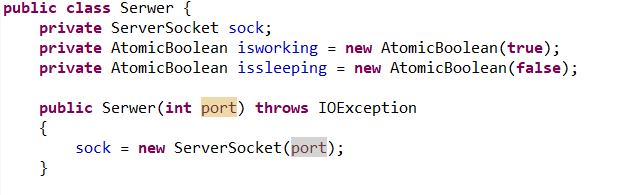
|  |  |  |
| --- | --- | --- |
| **Politechnika Świętokrzyska** | | |
| **Studia stacjonarne (semestr letni)** | | |
| **Programowanie obiektowe(Java) - Projekt** | | **2022** |
| **Skład zespołu:**   * Przemysław Kałuziński * Grzegorz Kalarus * Michał Kaczor | **Grupa:** 2ID13A | |
| **Temat projektu:** Aplikacja realizująca system zarządzania biurem podróży | | |
| **Temat sprawozdania:** Opis interfejsu komunikacji klient-serwer | | |

***Opis interfejsu komunikacji klient serwer:***

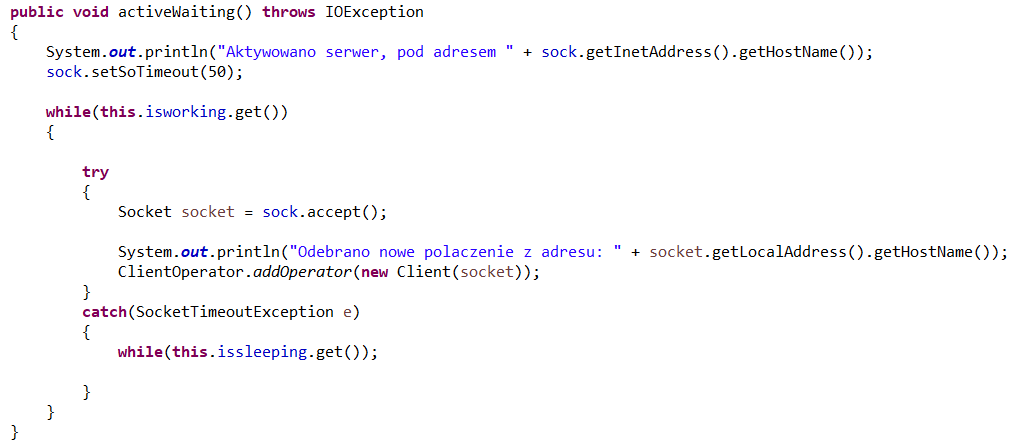
# ***Po stronie serwera (administratora)***

* Aby sprawnie móc korzystać z interfejsu komunikacyjnego klient-serwer, musieliśmy stworzyć klasę przyjmującą nowe połączenia oraz tworzącą nowe wątki. W konstruktorze tej klasy tworzymy nowe gniazdo do przyjmowania użytkowników. Gniazdo przyjmuje port przekazany mu przez parametr.

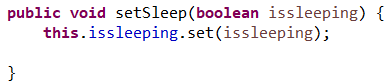


* Następnie zaimplementowaliśmy wewnątrz tej klasy kilka przydatnych funkcji. Są one odpowiedzialne min. za:

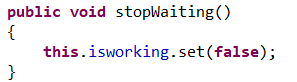
Przyjmowanie i akceptowanie nowych połączeń



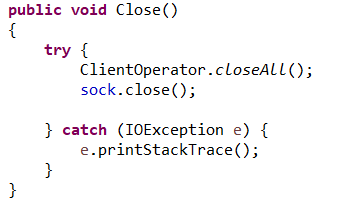
Uśpienie akceptowania nowych połączeń



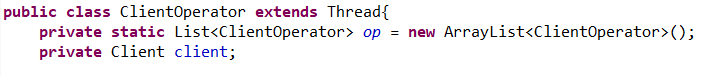
Wyłączenie oczekiwania na połączenia



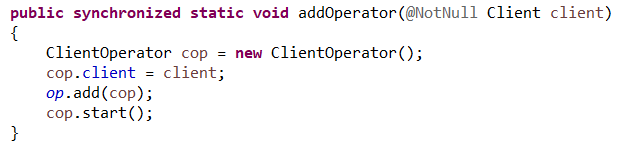
Zamknięcie wszystkich aktywnych połączeń



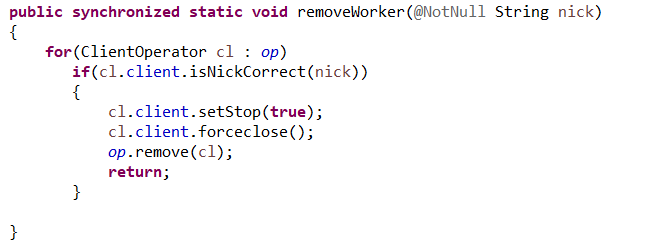
* Aby powyższe metody działały poprawnie potrzebna nam jest jeszcze jedna klasa, w której będziemy przechowywać nowo zaakceptowanego użytkownika.



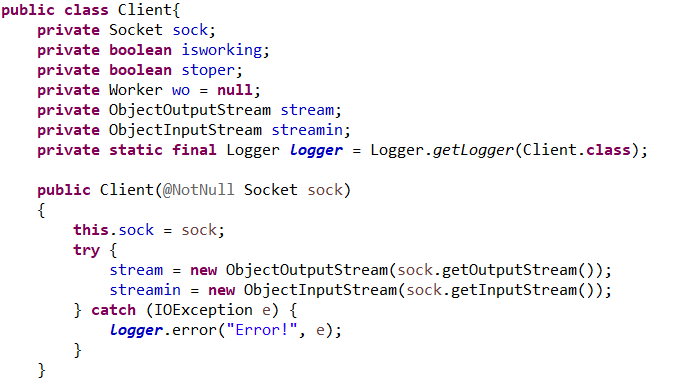
Zawiera ona także metody potrzebne do zarządzania tym użytkownikiem. Między innymi metodę tworzącą nowy obiekt dla zaakceptowanego klienta oraz dodającą go do listy wszystkich aktywnych klientów.



Kolejną przydatną metodą z tej klasy jest metoda wymuszająca zamknięcie połączenia dla konkretnego klienta.



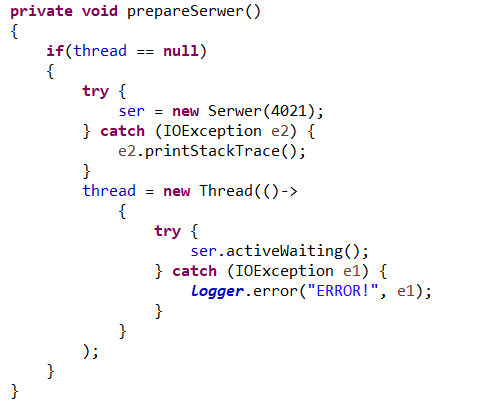
* Klasą, bez której powyższa struktura nie będzie mogła zostać poprawnie użyta jest klasa obrazująca pojedyncze połączenie klienta z serwerem. To właśnie obiekt tej klasy jest przekazywany jako parametr dla metody dodającej nowego użytkownika do listy (addOperator(Client client)). Wywoływana jest ona w metodzie ActiveWaiting() z klasy Serwer, oczekującej i akceptującej nadchodzące połączenia klientów.



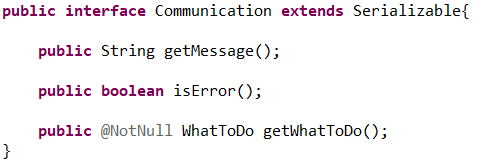
Klasa zawiera także szereg innych metod obsługujących wymianę danych z klientem. Przykładem może być poniższa metoda, która służy do wysyłania klientowi obiektu zawierającego umowy wycieczkowe.



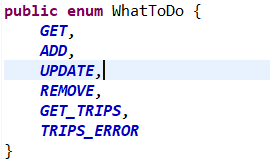
* Serwer tworzymy w metodzie z klasy głównej (Window.java) poprzez wywołanie metody prepareServer(). Uruchamia ona metodę nasłuchującą nadchodzących połączeń.



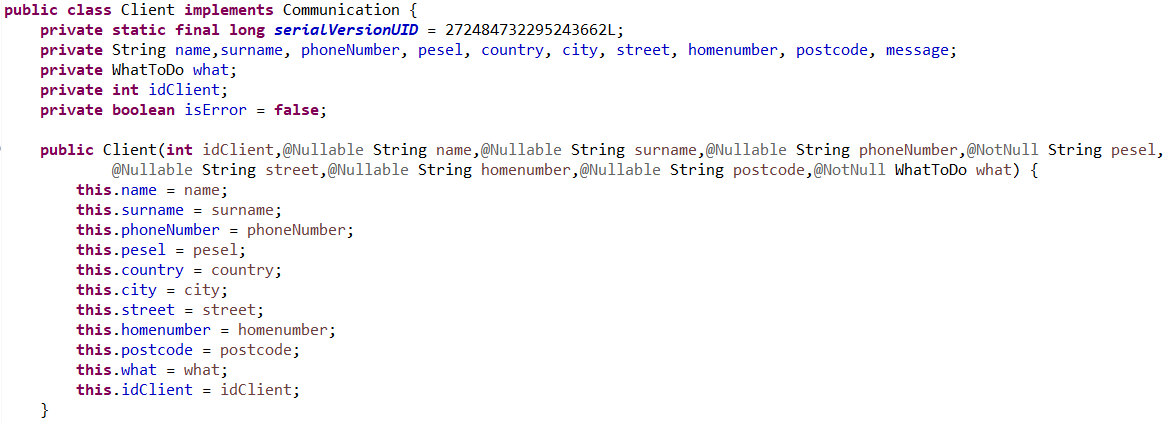
* Następnie potrzebujemy czegoś zapewniającego nam komunikację. W tym celu wykorzystujemy interfejs Communication, który zapewnia trzy podstawowe metody wykorzystywane przez wszystkie inne klasy komunikacyjne. Pierwsza z nich zwraca wiadomość błędu, druga zwraca true jeśli wystąpił błąd, a ostatnia zwraca rodzaj operacji wybrany z typu wyliczeniowego enum.



* Wspomniany wyżej typ wyliczeniowy enum służy nam do określania jaką operację zamierzamy wykonać na obiekcie. Np. operacja GET powiadamia serwer, że klient prosi o zwrócenie konkretnych danych.

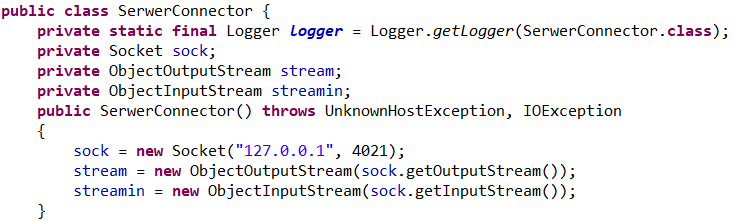


* Z dwoma powyższymi punktami bezpośrednio wiąże się zestaw klas komunikacyjnych. To w tym miejscu ulokowaliśmy metody służące do przesyłania konkretnych danych związanych z używaniem aplikacji przez klienta. Przykładem może być klasa Client.java, która zawiera szczegółowe dane osobowe na temat dodanego użytkownika. Klasa nazywa się identycznie jak poprzednia, ale spełnia zupełnie inne zadanie. Poprzednia odpowiadała obsługę informacji połączeniowych klienta (np. socket), a ta służy do przechowywania danych użytkowych (np. dane osobowe). Dzięki różnym paczkom, w których owe klasy się znajdują, nie występuje konflikt nazw.

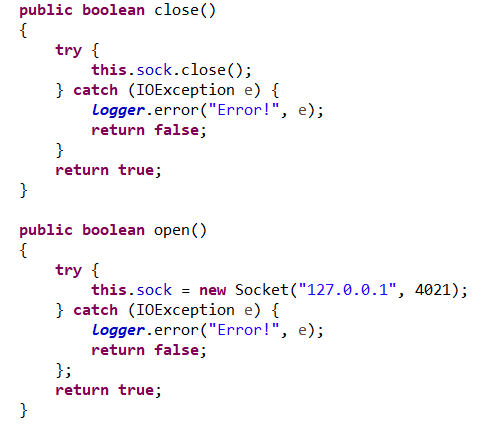
Tego typu klasy służą nam do przechowywania danych użytkowych z całej aplikacji.

# ***Po stronie użytkownika***

* Po stronie użytkownika sytuacja jest o wiele prostsza, gdyż do komunikacji z serwerem wykorzystujemy tylko jedną klasę. W jej konstruktorze tworzone jest nowe gniazdo użytkownika do komunikacji z serwerem oraz dwa strumienie – wejściowy i wyjściowy.



* Zawiera ona dwie bardzo ważne metody, które zamykają lub inicjalizują połączenie z serwerem. Obie zwracają false w przypadku niepowodzenia.

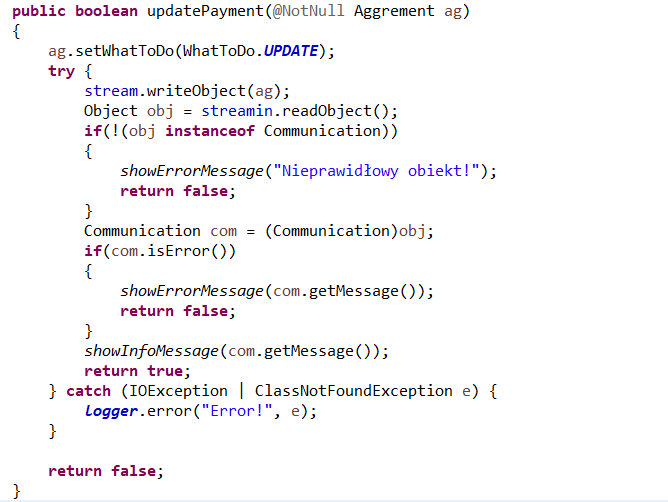


* Znajduje się tu także zestaw metod, które mają za zadanie wykonywanie odpowiednich operacji wymaganych do zapewnienia spójności danych podczas korzystania z aplikacji. Oto kilka przykładowych metod:

Wysyła żądanie do serwera o zalogowanie użytkownika



Aktualizuje płatność klienta naszego biura



Dodaje wycieczkę do bazy danych



* Na koniec w metodzie main() aplikacji klienta tworzony jest nowy obiekt klasy SerwerCOnnector(). Pozwala to na nawiązanie połączenia między bieżącym użytkownikiem, a serwerem.

