Architektura programu JP2PChat

Hubert Jatkowski, Mateusz Forc

29.05.2016

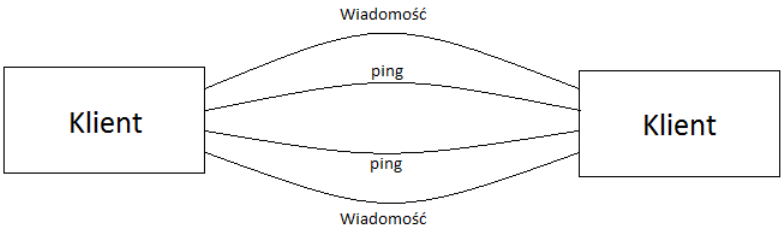
Streszczenie

Tekstowy komunikator internetowy z Graficznym Interfejsem Użytkownika. Projekt wykonywany w semestrze 2016L na przedmiot PROZ.

1 Architektura

1.1 Opis

Program JP2PChat został napisany w języku programowania Java (1.8) z wykorzystaniem biblioteki graficznej swing.
Komunikator został zrealizowany w architekturze p2p (peer-to-peer).



Po uruchomieniu programu, zestawiane jest połączenie pomiędzy klientami. Każdy z nich musi uprzednio wpisać port na którym aplikacja będzie nasłuchiwać nadchodzących wiadomości oraz port na który będą one wysyłane. Aby połączenie zostało poprawnie zestawione porty powinny być wpisane naprzemiennie u rozmówców. Po wpisaniu portów wyliczane są następnie numery portów do badania stanu połączenia - dla każdego klienta odpowiednio port na którym będzie nasłuchiwał wiadomości ”ping” oraz port na który będzie je wysyłał. Oznacza to, że do rozmowy wykorzystywane jest do 4 portów na każdym komputerze:
2 do przesyłania konwersacji, oraz 2 do ”pingowania” się nawzajem.

1.2 Opis klas

1.2.1 MainWindow

Klasa reprezentująca Graficzny Interfejs Użytkownika. Jest odpowiedzialna za ogólny zarys działania programu. Jej składowymi są między innymi przyciski, których akcje sprzężone są z uruchamianiem wątków klas odpowiedzialnych za zestawienie połączenia i rozmowy.

1.2.2 MessageListener

Klasa, której zadaniem jest nasłuchiwanie na wprowadzonym przez użytkownika porcie nowych nadchodzących wiadomości.
Każda instancja tej klasy jest uruchamiana w osobnym wątku.

1.2.3 MessageSender

Klasa, której zadaniem jest wysyłanie na wprowadzony przez użytkownika port wiadomości.
Każda instancja tej klasy jest uruchamiana w osobnym wątku.

1.2.4 PingListener

Klasa, której zadaniem jest nasłuchiwanie na wyliczonym na podstawie wprowadzonego do słuchania wiadomości portu przez użytkownika.
Nieodbieranie wiadomości "ping" przez dłużej niż 300 milisekund sprawia, że użytkownikowi zostaje zakomunikowane przekroczenie czasu oczekiwania na połączenie, czego rezultatem jest zmiana koloru kontrolki połączenia w graficznym interfejsie użytkownika na czerwony. Każda instancja tej klasy jest uruchamiana w osobnym wątku.

1.2.5 PingSender

Klasa, której zadaniem jest wysyłanie na wyliczony na podstawie wprowadzonego portu do wysyłania wiadomości przez użytkownika.
Wiadomości "ping" wysyłane są cyklicznie w 300 milisekundowych odstępach czasowych. Każda instancja tej klasy jest uruchamiana w osobnym wątku.

2 Napotkane problemy w implementacji

2.1 Historie rozmów

Pierwsze próby implementacji tworzenia plików z logiem napotykały problem nazewnictwa plików. Postanowiono utrzymywać unikalność nazw poprzez dodawanie do nazwy aktualnej daty i godziny. Próba zapisania pliku

z czasem w notacji HH:MM:SS w nazwie kończyła się poważnym błędem programu i niemożnością jego normalnego zamknięcia. Problem rozwiązano, zmieniając notację na HH-MM-SS. Powód wystąpienia problemu: Implementacja niektórych systemów plików zabrania używania w nazwach plików i folderów części metaznaków.

2.2 Zwalnianie portów

Napotkaliśmy na początku pisania aplikacji problemy związane z zestawianiem połączenia. Za pierwszym razem gdy próbowaliśmy zainicjować rozmowę testową wszystko wydawało się działać bardzo dobrze, jednak jej ponowne zestawienie okazywało się niemożliwe zarówno w trakcie wykonywania programu jak i po jego ponownym uruchomieniu. Problem został zdiagnozowany dzięki zmianie portów po przy każdym zestawianiu rozmowy, jednak nie było to bez wątpienia jego najlepszym, jeżeli jakimkolwiek rozwiązaniem. Aby uporać się z tym problemem należało zwolnić zasoby tuż przed zakończeniem wykonywania programu.

2.3 Zielono-czerwona lampka

Najwięcej problemów niewątpliwie przysporzyła nam implementacja kontrolki.

2.3.1 Jedna z lampek zawsze czerwona

Podczas pierwszych prób implementacji omawianego elementu graficznego interfejsu użytkownika, jednym z pierwszych problemów które napotkaliśmy był czerwony kolor kontrolki u drugiego w kolejności dołączania do konwersacji klienta. Po wnikliwej analizie kodu źródłowego zauważyliśmy, że przyczyną jest paradoksalnie pierwszy klient, ponieważ gdy zaczyna on nasłuchiwać wiadomości na porcie i jednocześnie wysyłać wiadomości "ping", w momencie odrzucenia pierwszej takiej wiadomości, wątek PingSender kończy swoje działanie. Problem rozwiązaliśmy poprzez zapętlenie wysyłania zapytania z odpowiednim interwałem czasowym (aby nie zwiekszać niepotrzebnie ruchu w sieci).

2.3.2 SocketException: Too many opened files

W trakcie początkowej fazy testowania kontrolki, trafiliśmy na problem, na pierwszy rzut oka, nieuzasadnionego rzucania wyjątku wymienionego w tytule. Problem był spowodowany niezwalnianiem przez klasy nasłuchujące i wysyłające wiadomości "ping" zasobów, tj. obiektów BufferedReader i ServerSocket. Rozwiązanie okazało się trywialne.

2.3.3 Migotanie

W trakcie symulacji i testowania zerwania połączenia przez jednego z rozmówców, po ponownym jego zestawieniu zaobserwowaliśmy migotanie kontrolki statusu. Okazał się to być klasyczny problem synchronizacyjny:
Wątek PingListener zbyt gwałtownie próbował pobierać wiadomości od wątku PingSender, co sprawiało, że przedwcześnie dawał znać o utracie połączenia. Spowodowane było to ciągłym bez jakiegokolwiek opóźnienia nasłuchiwaniem wiadomości ping oraz zastosowanie prostych liczników typu int do odmierzania taktów. Problem został rozwiązany poprzez spowolnienie nasłuchiwania za pomocą wprowadzenia interwału czasowego oraz zastosowaniem liczników pobierających aktualny czas.