Programowanie obiektowe

Komunikacja z użyciem gniazd – aplikacje klient-serwer

Paweł Rogaliński
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

pawel.rogalinski@pwr.wroc.pl

Architektura klient-serwer

Architektura klient-serwer to technologia budowy systemów informatycznych, w których współdziałają dwie kategorie programów (procesów): klienci i serwery.

Klient to program (proces), który łączy się z innym programem (procesem) zwanym serwerem, i poprzez kanały komunikacyjne zleca mu wykonanie określonych działań (np. dostarczenie pewnych danych).

Serwer to program (proces), który na zlecenie klientów świadczy określone usługi (np. dostarcza określone dane lub wyniki przetwarzania przesłanych danych).

Uwaga:

Pojęcie „**klient-serwer**” jest natury softwarowej, a nie hardwarowej
tzn. dotyczy programów (procesów lub wątków) a nie sprzętu.

Architekturę „klient serwer” możemy zastosować do budowy systemu działającego na jednym komputerze w jednym systemie operacyjnym, a nawet systemu składającego się z jednego programu.

Architektura klient-serwer

Komunikację pomiędzy procesami-serwerami i procesami-klientami w różnych systemach operacyjnych zapewniają mechanizmy komunikacji międzyprocesowej (*IPC – interprocess communication*):

- potoki,
- wspólne (dzielone) obszary pamięci,
- kolejki komunikatów,
- semafory.

Powyższe środki współdziałania aplikacji klient-serwer mają dwie wady:

- są różne, co do sposobu i zakresu, w różnych systemach operacyjnych,
- nie dają się łatwo uogólniać na interakcję pomiędzy serwerami i klientami w środowisku sieciowym

Protokoły interakcji sieciowej

Komunikację w środowisku sieciowym zapewniają specjalne protokoły interakcji sieciowej.

Protokół to swoisty język i zasady, za pomocą których komunikują się programy i procesy, w szczególności serwery i klienci.

Protokoły interakcji sieciowej są podzielone na warstwy, które wyznaczają poziomy komunikacji. Dzięki temu złożony problem interakcji sieciowej jest podzielony na podproblemy, za rozwiązanie których odpowiadają protokoły poszczególnych warstw.

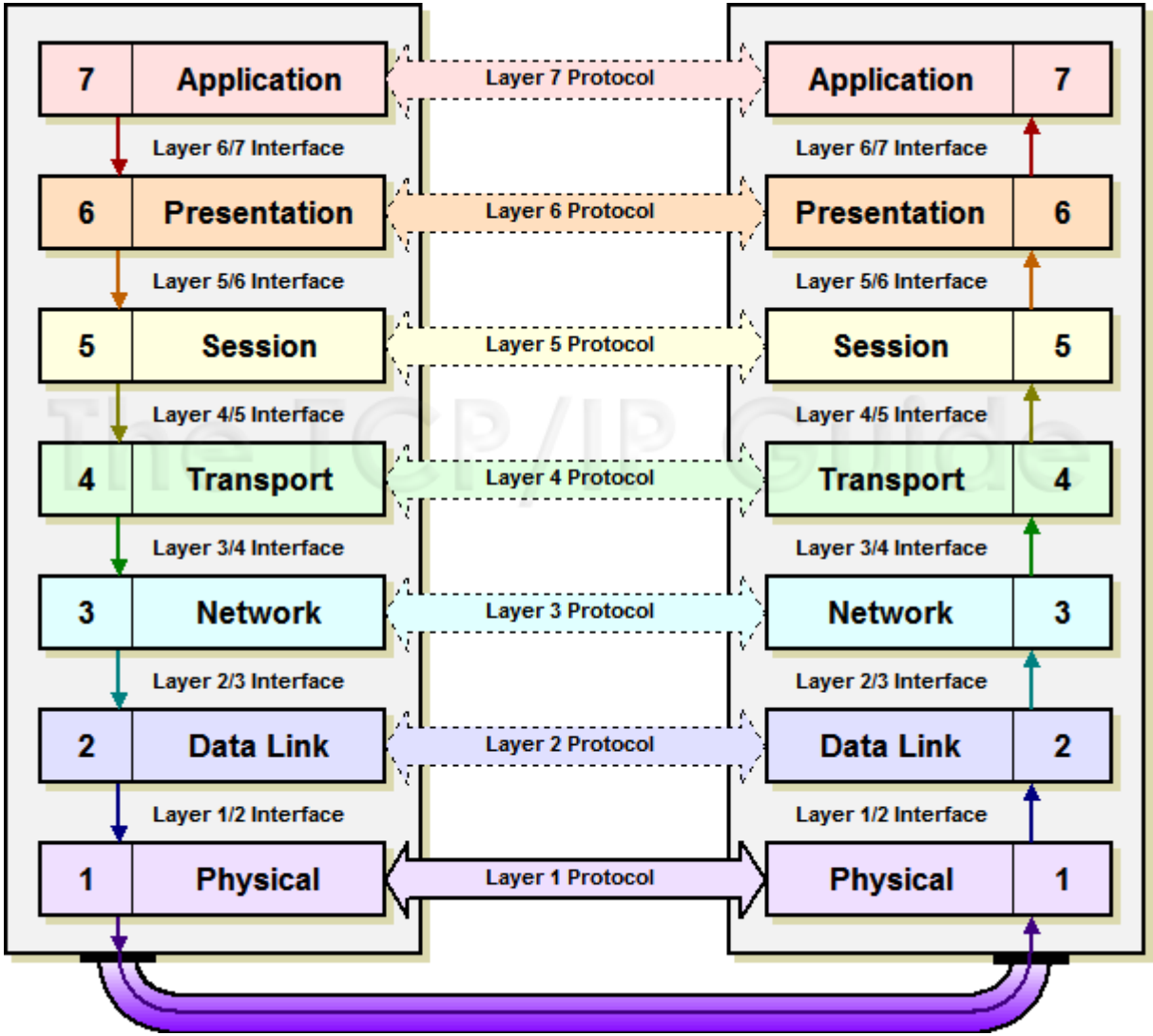
Model ISO – OSI – ogólny standard podziału komunikacji sieciowej na 7 warstw współpracujących ze sobą w ściśle określony sposób.

(OSI - Open Standards Interconnection)

Uwaga:

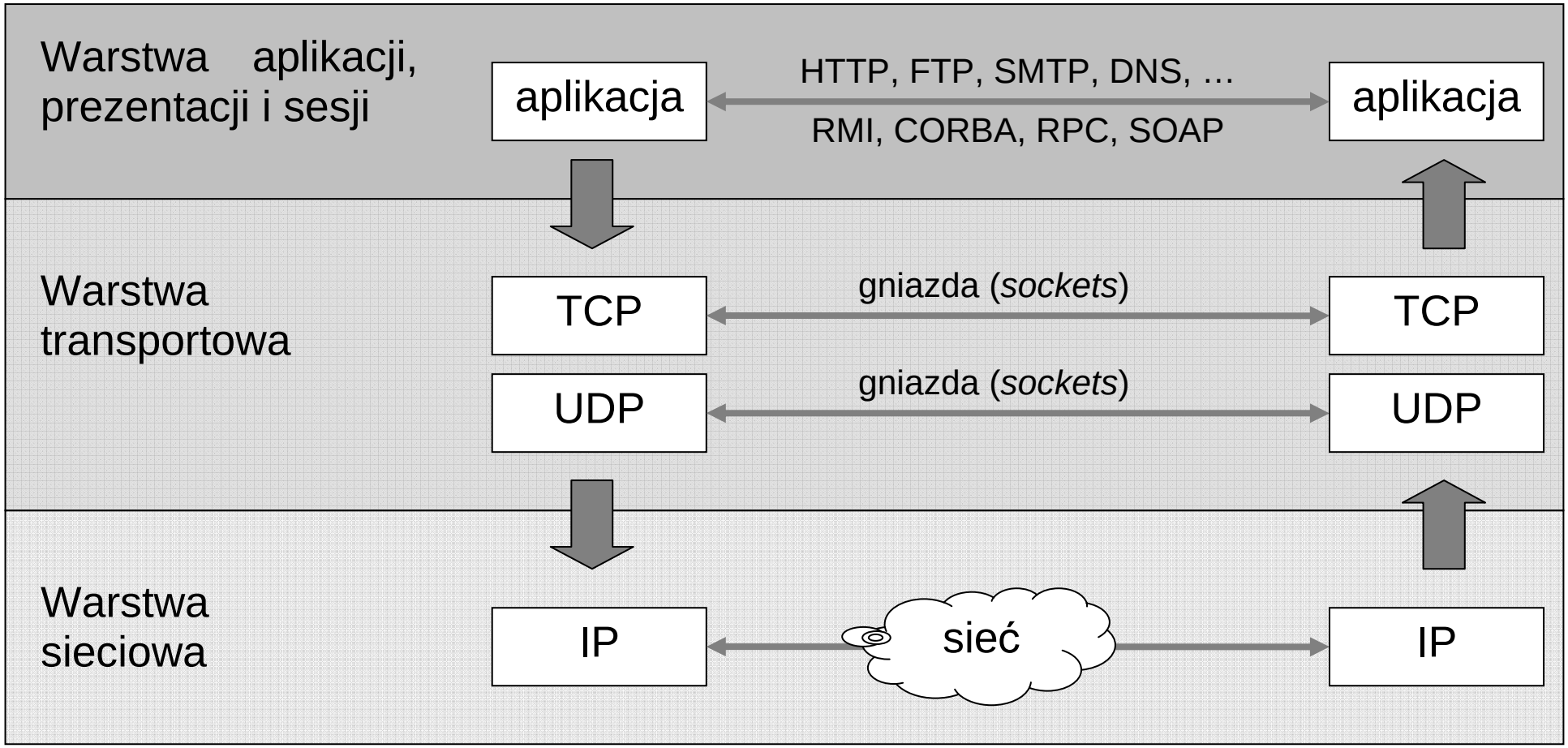
Model ISO - OSI jest traktowany jako model odniesienia (wzorzec) dla większości rodzin protokołów komunikacyjnych.

Model ISO - OSI



Model TCP/IP

Typową realizacją modelu OSI stosowaną w Internecie jest **Model TCP/IP**, który wykorzystuje protokoły **TCP** i **UDP** w warstwie transportoerj oraz protokół **IP** w warstwie sieciowej



Protokół TCP/IP

Protokół IP (*Internet Protokole*) – protokół bezpołączeniowy (zawodny), który umożliwia przesyłanie pakietów pomiędzy komputerami połączonymi w Internecie i identyfikowanymi za pomocą adresu IP.

Protokół TCP (*Transport Control Protocol*) - protokół połączeniowy, który zapewnia że dane posyłane przez gniazda docierają w całości i w odpowiedniej kolejności.

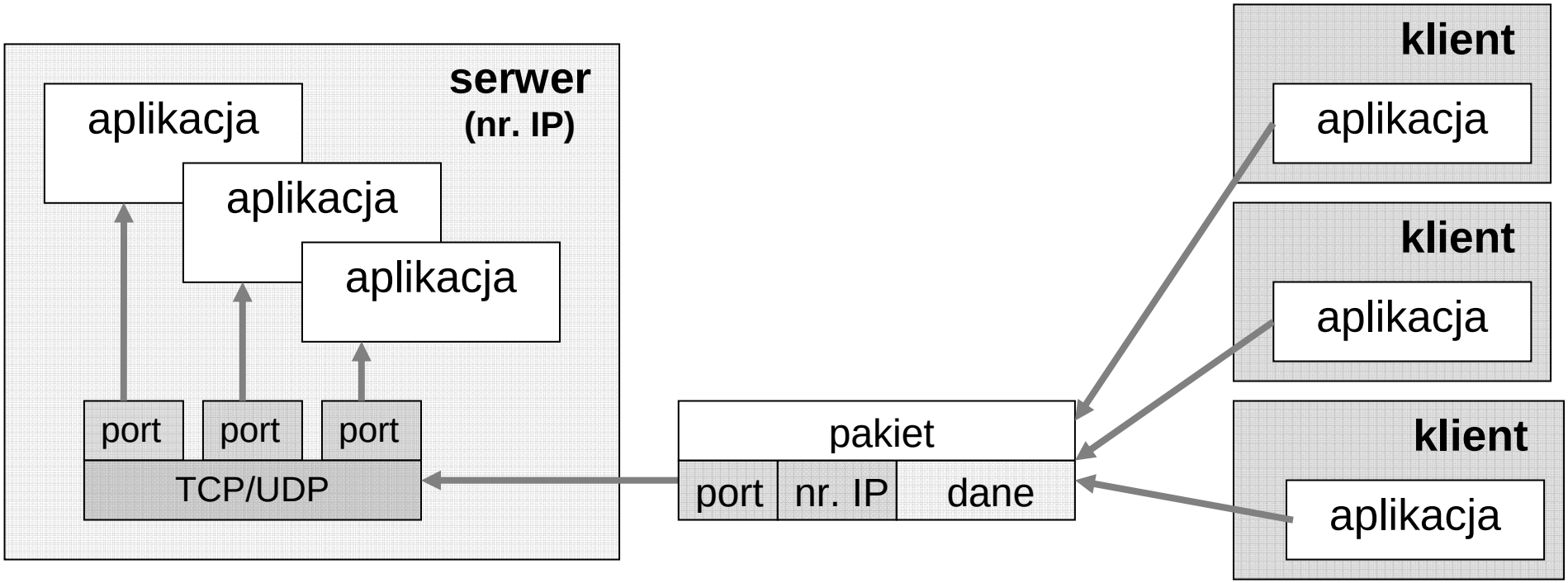
Protokół UDP (*User Datagrams Protocol*) – protokół bezpołączeniowy , w którym dane są przesyłane jako datagramy, które mogą docierać w dowolnej kolejności, lub mogą zostać zagubione.

Uwaga:

Protokoły **TCP** i **UDP** są protokołami „*point-to-point*” czyli zapewniają komunikację pomiędzy dwoma procesami identyfikowanymi za pomocą portów.

Komunikacja implementowana jest za pomocą gniazd, które mają przypisany określony numer portu.

Protokół TCP/IP



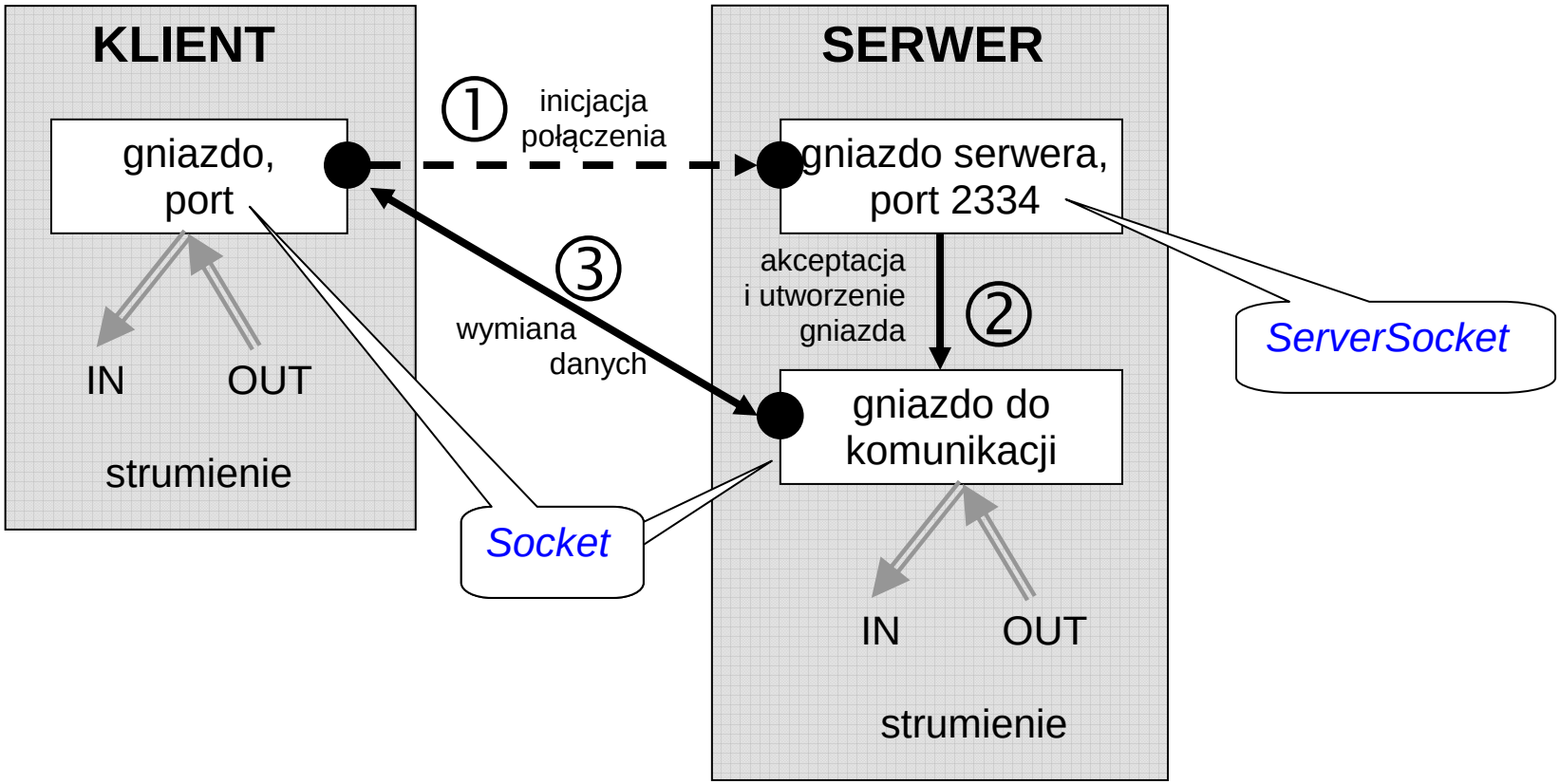
Komunikacja klientów z serwerem poprzez porty

Identyfikacja hosta $\rightarrow$ nr. IP
Identyfikacja aplikacji $\rightarrow$ nr. portu

Model komunikacji „klient-serwer”

- Serwer tworzy gniazdo związane z określonym portem i na tym kanale komunikacyjnym czeka na prośbę połączenia od klienta.
- Inicjatywa połączenia wychodzi od klienta. Klient musi znać host serwera oraz numer portu otwartego do przyjmowania połączeń. Podaje te informacje przy tworzeniu gniazda.
- Serwer akceptuje połączenie od klienta i aby pozostać dostępnym dla innych klientów na „kanale połączeniowym”, tworzy inne gniazdo do komunikacji z danym klientem.
- Z punktu widzenia klienta jest to samo gniazdo, na którym zainicjowano połączenie. Strumienie wejściowy i wyjściowy związane z tym gniazdem służą do komunikacji pomiędzy klientem i serwerem.

Model komunikacji „klient-serwer”



- **Gniazda serwerowe** – używane przy programowaniu serwerów
  (w Javie klasa *ServerSocket*)

- **Gniazda klienckie** – używane w programowaniu klientów oraz serwerów po zaakceptowaniu połączenia
  (w Javie klasa *Socket*)

Aplikacja klienta (szablon)

Oprogramowanie klienta, komunikującego się z serwerem, polega na wykonaniu następujących czynności:

- utworzenie gniazda czyli obiektu klasy ***`Socket`***,
- pobranie od tego obiektu strumieni wejściowego i wyjściowego, związanych z gniazdem,
- posyłanie zleceń do serwera poprzez zapis danych do strumienia wyjściowego gniazda
- odczytywanie odpowiedzi serwera poprzez odczyt ze strumienia wejściowego,
- zamknięcie strumieni
- zamknięcie gniazda .

Uwaga:

Operacje zapisu i odczytu można wielokrotnie powtarzać.

Strumienie wejściowy i wyjściowy można opakowywać by zapewnić określony rodzaj przetwarzania np. buforowanie, kodowanie, odczyt danych binarnych.

Aplikacja klienta (szablon)

```
try {
        String serverHost = "localhost"; // adres IP serwera
        int serverPort = 12312;          // numer portu, na którym
                                         // nasłuchuje serwer
        Socket socket = new Socket(serverHost, serverPort);

        // uzyskanie strumieni do komunikacji
        OutputStream sockOut = socket.getOutputStream();
        InputStream sockIn = socket.getInputStream();

        // wysłanie zlecenia do serwera
        sockOut.write(...);

        // odczyt odpowiedzi serwera
        sockIn.read(...);

        // zamknięcie strumieni i gniazda
        sockOut.close();
        sockIn.close();
        socket.close();
    } catch (UnknownHostException e) {
        // nieznany host
    } catch(SocketException e){
        // wyjątki związane z komunikacją przez gniazdo
    } catch (IOException e) {
        // inne wejątki wej/wyj
    }
```

Aplikacja serwera (szablon)

Typowe działanie serwera polega na wykonaniu następujących czynności:

- utworzenie gniazda serwera czyli obiektu klasy ***`ServerSocket`***,
- związanie gniazda z określonym adresem i numerem portu,
- oczekiwanie na połączenie od klienta „na tym” gnieździe,
- utworzenie nowego gniazda do wymiany informacji z klientem czyli obiektu klasy ***`Socket`***,

- pobranie strumieni wejściowego i wyjściowego, związanych z nowym gniazdem,
- odczytywanie zleceń klienta poprzez odczyt ze strumienia wejściowego,
- posyłanie odpowiedzi do klienta poprzez zapis danych do strumienia wyjściowego,
- zamknięcie strumieni oraz gniazda do komunikacji z klientem

- kontynuacja „nasłuchu” połączeń od innych klientów na gnieździe serwera
- zamknięcie gniazda serwera.

Aplikacja serwera (szablon)

Uwaga:

Operacje związane z nowym gniazdem mogą być wykonywane w dodatkowym wątku współbieżnie z wątkiem realizującym „nasłuch” na gnieździe serwera.

Operacje odczytu i zapisu można wielokrotnie powtarzać.

Strumienie wejściowy i wyjściowy można opakowywać by zapewnić określony rodzaj przetwarzania np. buforowanie, kodowanie, odczyt danych binarnych.

Aplikacja serwera (szablon)

```
try{
        String host = "localhost"; // nazwa hosta, na którym pracuje serwer
        int port = 12312;          // numer portu, na którym nasłuchuje serwer

        //Utworzenie gniazda serwera
        ServerSocket serverSock = new ServerSocket(port);

        // Akceptacja połączeń od klientów
        // Zwykle powtarzane w pętli w oczekiwaniu na kolejnych klientów
        while(true){
            // Akceptacja połączenia i utworzenia nowego gniazda
            Socket socket = serverSock.accept();

            // uzyskanie strumieni do komunikacji
            OutputStream sockOut = socket.getOutputStream();
            InputStream sockIn = socket.getInputStream();

            // odczyt zlecenia oraz wysłanie odpowiedzi do klienta
            sockIn.read(...);       sockOut.write(...);

            // zamknięcie strumieni i gniazda
            sockOut.close();    sockIn.close();    socket.close();
           }

        serverSock.close();   // Zamknięcie gniazda serwera
    }catch(IOException){
        // obsługa wyjątków
    }
```

Książka telefoniczna – aplikacja typu „klient-serwer”

Założenia

Aplikacja składa się z trzech klas:

- ***`PhoneBook`*** – implementacja operacji na kolekcji typu ***`ConcurrentMap`***, w której są przechowywane pary <*imię* , *numer telefonu*>
- ***`PhoneBookServer`*** – implementacja serwera, przyjmującego zlecenia od klientów i wykonującego operacje na kolekcji
- ***`PhoneBookClient`*** – implementacja prostego klienta wysyłającego ciąg zleceń do klienta

Książka telefoniczna – aplikacja typu „klient-serwer”

Założenia (cd)

Polecenia klienta akceptowane przez serwer:

„**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.

Książka telefoniczna – aplikacja typu „klient-serwer”

Założenia (cd)

Odpowiedzi serwera wysyłane do klienta:

„**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.