba upewnić się, że jest włączona opcja uruchomienia współbieżnego dla klasy LoginWindow, aby można było uruchomić wiele instancji logowania dla localhosta.

Dla Intellij IDEA:

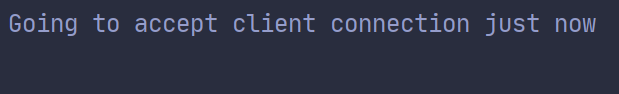
*Zakładka* ***Run -> Edit Configurations…***



***3.*** Określamy numer portu serwera, który znajduje się w klasie **Constants** (domyślnie 8819):

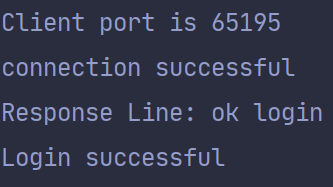


***3.*** *Następnie uruchamiamy klasę* ***ServerMain****, powinien pojawić się w konsoli dla klasy komunikat:*



*Właśnie została uruchomiona instancja serwera.*

**4.** Uruchamiamy klasę kliencką **ManagerClient**, w konsoli tej klasy ukazuje się:



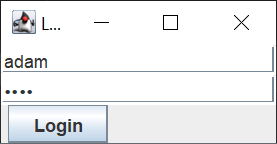
W klasie **ServerMain** za to komunikat:



*Instancja klienta działa, jej entry point (main) loguje automatycznie użytkownika* ***guest****. Konsola dla tej klasy wyświetla dalsze informacje dla kolejnych zalogowanych użytkowników. Jesteśmy gotowi na logowanie manualne poszczególnych użytkowników.*

**LOGOWANIE GUI**

**5a.** Uruchamiamy klasę **LoginWindow,** pojawia się okno logowania:



wpisujemy jedno z zestawów danych (zakodowane na sztywno, aktualnie brak podpiętej bazy danych):

**1 )**

**Login:** adam

**Hasło:** adam

**2 )**

**Login:** jqb

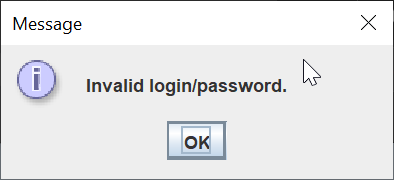
**Hasło:** jqb

**3 )**

**Login:** tomek

**Hasło:** tomek

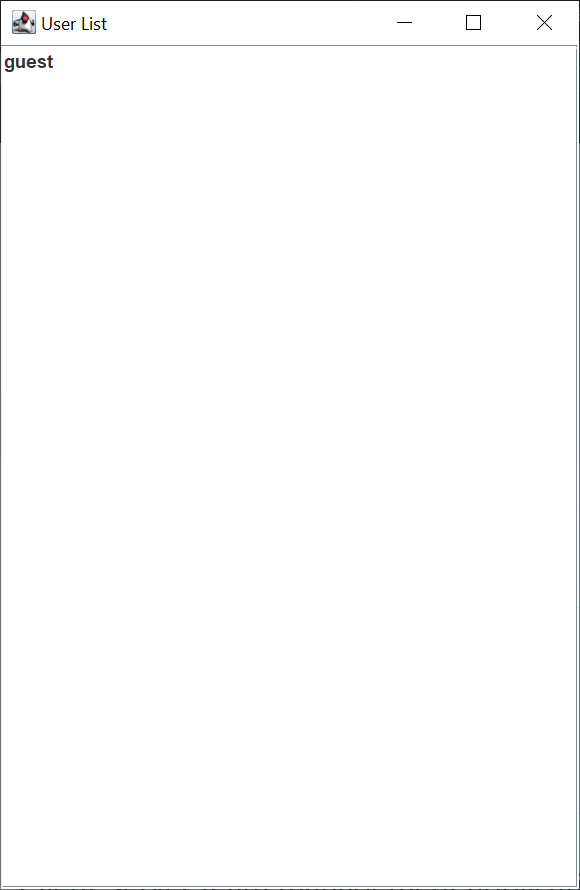
W przypadku wprowadzenia niepoprawnych danych wyświetla się baner:



Oraz komunikat w konsoli klasy **LoginWindow**:



W przypadku poprawnego logowania pojawia się lista zalogowanych użytkowników:



W konsoli **ServerMain**:



W konsoli **ManagerClient**:



**6a.** Wysyłanie wiadomości:



W celu wysłania wiadomości dwukrotnie klikamy na login użytkownika z poziomu okna **‘User List’**,

Pojawia się okno z etykietą **‘Message: nazwaUżytkownika’.** Wiadomość wpisujemy w dolnym pasku (input field), a wysyłamy za pomocą wciśnięcia klawisza **ENTER.**

**7a.** Aby się wylogować z konta danego użytkownika należy nacisnąć domyślny przycisk przeznaczony do zamykania okna **‘User List’** ( krzyżyk )

Wtedy w konsoli **ServerMain** pojawia się komunikat:



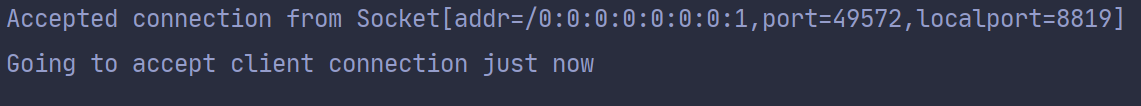
**LOGOWANIE CLI**

**5b.** Uruchamiamy dowolny emulator konsoli (np. powershell), a następnie za pomocą telnetu logujemy się na localhosta za pomocą wprowadzonego portu (domyślnie 8819):



Naciskamy ENTER

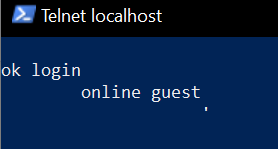
W konsoli **ServerMain** pojawia się komunikat:



**6b.** Logowanie opiera się o wpisanie słowa kluczowego **login**, następnie spacja, nazwaUżytkownika, spacja i hasłoUżytkownika, np.:

**login** jqb jqb

Przy poprawnym zalogowaniu w emulatorze wyświetla się:



Wyświatlają się powiadomienia kto jest online (tutaj *guest* z instancji początkowej **ManagerClient**)

W konsoli **ServerMain**:



W konsoli **ManagerClient**:

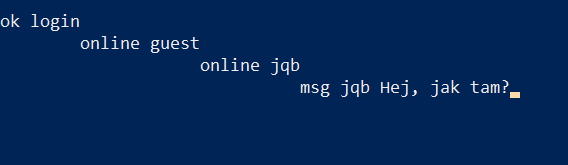


**7b.** Wysyłanie wiadomości do konkretnego użytkownika:

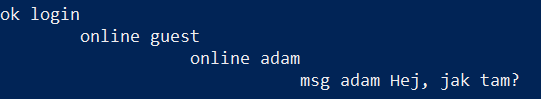
Składnia opiera się o słowo kluczowe: **msg**

Instrukcja jest cięta na 3 tokeny: komenda (tutaj **msg**), potem adresat, a na końcu treść.

Wysłanie od użytkownika **adam** do **jqb**:

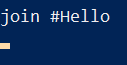


Odebranie powyższej wiadomości od **adama** przez **jqb**:



**8b.** Dołączanie do pokoju, aby wysyłać wiadomość rozgłoszeniową (broadcast message) do każdego użytkownika, który też dołączył do tego pokoju:

Słowo kluczowe **join** oraz znak **#** przed nazwą pokoju (bez spacji pomiędzy) :

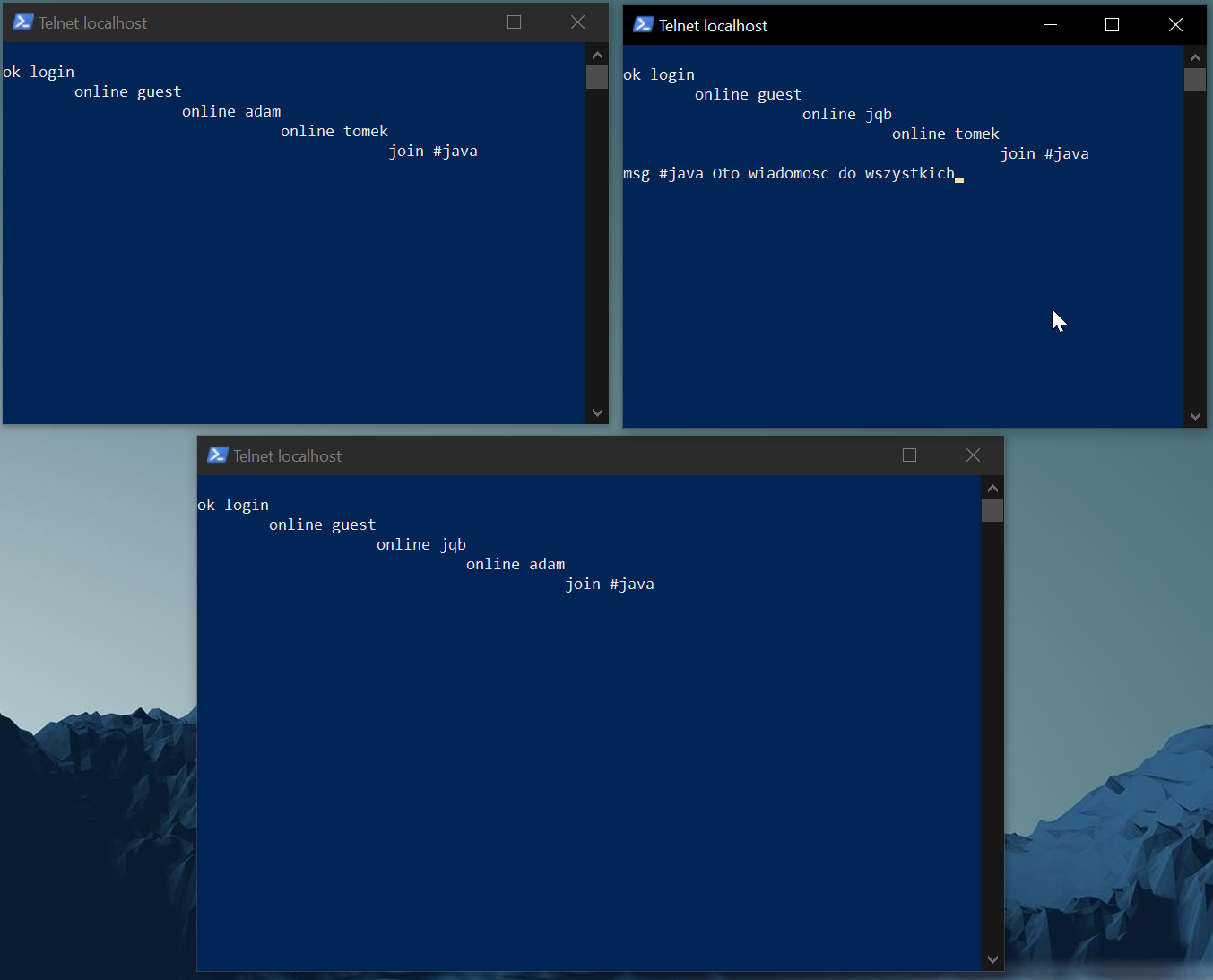


Pokoje nie istnieją wcześniej, są tworzone w czasie rzeczywistym przez użytkowników. Gdy jeden użytkownik dołączy do pokoju, a on nie istnieje, zostaje automatycznie utworzony.

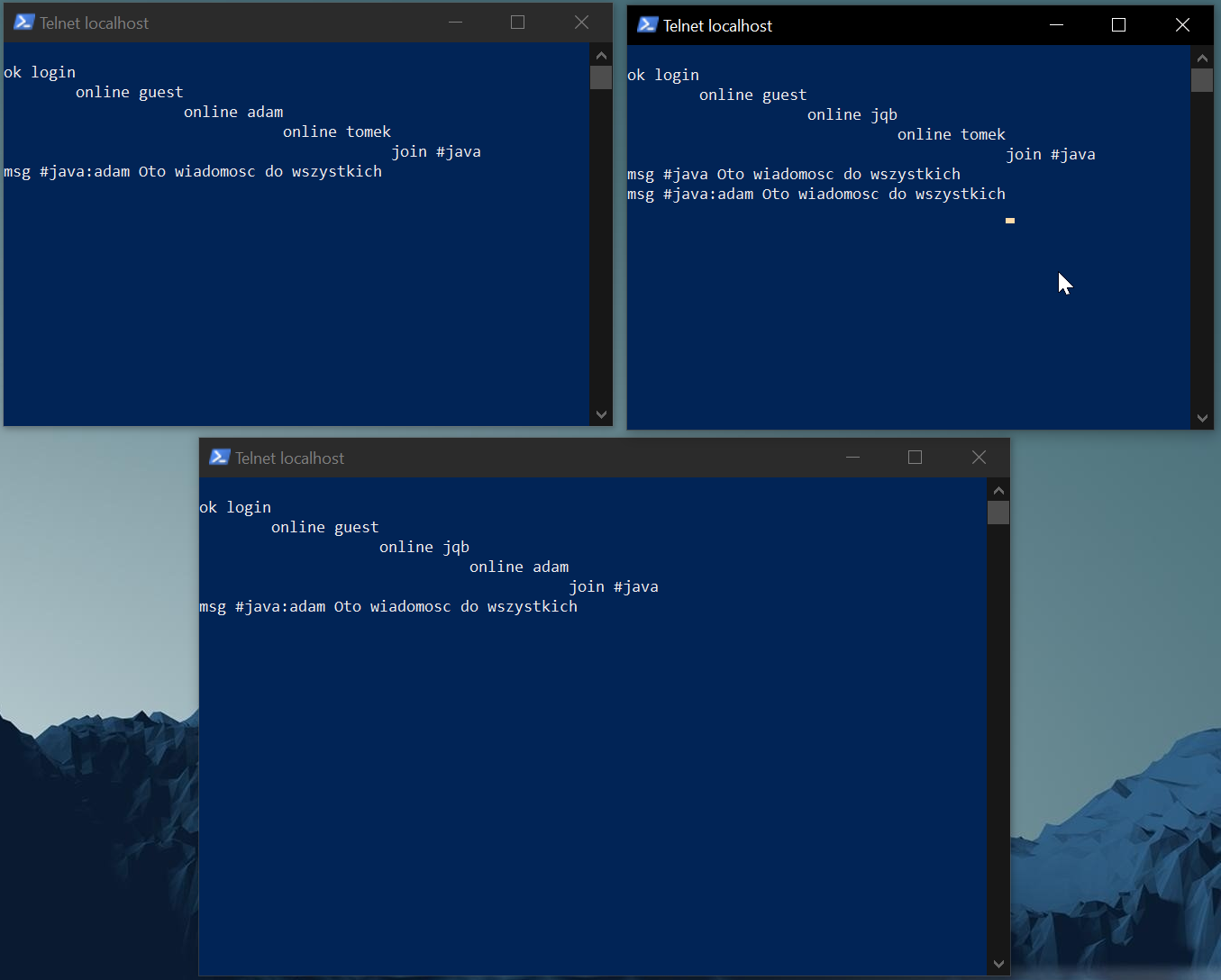
**9b.** Wysyłanie wiadomości do wszystkich członków pokoju:

*Składnia komendy:* ***msg******#****nazwaPokoju treść wiadomości*

Przed wysłaniem:



Po wysłaniu:



**10b.** Opuszczanie pokoju przez użytkownika:

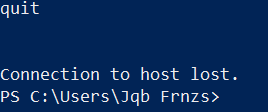
*Składnia komendy:* ***leave******#****nazwaPokoju*



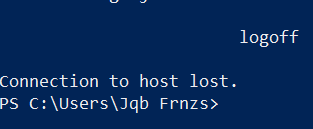
*Po opuszczeniu pokoju przez użytkownika, wiadomości do tego pokoju wysyłane, nie będą się mu wyświetlać.*

**10b.** Wylogowanie się poszczególnych użytkowników:

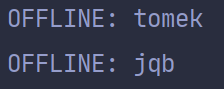
Wykorzystywane są 2 komendy: **quit**



lub **logoff**



Po wylogowaniu w konsoli **ManagerClient** wyświetla się komunikat:



**4. Analiza kodu:**

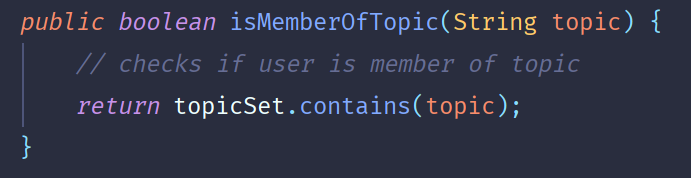
**Klasa ServerWorker:**



Tworzony jest strumień wejścia i wyjścia dla klienta. Następnie za pomocą BufferReadera jest czytany strumień wejścia. Następnie czytana jest linia z konsoli. Dzielona na tokeny i za pomocą instrukcji warunkowych dokonywana jest weryfikacja poprawności komend takich jak **msg, login, join, leave, quit i logoff.** Pierwszym tokenem tokens[0] jest zawsze komenda np. login. Maksymalna ilość tokenów dla wprowadzanej z CLI instrukcji to 3 ( 3 słowa ) np. *msg jqb Czesc.*



*Obsługuje komendę leave, usuwa pokój (topic) z instancji użytkownika, jeśli go opuścił.*



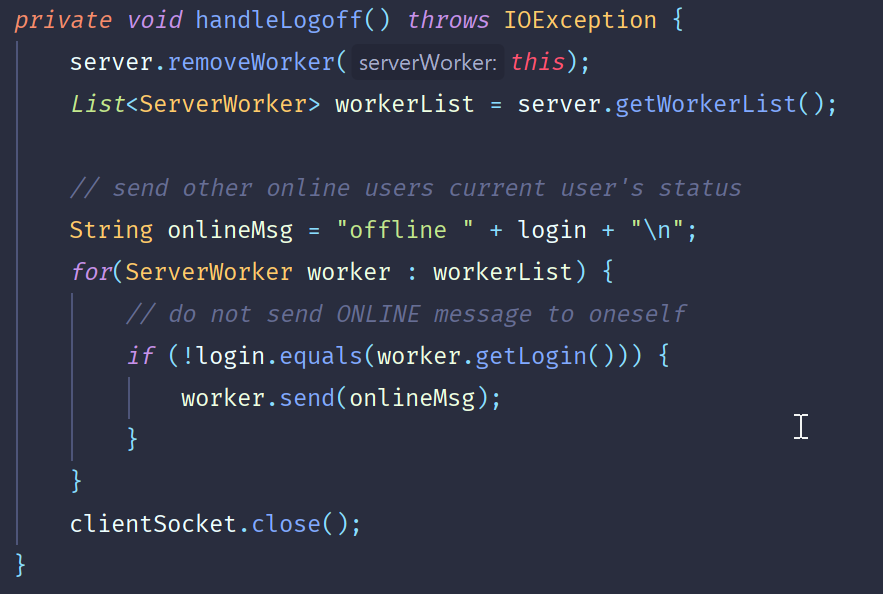
*Sprawdza czy użytkownik jest członkiem danego pokoju, tak aby wysyłać wiadomość rozgłoszeniową do poprawnej grupy docelowej.*



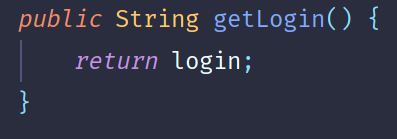
Obsługuje komendę join, dołącza pokój do instancji użytkownika.



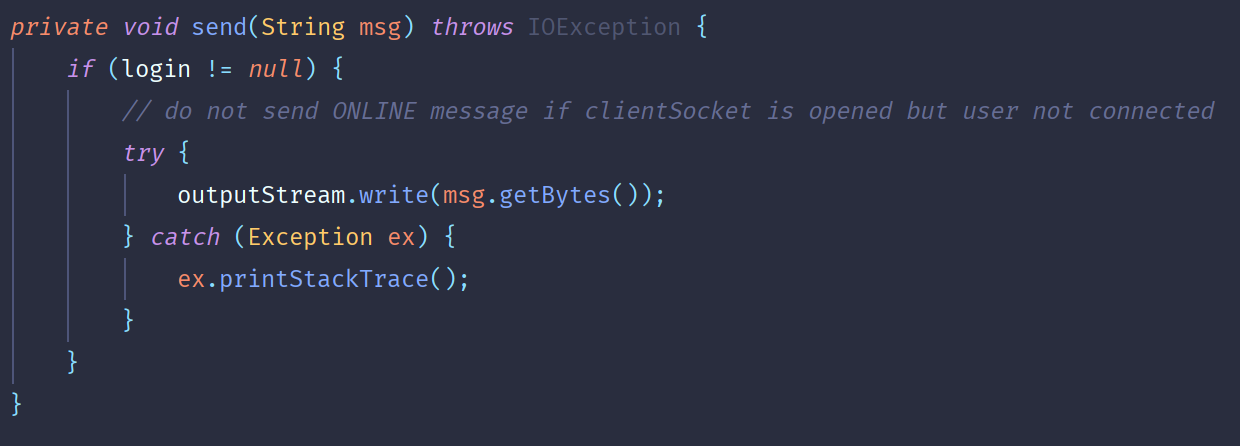
Obsługuję komendę **msg**, tokens[1] to adresat, tokens[2] to treść wiadomości. Sprawdza czy adresat ma w pierwszym znaku **‘#’**, jeśli tak, jest to wiadomośc rozgłoszeniowa do pokoju (topic). W pętli sprawdzana jest cała lista zalogowanych użytkowników w poszukiwaniu tych z podanym w komendzie pokojem – do nich wysyłana jest wiadomość rozgłoszeniowa, w przeciwnym przypadku wysyłana jest zwyczajna, bezpośrednia wiadomość do konkretnego użytkownika za pomocą zwykłej składni **msg nazwaUżytkownika treść**.



Obsługuje komendę **logoff**, usuwa użytkownika, który ją wpisał z listy zalogowanych użytkowników.

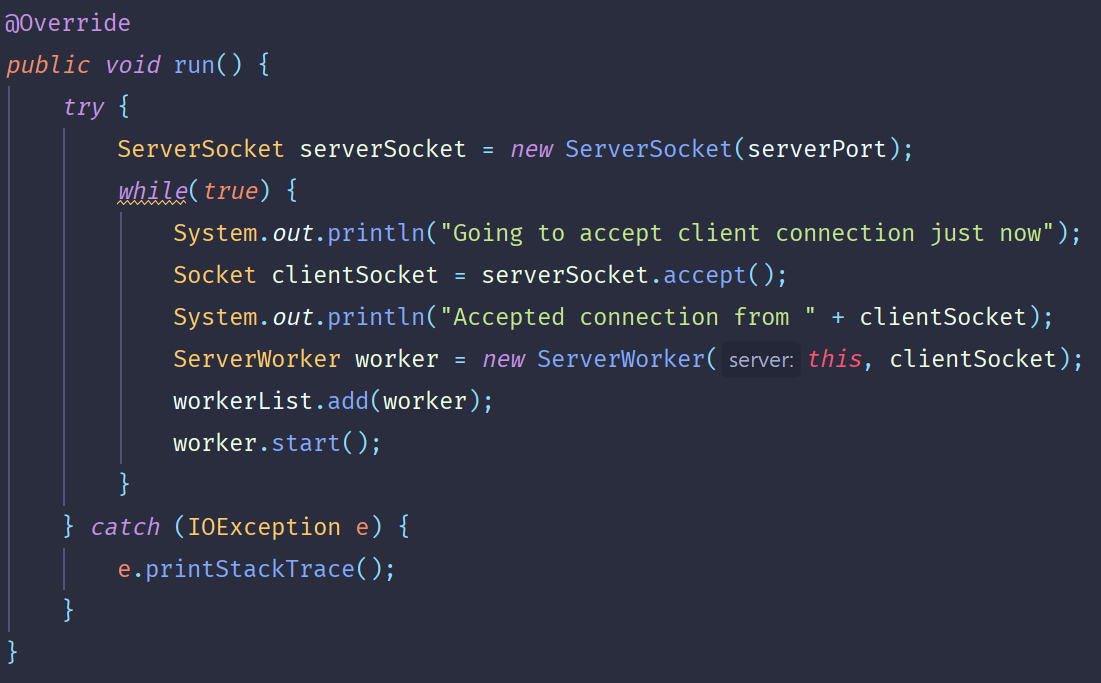


Zwraca login pojedynczego użytkownika (instancji).

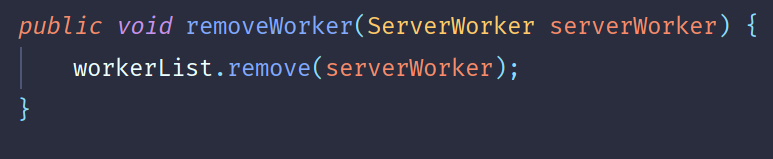


Obsługuję wysyłanie wiadomości do użytkowników np. komunikat o statusie ONLINE innego użytkownika.

**Klasa Server:**

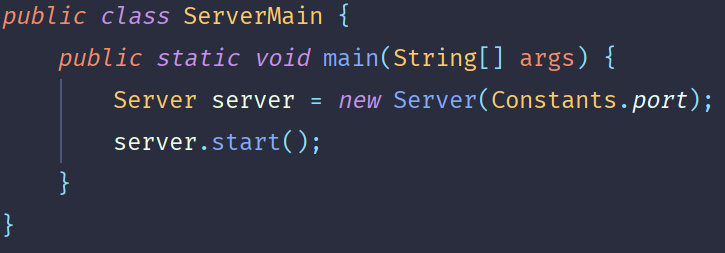


Obsługuje ServerSocket, wysyła komunikaty odnośnie połączeń klienckich z serwerem. Dodaje użytkownika do listy ONLINE.



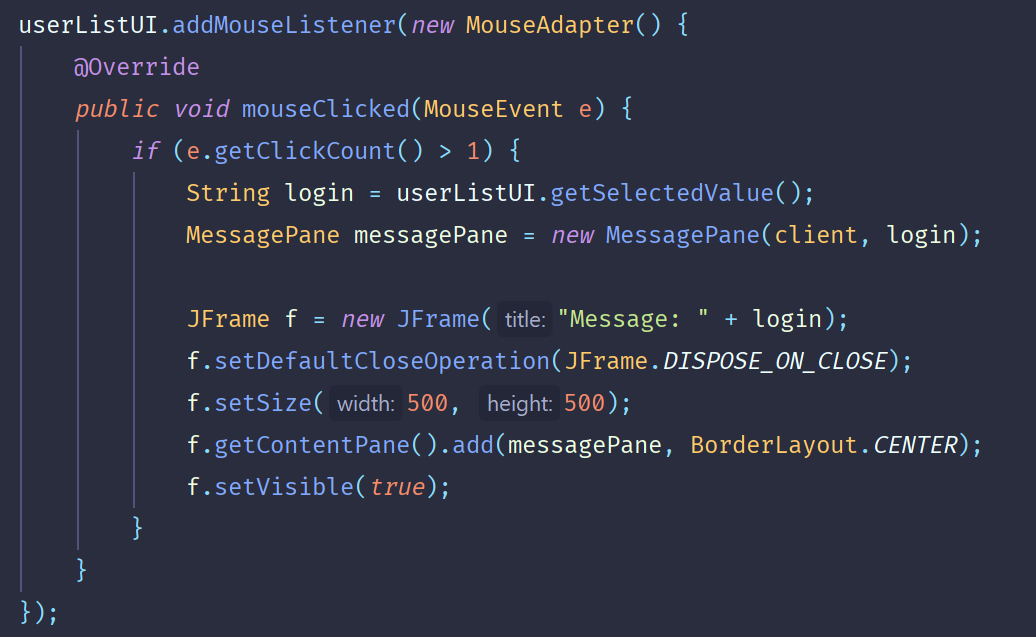
Usuwa użytkownika z listy ONLINE.

**Klasa Server:**

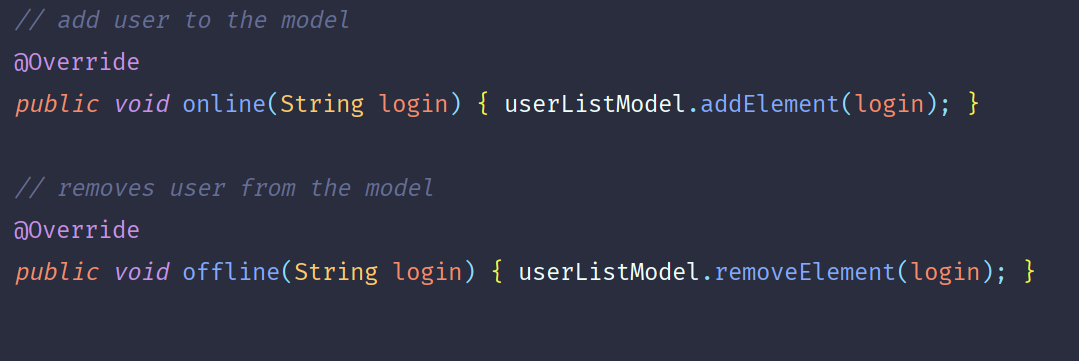


Główny entry-point programu. Uruchamia całą infrastrukturę serwera. Korzysta z klasy ze stałą posiadającą numer lokalnego portu dla serwera.

**Klasa UserListPane:**



Obsługuje podwójne kliknięcie na nazwę użytkownika w liście użytkowników GUI, po którym wyświetla się okno z pisaniem wiadomości.

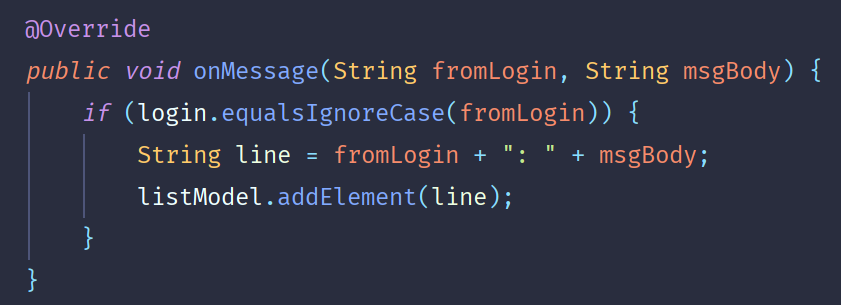


Funkcje obsługują dodawanie/usuwanie z listy użytkowników, po tym jak oni się wylogują.

**Klasa MessagePane:**

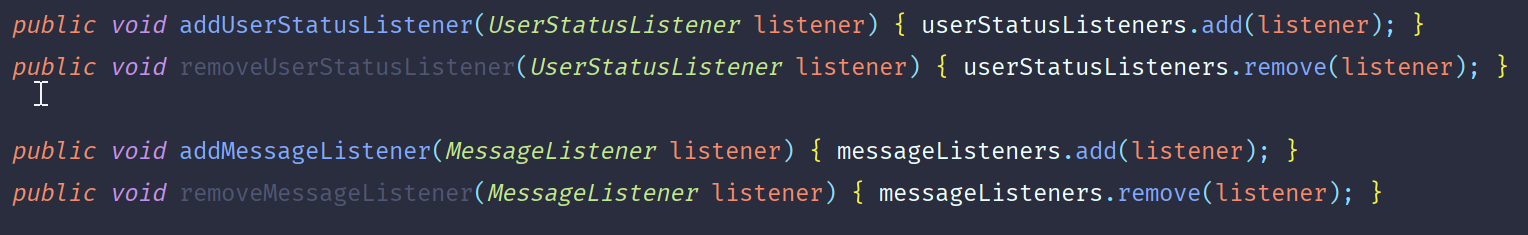


Nasłuchuje na pole z wysyłaniem wiadomości, następnie wyświetla wysłaną wiadomość w polu rozmowy oraz zeruje input-field, aby można było wprowadzić kolejną wiadomość.



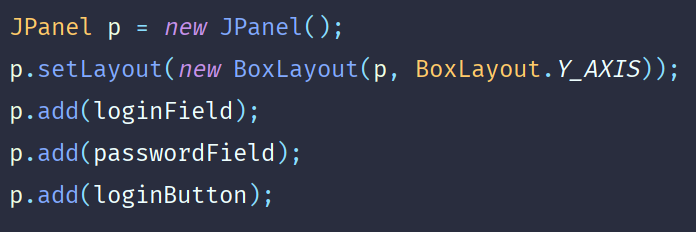
Obsługuje wyświetlanie wiadomości od nadawcy do odbiorcy w panelu rozmowy.

**Klasa ManagerClient:**



Dodawanie/usuwanie nasłuchu statusu/wiadomości.

**Klasa LoginWindow**



Stworzenie panelu logowania, dodanie przycisku, inputów dla loginu i hasła.

**5. Wnioski i uwagi**

- projekt pozwolił utrwalić fundamenty programowania obiektowego takie jak: enkapsulacja, dziedziczenie czy polimorfizm oraz zrozumieć istotę połączenia klient-serwer

- miałem okazję bliżej zapoznać się, z systemem kontroli wersji GIT oraz ekosystemem Github, które są niezastąpione przy pracy nad bardziej rozbudowanymi projektami

- commity GITa są szczególnie przydatne przy powrocie do pracy nad projektem po dłuższej przerwie, dlatego powinny być precyzyjne i nieobszerne

- Swing okazał się bardzo dobrą opcją jako biblioteka graficzna do prostego zastosowania