SKJ projekt
Serhii Kovalenko s17187, 25c

I. Ogólny opis rozwiązania:

Głownym celem projektu jest implementacja gry “w marynaża” dla dwóch i więcej klientów, które nawiązują połączenie TCP. Dla poprawnej realizacji połączenia użyłem takiej techniki jak programowanie wilelowątkowe, a także stworzyłem słuchacza zdarzeń (action listener), które mogą powstać w trakcie połączenia. W tym celu stworzono interfejs **TCPConnectionListener**, który służy do definiowania zdarzeń, które mogą wystąpić w **TCPConnection**, klasę **TCPConnection**, która pełni rolę klienta, **Server**, zadaniem którego będzie rozpoczęcie gry, dodawanie nowych klientów, przyjmowanie poleceń od klientów, realizacja gry pomiędzy dwoma klientami i zapisania jej wyników, a także **ClientWindow**, który tworzy interfejs GUI kienta.

II. Szczególowy opis rozwiązania:

Wyróżniłem 4 zdarzenia, ktore mogą powstać:

-void onConnectionReady(TCPConnection tcpConnection);
(kiedy klient nawiązał połączenie z serwerem)

-void onReceiveCommand(TCPConnection tcpConnection, String command); (kiedy serwer przyjmuje String, wysyłany przez klienta)

-void onDisconnect(TCPConnection tcpConnection);
(kiedy klient opuscił grę)

-void onException (TCPConnection tcpConnection, Exception e);
(kiedy powstał wyjątek)

TCPConnection (czyli klient) ma pole **eventListener** typu **TCPConnectionListener,** które w trakcie działania programu wywołuje jedną z tych 4 metod, natomiast **Server** i **ClientWindow** implementują interfejs **TCPConnectionListene**r (czyli są one słuchaczami tych zdarzeń) i definiują te metody (w szczególności interesuje nas, jak to robi **Server**).

Klient jest identyfikowany poprzez imię (ktore jest unikalne), numer portu (żeby ustawić port klienta skorzystałem z metody **socket.getPort()**), IP adres (**socket.getInetAddress()**) i numer portu klienta, którego imie wpisujemy jako parametr polecenia JOIN (ponieważ jest ono unikalne, zrobiłem tak, że nie mogą istnieć klienty o jednakowych imionach!!!). W konstruktorze klienta tworzymy wątek (który działa, dopóki jego nie przerwiemy metodą **interrupt()**), w którym wysyłamy jakiś tekst do serwera (mogą to być polecenia lub liczby całkowite (w przypadku gry)), gdzie jest on przetwarzany w określony przezemnie sposób. Ten konstruktor przyjmuje jako parametr słuchacza zdarzeń (**TCPConnectionListener**).

Server w nieskończonym cyklu przyjmuje nowe połączenia i dla każdego z nich tworzy nowy **TCPConnection** (w konstruktorze **TCPConnection** jako słuchacza zdarzeń wpisujemy **Server** (czyli **this**)). Nowo stworzony **TCPConnection** jeszcze nie jest podłączony do gry, a po prostu czeka w "kolejce". Wszystkie klienty, który czekają na początek gry są zapisane na liście **ArrayList<TCPConnection> connections** (**connections** – to jest lista tych klientów, który już ustawili swoje imie i wysłali do serwera komunikat JOIN z imionem jednego z klientów na tej liście (czyli, to nie jest lista klientow, który czekają w "kolejce"!!!)). Klient, który czeka w "kolejce" i chce zagrać w grę musi najpierw ustalić swoje imie poprzez polecenie SET <imie>. Przez polecenie ACK on może dostać informację o klientach, który czekają na grę (czyli tych, kto już jest na liście **connections**). Podłączyć się do gry on może uzywając polecenia JOIN imie, gdzie imie musi być imionem jednego z klientów z listy **connections** (jeżeli ta lista jest pusta, uzywamy swoją nazwę) (polecenie JOIN może być uzywane tylko wtedy, gdy imie klienta już jest ustalone!). W celu zapewnienia unikalności imion wszyskich klientów, po dodaniu klienta na listę, dodajemy jego imie do listy imion wszystkich klientów z listy **connections**. Jeżeli takie imie już istnieje, ten nowy klient nie będzie dodany na listę **connections**, i będzie musiał powtórzyć operacji ustawienia nazwy i podłączenia do listy. Jeżeli on został dodany na listę, może tą listę opuścić poprzeż polecenie QUIT, ale tylko wtedy, gdy gra jeszcze nie zaczeła się. Kiedy klient uzyje polecenia QUIT, jego nazwa sostanie usunięta z listy imion, żeby inny klient mógł potem użyć tej nazwy.

W programie serwera została stworzona metoda **onGame()**, ktora staruje grę dla listy klientów **connections**, oczekujących na grę. Ta metoda musi być wywolana rekuręcyjnie i tylko wtedy, gdy co najmniej 2 klienta czekają na początek gry, czyli **connections.size()** > 1. Ponieważ nie ma mozliwości wycofać się z gry w trakcie jej działania, jak również nie ma możliwości dla klientów, oczekujących w “kolejce” podłączyc się do gry (kiedy gra się staruje, “kolejka” zamyka się), musi być stworzony timer, który startuję grę przez jakiś czas i zapewnia przerwy pomiędzy kolejnymi grami, żeby klienty mogli opuścić listę **connections** i wrócić do “kolejki” lub jakiś nowy klient z “kolejki” dołączył się do tej listy i będzie mógł zagrać w nastepną grę. Timer startuje się kiedy pirwszy klient zostanie dodany na liście **connections**. Po wystartowaniu Timer co 10 sekund sprawdza rozmiar tej listy, i jeżeli jest on większy od 1, zamyka “kolejkę”, kasuje samego siebie (**time.cancel()**) i startuje grę. W metodzie **onGame()** rozbijamy na pary wszystkich graczy z listy **connections** i zapisujemy ich w **List<List<TCPConnection>> pairs** (n.p jeżeli mamy listę klientów {A, B, C, D}, to pary bedą: [{A,B}, {A, C}, {A, D}, {B, C}, {B, D}, {C, D}]). Tworzymy również listę wyników wszystkich gier. Potem uzywajc petli for() dla każdej pary rozgrywamy grę: oba podają liczbę, ktore natychmiast zostaną zaszyfronane przez dodanie do tej liczby klucza klienta, który został wygenerowany dla niego jeszcze wtedy, kiedy klient tylko nawiązal połączenie z serwerem (**TCPConnection** ma pole int KEY, wartości klucza są generowane przez funkcję **Math.random()*1000**). Potem ustała się od kogo zaczyna się odliczanie (znowu poprzez funkcję **Math.random()*2**, która może przyjmować wartości 0 lub 1; jeżeli ona ma wartośc 0, zaczynamy od pirwszego gracza z tej pary, czyli zapisujemy jego w pole **TCPConnection start**, a innego w pole **TCPConnection other**, a jeżeli 1, to na odwrót). Potem sumujemy te dwie zaszyfrowane liczby i od tej sumy odejmujemy sumę kluczy tych graczy, wartości których dostaniemy przez metodę **getKEY()** w klasie **TCPConnection**. Jesli ta suma jest liczba nieparzystą, to wygrywa ten, od którego zaczynamy odliczanie, a jeżeli jest parzystą, to wygrywa ten drugi. Potem wyniki tej gry (gdzie jest informacja o nazwach klientów, o liczbach, podanych przez nich, od kogo zaczeło się odliczanie i kto wygrał) zapisują się na listę wyników. Kiedy gra się skończy dla wszystkich klientów na liście **connections**, znowu wywolujemy metodę **Timer()**, która czeka 10 secund, a potem co 10 secund sprawdza rozmiar tej listy i zaczyna nową grę, jeżeli ten rozmiar jest wiekszy od 1.

III. Obserwację eksperymenty i wnioski

Teraz sprobujemy uruchomić grę dla 3 graczy. Uruhomiamy okno klientskie pierwszego klienta. Jeżeli sprobujemy napisać polecenie JOIN A, zostaniemy poproszeni ob ustaleniu nazwy klienta przed wywolaniem JOIN'a. Ustalamy nazwę SET A, a potem piszemy JOIN A i już jesteśmy dodani do listy connections. W tym momencie została uruchomiona metoda **Timer()**, ktora czeka 10 sekund, a potem co 10 sekund sprawdza rozmiar listy **connections**. W danym momencie jej rozmiar jest równy 1, wiec, nie możemy zacząc grę i czekamy na innych graczy. Teraz uruhamiamy nowego klienta, ustawiamy imię na B i dołączamy się do gry. Uruhamiamy jeszcze jedno okno i spróbujmy ustawić imię jako B, a potem dołączyć się do gry. Otrzymaliśmy informację, że takie imie już istnieje, wiec musimy powtórzyć te operację. Teraz ustawiamy imię jako C i dołączamy się do B i teraz jest OK. Rozmiar listy **connections** jest już równy 3, wiec możemy zacząć grę. Jeżeli teraz sprobujemy ją opuścić lub dołączyć się, to nic nam się nie uda.
Teraz A i B muszą podać liczby, a C musi czekać. Jeżeli sprobujemy wpisać coś opróc liczby całkowitej, to otrzymamy informację, że to nie jest int i musimy wpisać liczbę ponownie. Wtedy wpiszemy jako A liczbę 2, a jako B – 5. W wyniku widźimy że odliczanie zaczeło się od B i B wygrywa. Teraz grają A i C, a B czeka. Jako A podajemy 4, a jako C – 12, i w wyniku otrzymujemy, że odliczanie zaczeło się od C i wygrywa A. No i w końcu grają B z C, B podaje 6, a C – 9, i w wyniku mamy, że odliczanie zaczeło się od B i wygrywa B. Teraz gra jest skończona i możemy ją opuścić (czyli wrócić do "kolejki") lub ktoś z kolejki może dołączyć się do gry, która zacznie się za 10 sekund, jeżeli będzie wystarczająca ilośc graczy. Jeżeli wszyscy opuszczą grę, Timer() nie będzie nic sprawdzał, a po prostu wykasuje się i nie zacznie swoje działanie, dopóki ktoś z "kolejki" nie dołączy się do gry.