# I. Wstep

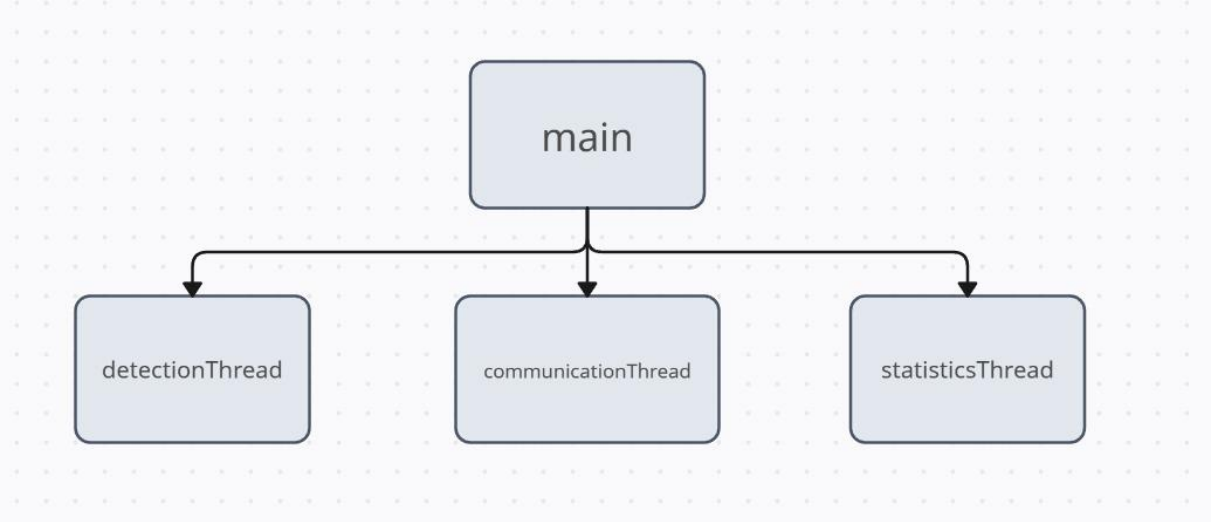
Na aplikację składa się jeden program implementujący klasę CCS (Centralized Computing System) uruchamiany poleceniem:

> java -jar CCS.jar <port>

gdzie <port> jest liczbą określającą numer portu UDP/TCP. Aplikacja jest serwerem obliczeniowym udostępniającym trzy funkcjonalności: detekcji usługi, komunikacji z klientem oraz raportowania statystyk. Wszystkie te funkcjonalności są dostępne jednocześnie.

*Treść zadania*

Program się składa z metody main i 3 odgałęzień - wątków detekcji, komunikacji i statystyki.




# II. Main

Implementacja zaczyna się od sprawdzania ewentualnych błędów. Pierwszy ewentualny błąd, który jest sprawdzany przez konstrukcję if, to argument, który zawiera więcej niż jedno słowo. Reszta błędów jest opracowywana przez try-catch. Szczególnym przypadkiem jest błąd, kiedy argument nie jest liczbą. Dwa powyższe błędy skutkują awaryjnym zakończeniem programu. Serwer konwertuje podany port w liczbę całkowitą.

W przypadku pomyślnego numeru portu, serwer jednocześnie przechodzi do trzech ww. trybów.

# III. Detekcja

Tryb detekcji polega na sprawdzaniu na zawartości nastepnie przekazanej wiadomości i wysłaniu odpowiedzi. Po otrzymaniu wiadomości serwer dalej nasłuchuje wiadowości klienta, dlatego wątek mieści się w nieskończonym cyklu.

W tym celu serwer otwiera gniazdo na porcie pod tym numerem, gdzie nasłuchuje wiadomości od klienta. Tworzy również bufor, w którym będą przychowywane dostarczane dane. Jako że mamy do czynienia z protokołem UDP, używamy abstrakcji pakietu: tworzymy pakiet, który będzie dostarczał dane do buforu. Dostarczane dane również konwertujemy. Tym razem w format String.

W przypadku otrzymania wiadomości zaczynającej się od CCS Discover, serwer wysyła wiadomość zwrotną – CCS Found. W tym celu konwertujemy tę wiadomość w tablicę bajtową. Pobieramy adres i port klienta. Tworzymy pakiet, który zawiera tę wiadomość i dane klienta. Wysyłamy. Problem: w zadaniu nie jest określony numer portu klienta, więc domyślnie uznałem, że będzie to port, z którego była wysyłana wiadomość przez klienta, w przeciwnym razie, wymaga to dodatkowej implementacji.

Wątek jest zawarty do funkcji try-catch, ewentualne błędy będą zostaną zasygnalizowane, ale nie przerwą działania programu. Za jeden z krytycznych błędów na tym etapie uważam brak łączności ze wskazanym portem. Aczkolwiek sugestię, że klient dokładnie zna port, z którym chce się połączyć, uznałem za podstawę nieuznawania tego błędu (błędny numer portu lub brak łączności z portem) za istotny w kontekście tego programu, w przeciwnym razie, wymaga to dodatkowej implementacji.

# IIII. Komunikacja

Zapoznanie się z implementacją communicationThread należy zacząć od rozdziału Input. Pierwszym punktem jest odbiór połączenia przez inputStream, który został użyty zamiast bufferReader w celu bardziej przejrzystego zademonstrowania przebiegu procesu, a także zrobienia go podobnym do procesowania poprzedniego wątku. Dane zapisują się do buforu, potem konwertują się do String, aż w końcu będą podzielone na tablicę stringów.

Dalej jest implementacja odpowiedzi na ewentualne błędy. Pierwszym jest niewłaściwa liczba argumentów. Drugim jest sytuacja , kiedy argument drugi i trzeci nie są liczbami całkowitymi, jeśli są, to następnie konwertowane do int.

Drugim punktem jest obliczanie wyniku. Wewnątrz tego obliczania jest również opracowanie błędu dzielenia przez zero. Obliczenie bazuje na pętli switch, która sprawdza poprawność pierwszego argumentu. W przypadku podania przez użytkownika wartości pierwszego argumentu równej ADD, funkcja przeprowadzi matematyczną operację sumy. W przypadku argumentu SUB znajdzie różnicę, MUL to operacja mnożenia, DIV poda całkowitą część wyniku operacji dzielenia. Wartość pierwszej zmiennej różna od jednej z czterech wyżej wymienionych wartości poskutkuje wypisaniem informacji w polu błędów, nadpisaniem znaczenia true zmiennej-fladze error i zakończeniem bieżącej iteracji i przystąpieniem do nowej. W

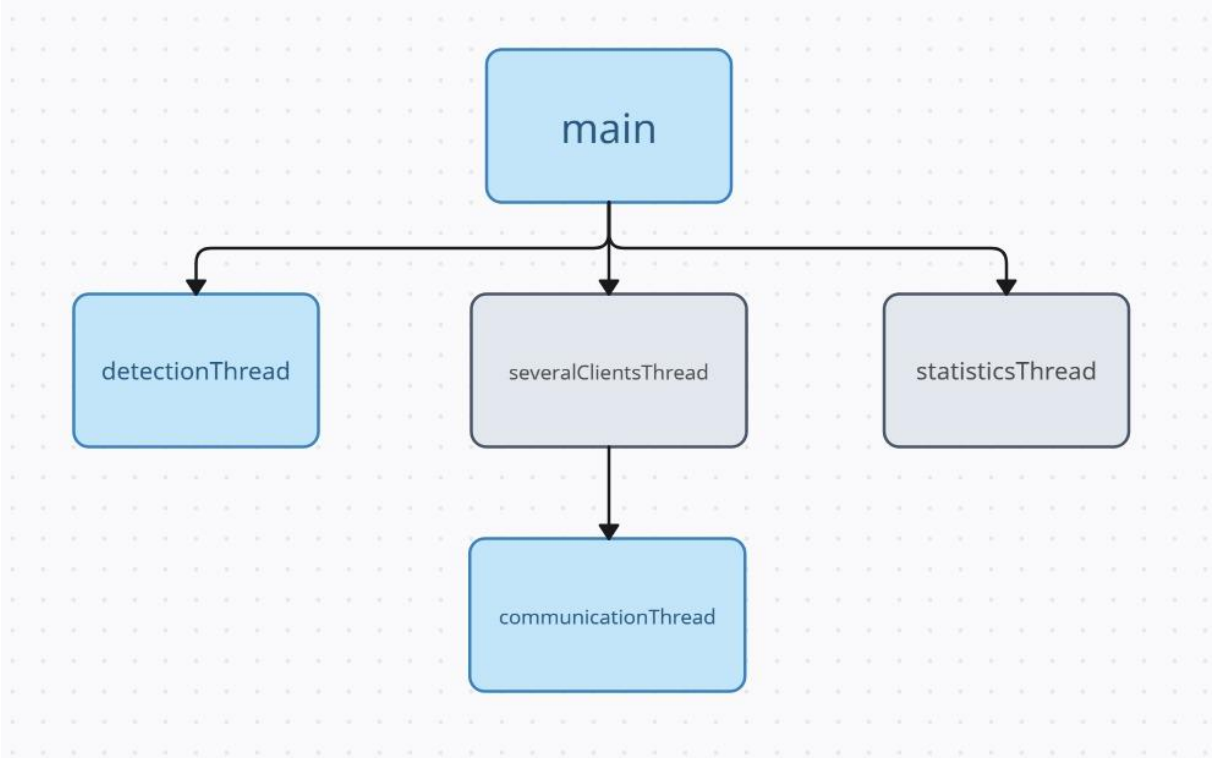
taki sposób opracowywany jest każdy wyżej wymieniony error. W celu dalszej analizy przebiegu procesu opracowania błędów warto się przenieść na początek metody. Technicznie zacznie się nowa iteracja, lecz w sensie logicznym będzie to kontynuacja starej. Zmienna response nabędzie wartość "ERROR" i zostanie wysłana do użytkownika. Z flagi error zostanie zdjęte znaczenie true. W sensie logicznym zatem zacznie się nowa iteracja. W sensie technicznym będzie to kontynuacja wyżej opisanej.

W przypadku niewystąpienia żadnego z powyższych błędów, iteracja nie napotka żadnego continue i nie zakończy swojego działania, dochodząc do nadpisania zmiennej response znaczeniem result, które otrzymamy jako wynik matematycznej operacji w switch. Zostanie ono wysłane do klienta, na konsolę zostanie wypisany wynik i rodzaj przeprowadzonej operacji matematycznej. Był to punkt trzeci i czwarty.

Piątym punktem metody jest odświeżenie statystyk, które odbywa się w trakcie całej pętli i przeniesienie się na początek pętli. Jako, że flaga error przez cały czas ma znaczenie false, rozpoczęcie nowej pętli nie skutkuje wysłaniem wiadomości error do użytkownika.

# V. Kilka klientów

Obsługę wielu klientów umożliwi severalClientsThread. Wymaga tego specyfika TCP polegająca na budowaniu stabilnych kanałów, z kolei kanały datagramowe pozbawione są tego wymogu. Jedynym zadaniem tego wątku jest założenie gniazd dla nieokreślonej liczby klientów.



# VI. Statystyka

Zmienne umożliwiające funkcjonowanie wątku statystyka zaimplementowane i nadpisywane są w wielu miejscach klasy. Dodane są pole statyczne klasy, które umożliwiają gromadzenie statystyk ze wszystkich metod-wątków danej klasy. Zbieranie liczby klientów odbywa się w wątku severalClientsThread. Reszta statystyk, takich jak liczba błędnych operacji, liczba obliczonych operacji, liczba wszystkich operacji i suma rezultatów, dostarczane są z wątku communicationThread. Realizacja wątku statisticsThread polega na aktualizacji informacji o obecnym stanie statystyk, odczekaniu 10 sekund, ponownym zebraniu zaktualizowanych informacji, podaniu zaktualizowanych informacji i podaniu różnicy zaktualizowanych informacji i starych informacji, czyli salda zmian, które zaszły w ciągu ostatnich 10 sekund. Zmiany są wyświetlane na konsoli.