Laboratorium 6

Celem ćwiczenia jest budowa prostych aplikacji komunikujących się między sobą za pomocą połączeń sieciowych opartych na gniazdach klasy `Socket` i `ServerSocket`. Obsługa komunikacji powinna odbywać się w dodatkowych wątkach działających współbieżnie z głównym oknem aplikacji.

Program ćwiczenia:

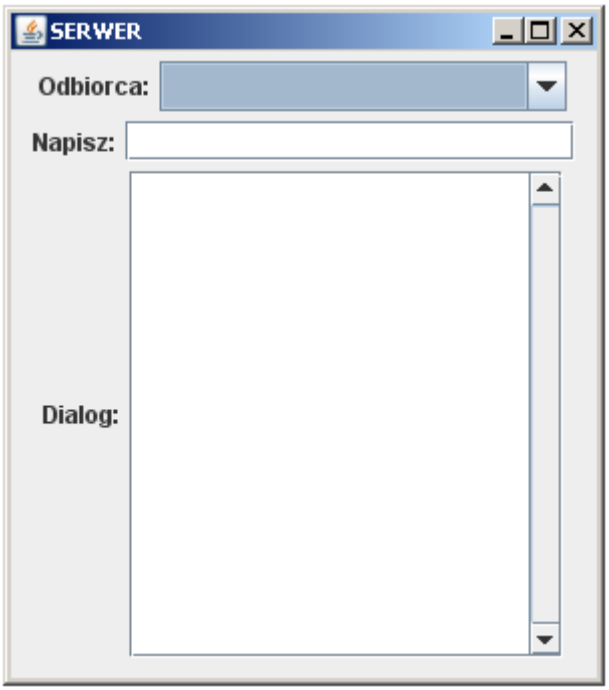
- tworzenie okna graficznego i uruchamianie wątków do obsługi wymiany komunikatów,
- tworzenie gniazda `ServerSocket`, które będzie czekać na połączenia sieciowe od innych aplikacji,
- akceptowanie połączeń sieciowych i uruchamianie dodatkowych wątków do obsługi połączeń przychodzących realizowanych za pomocą gniazda `Socket`,
- wymiana komunikatów za pomocą strumieni `InputStream` oraz `OutputStream`. połączonych z gniazdem `Socket`

Programy przykładowe

Proszę zapoznać się z poniższymi programami przykładowymi: `Serwer.java`, `Klient.java`, `Tester.java`, które ilustrują sposób realizacji komunikacji sieciowej za pomocą gniazd `ServerSocket` i `Socket`.

Program `Serwer.java` oczekuje na żądania utworzenia połączenia sieciowego wysyłanego przez aplikacje klientów. Po odebraniu żądania od innej aplikacji tworzone jest połączenie sieciowe umożliwiające przesyłanie komunikatów pomiędzy aplikacją serwera i klienta. Komunikacja między klientem i serwerem może być zakończona po wysłaniu komunikatu o treści „exit”.

Obiekt klasy `Serwer` tworzy okno graficzne przedstawione na poniższym rysunku.



Pole „Odbiorca:” zawiera rozwijaną listę klientów z którymi nawiązane jest połączenie sieciowe. W polu „Napisz:” należy wpisywać komunikat, który ma być wysłany do klienta wybranego w polu „Odbiorca:”. W oknie „Dialog:” wyświetlane są komunikaty przesyłane do i od klientów.

Po utworzeniu okna graficznego serwera tworzony jest dodatkowy wątek do obsługi połączeń sieciowych. W tym wątku (metoda `run()`) tworzone jest gniazdo `serwer` z klasy `ServerSocket`, które oczekuje na połączenia przychodzące od innych aplikacji. Dla każdego zaakceptowanego połączenia tworzony jest nowy obiekt z klasy `WatekKlienta`, do którego przekazywane jest gniazdo sieciowe `socket`. Obiekt `WatekKlienta` uruchamia nowy wątek, w którym będzie realizowana komunikacja sieciowa. W tym wątku otwierane są strumienie `InputStream` oraz `OutputStream` przyłączone do gniazda `socket`. Następnie w pętli `while` odczytywane są komunikaty przychodzące od klienta, które są wyświetlane w polu `textArea` (okno „Dialog:") w oknie graficznym serwera.

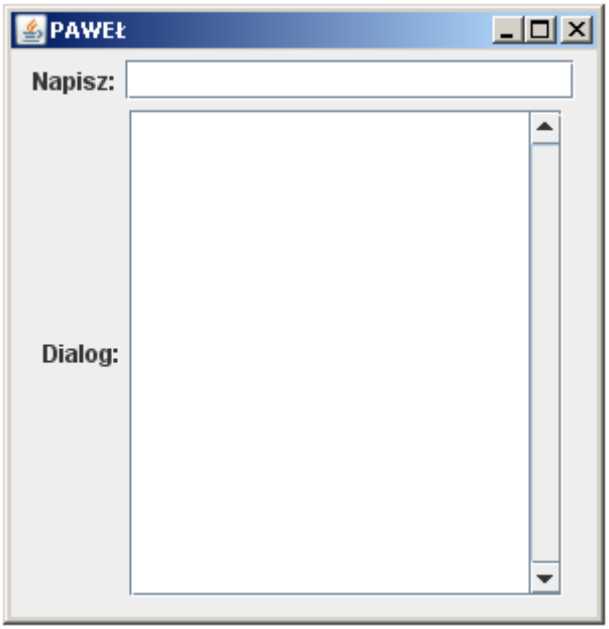
Metoda `actionPerformed` obsługuje zdarzenia generowane po wpisaniu tekstu komunikatu w polu `message` (pole „Napisz:"). Metoda ta odczytuje z listy `menuKlient` (pole „Odbiorca:") wątek obsługujący połączenie aktualnie wybranego klienta, a następnie wysyła tekst komunikatu do klienta oraz wypisuje ten tekst w polu `textArea` (okno „Dialog:").

UWAGA:

Program `Serwer.java` musi być uruchomiany jako pierwszy. Można uruchomić tylko jedną kopię tego programu. Próba jednoczesnego uruchomienia kilku kopii tego programu na tym samym komputerze nie powiedzie się, bo nie może być kilku aplikacji, które będą nasłuchiwać połączeń przychodzących na tym samym porcie.

Program `Klient.java` wysyła do serwera żądania utworzenia nowego połączenia sieciowego. Po nawiązaniu połączenia sieciowego możliwe jest przesyłanie komunikatów pomiędzy aplikacją serwera i klienta. Komunikacja między klientem i serwerem może być zakończona po wysłaniu komunikatu o treści „exit".

Klasa `Klient` tworzy okno graficzne przedstawione na poniższym rysunku.



W polu „Napisz:" należy wpisywać komunikat, który ma być wysłany do serwera. W oknie „Dialog:" wyświetlane są komunikaty przesyłane do i od serwera.

Po utworzeniu okna graficznego klienta tworzony jest dodatkowy wątek do obsługi połączeń sieciowych. W tym wątku (metoda `run()`) tworzone jest gniazdo `socket` z klasy `Socket`, a następnie otwierane są strumienie `InputStream` oraz `OutputStream` przyłączone do tego gniazda. Następnie w pętli `while` odczytywane są komunikaty

przychodzące od serwera, które są wyświetlane w polu `textArea` (okno „Dialog:") w oknie graficznym klienta.

Metoda `actionPerformed` obsługuje zdarzenia generowane po wpisaniu tekstu komunikatu w polu `message` (pole „Napisz:"). Metoda ta wysyła tekst komunikatu do strumienia `output` połączonego z gniazdem `socket` obsługującm połączenie sieciowe z serwerem, a następnie wypisuje ten tekst w polu `textArea` (okno „Dialog:"). Jeśli komunikat miał treść „exit" to zamykane są strumienie `input`, `output` oraz gniazdo `socket`, a następnie likwidowane jest okno graficzne klienta.

UWAGA:

Program `Klient.java` musi być uruchomiany dopiero po uruchomieniu programu serwera. Można uruchomić wiele kopii tego programu. Każda kopia tworzy własne okno graficzne, w którym odbywa się komunikacja z serwerem.

Program `Tester.java` uruchamia serwer, a po chwili uruchamia dwóch klientów „ADAM" i „EWA".

Uwaga: Proszę obowiązkowo wykonać pierwsze zadanie. Osoby ambitne mogą wykonać również drugie zadania.

Zadanie 1 (obowiązkowe dla wszystkich)

Proszę przygotować aplikację typu „klient-serwer" realizującą funkcję prostej książki telefonicznej. Aplikacja ma się składać z dwóch programów: ***PhoneBookClient*** oraz ***PhoneBookServer.*** Pierwszy program ma być implementacją prostego klienta umożliwiającego użytkownikowi wysyłanie ciągu zleceń tekstowych do serwera i wyświetlanie komunikatów tekstowych z odpowiedziami serwera. Drugi program ma być implementacją serwera przyjmującego polecenia tekstowe od programu klienta i wykonującego zadane operacja na książce telefonicznej.

Polecenia klienta akceptowane przez serwer powinny mieć następującą postać:

„**LOAD** *nazwa_pliku*" - wczytanie danych z pliku o podanej nazwie,
„***SAVE*** *nazwa_pliku*" - zapis danych do pliku o podanej nazwie,
„**GET** *imię*" - pobranie numeru telefonu osoby o podanym imieniu,
„**PUT** *imię numer*" - zapis numeru telefonu osoby o podanym imieniu,
„**REPLACE** *imie numer*" - zmiana numeru telefonu dla osoby o podanym mieniu,
„**DELETE** *imie*" - usunięcie z kolekcji osoby o podanym imieniu,
„**LIST**" - przesłanie listy imion, które są zapamiętane w kolekcji,
„**CLOSE**" - zakończenie nasłuchu połączeń od nowych klientów i zamknięcie gniazda serwera
„**BYE**" - zakończenie komunikacji klienta z serwerem i zamknięcie strumieni danych oraz gniazda.

Odpowiedzi serwera wysyłane do klienta powinny mieć następującą postać:

„**ERROR** *komunikat*" - podczas wykonywania ostatniego polecenia wystąpił błąd – komunikat zawiera informację o przyczynie błędu,
„**OK**" - polecenie LOAD, SAVE, PUT, REPLACE, DELETE, CLOSE, BYE zostało wykonane poprawnie,

„**OK** *numer*” - polecenie GET zostało wykonane poprawnie – numer zawiera numer osoby, której imię było podane w poleceniu,

„**OK** *imie1 imie2 …*” - polecenie LIST zostało wykonane poprawnie, - komunikat zawiera listę imion osób pamiętanych w kolekcji.

Książka telefoniczna to obiekt klasy ***PhoneBook,*** który zawiera kolekcję typu ***ConcurrentMap*** w której przechowywane są pary tekstów <*imię , numer telefonu*>**.** Klasa ***PhoneBook*** powinna posiadać metody wykonujące poszczególne operacja na kolekcji np.:

String LOAD(String nazwa_pliku);
String SAVE(String nazwa_pliku);
String GET(String imię):
String PUT(String imię, String numer);
String REPLACE(String imie, String numer);
String DELETE(String imie);
String LIST();

Wszystkie metody powinny zwracać łańcuch tekstowy zawierający odpowiedź, która powinna być przesłana przez serwer do klienta.

UWAGA: Implementacja serwera ma być zrealizowana tak, by serwer mógł współbieżnie obsługiwać wielu klientów, którzy się z nim połączą. Po nawiązaniu połączenia przez klienta program serwera musi utworzyć nowy wątek, w którym będzie realizowana komunikacja z klientem. W tym czasie główny wątek serwera powinien oczekiwać na kolejne zgłoszenia od innych klientów. Wszyscy klienci mają korzystać ze wspólnej książki telefonicznej. Dlatego w klasie ***PhoneBook*** należy użyć klasy implementującej kolekcję typu „mapa” w wersji bezpiecznej wątkowo czyli np. klasy **ConcurrentHashMap<K,V>** zamiast zwykłej klasy **HashMap<K,V>.**

Zadanie 2 (dla ambitnych)

Proszę przygotować programy będące rozproszonym komunikatorem sieciowym umożliwiającym przesyłanie komunikatów tekstowych między dowolnymi użytkownikami. Serwer komunikatora powinien być pośrednikiem umożliwiającym przekazywanie danych adresowych wszystkich aktywnych użytkowników.

Każdy użytkownik uruchamia swój własny terminal, który umożliwia nawiązanie komunikacji z innymi użytkownikami. Po uruchomieniu, program terminala wysyła swoje dane (nazwa użytkownika, adres komputera i numer portu) do serwera. Następnie terminal oczekuje na połączenie przychodzące od innego użytkownika. W czasie oczekiwania użytkownik może zainicjować połączenie z innym użytkownikiem. W tym celu program terminala powinien pobrać z serwera aktualną listę aktywnych użytkowników, wybrać jednego użytkownika z tej listy, a następnie pobrać z serwera dane adresowe (adres komputera i numer portu) wybranego użytkownika. Po otrzymaniu tych danych terminal inicjuje połączenie sieciowe bezpośrednio z terminalem wybranego użytkownika.

Program terminala powinien przed zamknięciem wysłać do serwera komunikat o zakończeniu pracy, po to by serwer usunął dane użytkownika z listy aktywnych użytkowników.