

Wykład 2

Gniazda

Custom Networking

<https://docs.oracle.com/javase/tutorial/networking/index.html>

Obiekty sieciowe – gniazda

Komputery w sieci Internet komunikują się ze sobą poprzez:

- TCP (Transmission Control Protocol)
- User Datagram Protocol (UDP).

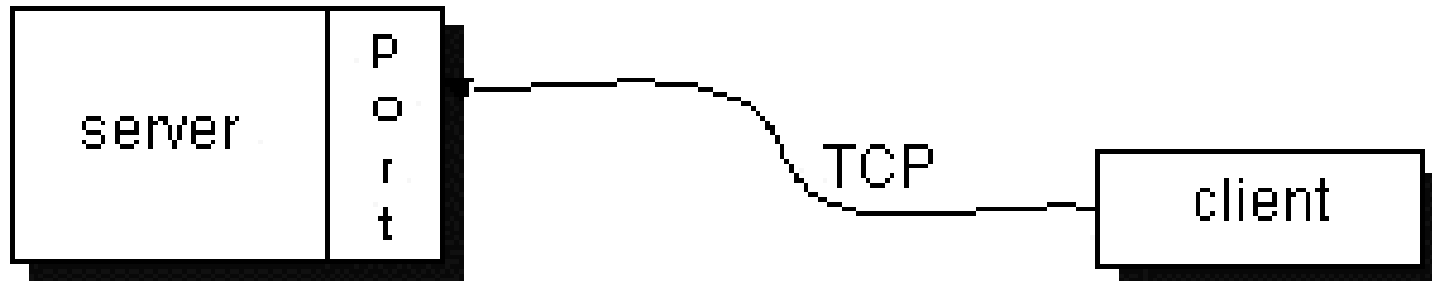
Warstwa Aplikacji	HTTP (Hypertext Transfer Protocol) FTP (File Transfer Protocol) TELNET	Udostępnia procesom aplikacyjnym APs (Application Processes) komunikację w sieci, czyli: <ul style="list-style-type: none">• Zasoby przetwarzające informację• Zasoby komunikacyjne Programy Javy komunikujące się przez sieć są tworzone w warstwie aplikacji
Warstwa Transportowa	TCP, UDP,...	Zapewnia poprawność przesyłanych danych między komputerami końcowymi
Warstwa Sieciowa	IP (Internet Protocol),	Nadzorowanie wiadomości przesyłanych między dwoma komputerami poprzez sieć
Warstwa Łącza danych	Device driver,...	Ustanawianie połączenia, nadzór nad bezbłędnym transferem bitów

„TCP (Transmission Control Protocol) to protokół oparty na połączeniu, które zapewnia niezawodny przepływ danych między dwoma komputerami.”

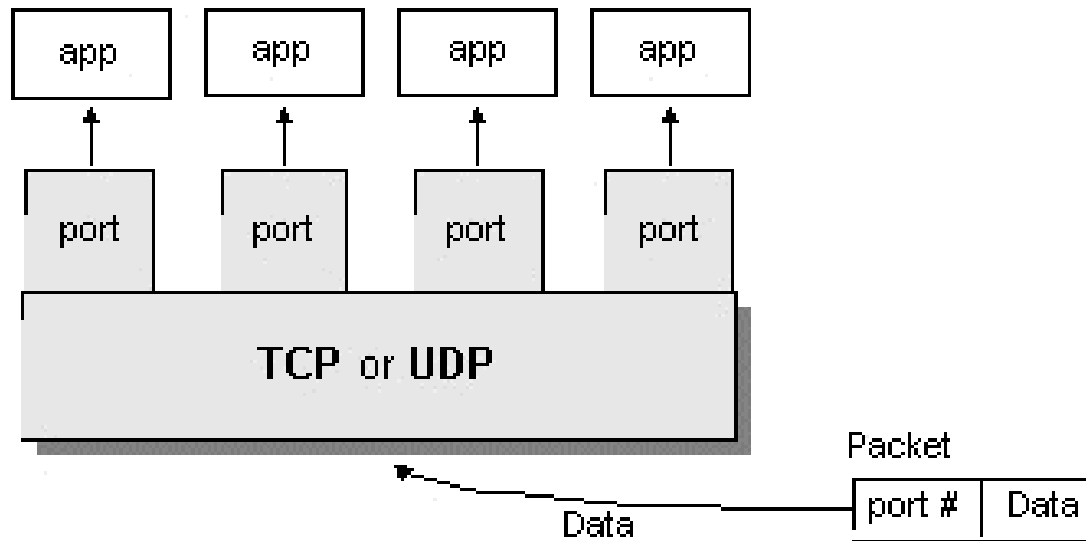
„UDP (User Datagram Protocol) jest protokołem, który wysyła niezależne pakiety danych, zwane datagramami, z jednego komputera na drugi bez gwarancji dotarcia tych pakietów. UDP nie jest oparty na połączeniu, jak TCP.”

- Bezstanowy protokół HTTP (definicja wg RFC 2616) jest zastosowany do przelania hipertekstowych dokumentów WWW w warstwie aplikacji wskazanych za pomocą: URL (*Uniform Resource Locator*), URI (*Uniform Resource Identifier*) lub URN (*Uniform Resource Name*). Powinien korzystać z niezawodnego kanału komunikacji punkt-punkt, jaki zapewnia protokół połączeniowy TCP, gdyż otrzymywane dane muszą wystąpić w tej samej kolejności, w której były przesyłane. HTTP jest również używany jako ogólny protokół komunikacji hipermedialnej między aplikacjami klienckimi i serwerami proxy, w tym obsługiwanych przez SMTP, NNTP, FTP, Gopher i protokoły WAIS.
- Komputery w sieci są identyfikowane przez [IPv4](#) (32-bitowe), [IPv5](#) i [IPv6](#) (128-bitowe) adresy IP, a procesy działające na tych komputerach są identyfikowane poprzez 16-bitowe numery portów (port reprezentuje kolejkę danych, które mają być dostarczane do danego procesu wykonującego pewien program). Program wykorzystujący protokół HTTP ma przydzielony domyślnie port o numerze 80.
- Programy odbierają lub wysyłają dane poprzez związane z numerami portów gniazda (sockets). Gniazdo jest definiowane przez warstwę transportową modelu ISO/OSI. Gniazdo jest końcowym punktem połączenia między programami wykonywanymi na komputerach sieciowych w systemie klient-serwer. Gniazdo pełni rolę interfejsu programowego umożliwiającego aplikacjom dostęp do protokołów TCP i UDP i wymianę danych poprzez sieć pracującą pod kontrolą protokołów TCP/IP.

- Program serwera wykonywany na konkretnej maszynie ma przypisane do pewnego numeru portu gniazdo (typu *ServerSocket*), poprzez które "nasłuchuje" za pomocą metody *accept* klasy *ServerSocket* ewentualnego żądania nawiązania łączności przez klienta (gniazdo typu *Socket*). Jest to możliwe, jeśli klient (program klienta) zna adres komputera sieciowego, na którym jest wykonywany serwer oraz numer portu, do którego serwer jest dołączony i utworzy odpowiednie gniazdo typu *Socket* (związane z numerem portu i adresem serwera), poprzez które może prowadzić komunikację z serwerem. Serwer akceptuje połączenie, a następnie tworzy dla klienta nowe gniazdo (typu *Socket*) ze zmiennymi wskazującymi na obiekty klas typu *InputStream* i *OutputStream*.
- W pakiecie *java.net* klasy *Socket* i *ServerSocket* służą do komunikacji w oparciu o protokół połączeniowy TCP
- W pakiecie *java.net* klasy *DatagramSocket*, *DatagramPacket* oraz *MulticastSocket* są wykorzystywane do komunikacji UDP.
- W pakiecie *java.net* klasy: *URL* oraz *URLConnection* i *URLEncoder* służą do nawiązania łączności z zasobami www.
- Komunikacja w sieci za pomocą klas z pakietu *java.net*, opiera się między innymi na implementacji gniazd.

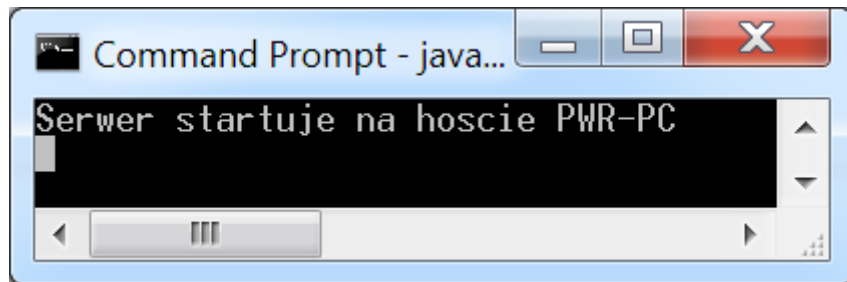


Podstawowa architektura typu Klient-serwer



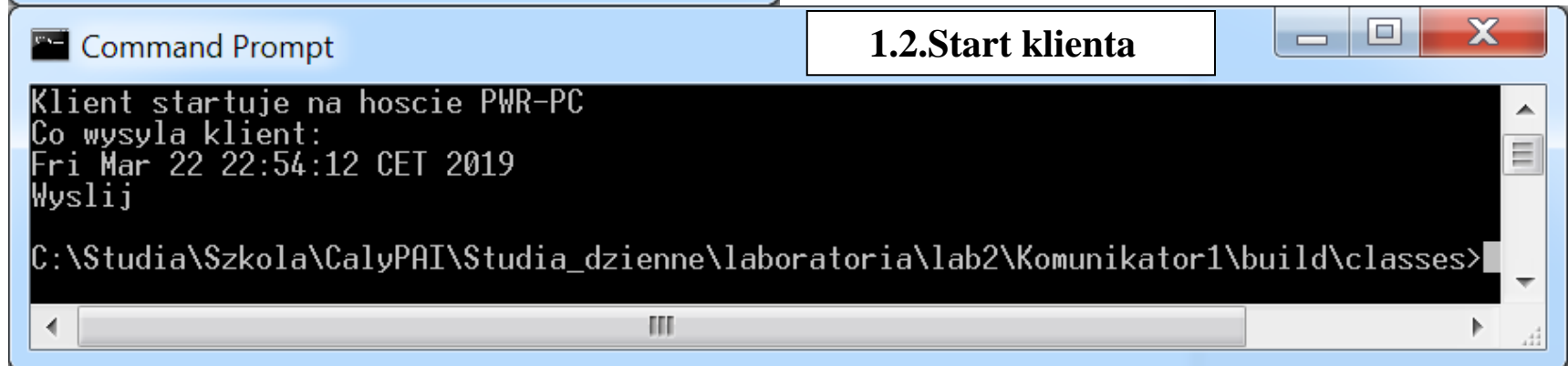
Datagram - pakiet przesyłanych danych

Przykład 1 - Aplikacja typu Klient-Serwer



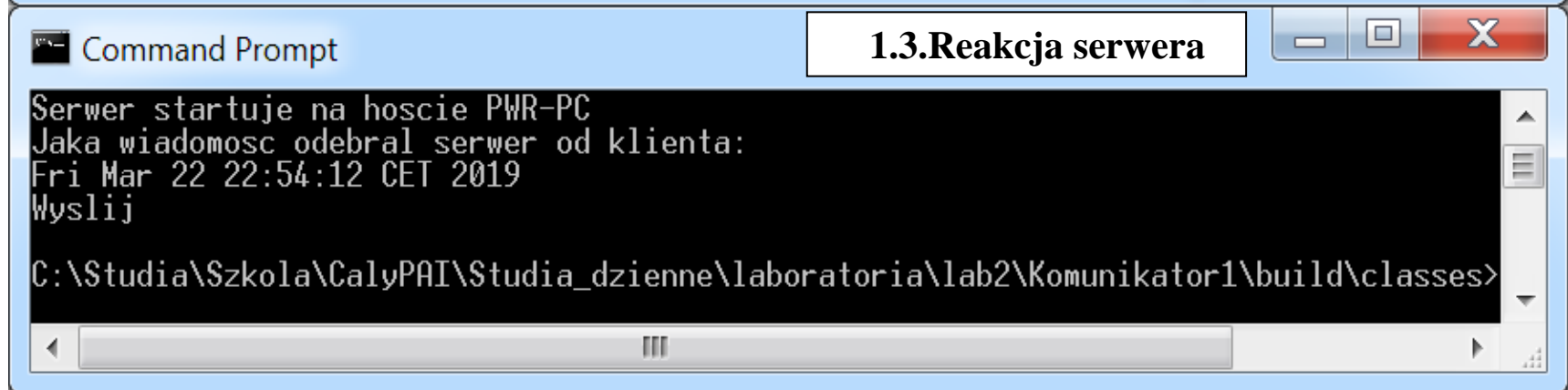
```
Command Prompt - java...  
Serwer startuje na hoscie PWR-PC
```

1.1.Start serwera



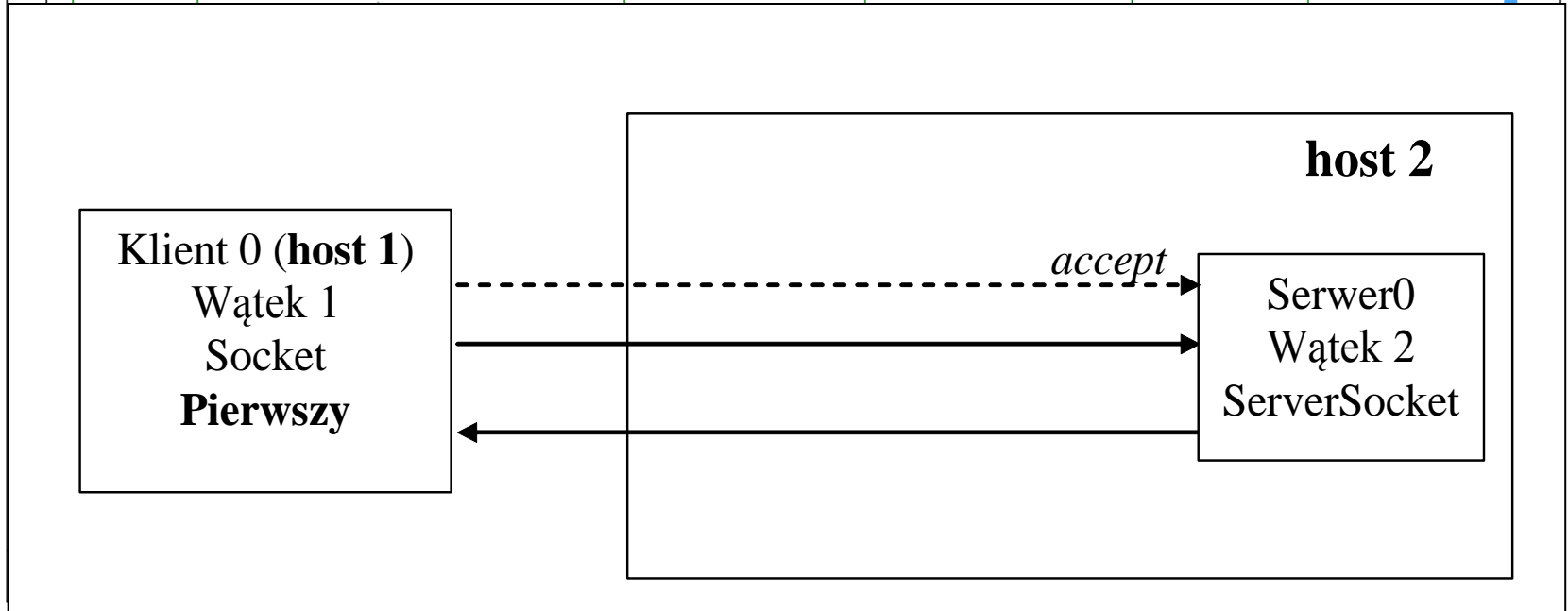
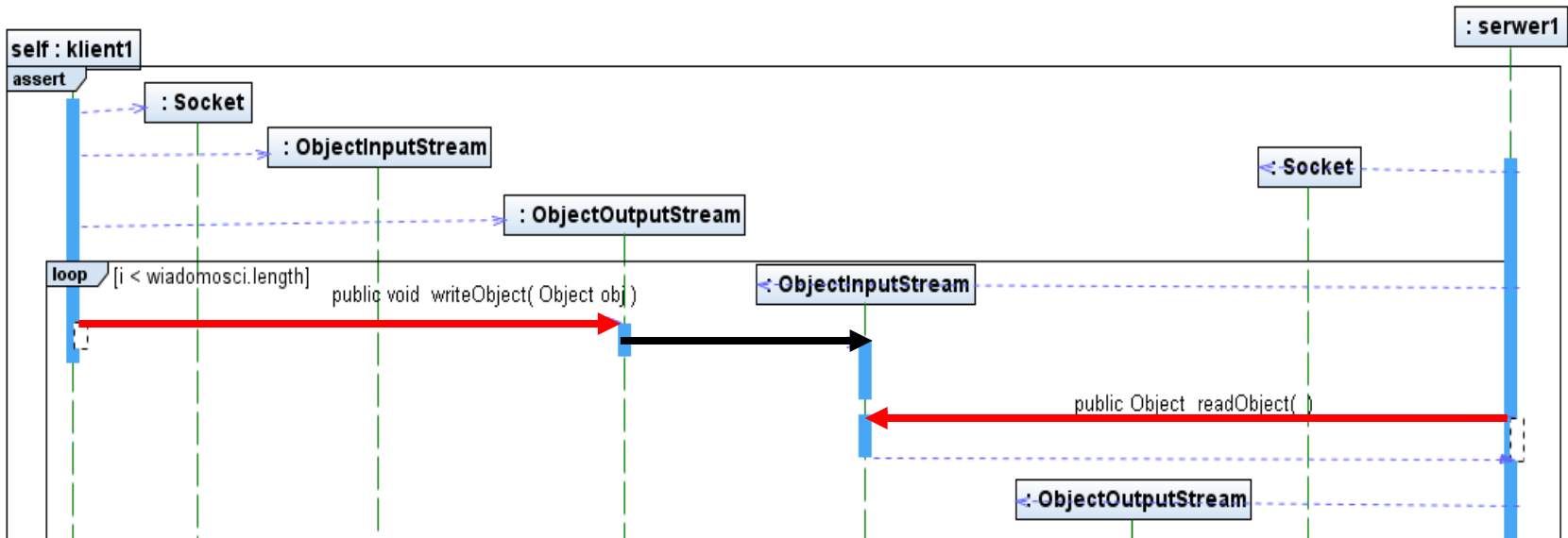
```
Command Prompt  
Klient startuje na hoscie PWR-PC  
Co wysyla klient:  
Fri Mar 22 22:54:12 CET 2019  
Wyslij  
C:\Studia\Szkola\CalyPAI\Studia_dzienne\laboratoria\lab2\Komunikator1\build\classes>
```

1.2.Start klienta



```
Command Prompt  
Serwer startuje na hoscie PWR-PC  
Jaka wiadomosc odebral serwer od klienta:  
Fri Mar 22 22:54:12 CET 2019  
Wyslij  
C:\Studia\Szkola\CalyPAI\Studia_dzienne\laboratoria\lab2\Komunikator1\build\classes>
```

1.3.Reakcja serwera



1.1. Instancje tej klasy, obowiązkowo implementującej interfejs Serializable są wykorzystane do przesyłania wiadomości przez sieć za pomocą strumieni

```
import java.io.*;  
import java.util.*;
```

```
public class Wiadomosc implements Serializable
```

```
{  
    String dane;  
    Date data;
```

```
public void zapiszWiadomosc()
```

```
{  
    data = new Date();  
    dane = "Wyslij"; }  
}
```

```
public void odczytajWiadomosc()
```

```
{  
    System.out.println(data);  
    System.out.println(dane); }  
}
```


1.2. Program serwera - ten program należy uruchomić jako pierwszy

```
import java.net.*;
import java.io.*;
public class serwer0 implements Runnable {
    public void run()
    { System.out.println("Serwer startuje na hoscie "+host);
      {try
        { try
          { gniazdo_klienta = serwer.accept();
            } catch (IOException e)
              { System.out.println("Nie mozna polaczyc sie z klientem "+e);
                System.exit(1); }
          wyjscie = new ObjectOutputStream(gniazdo_klienta.getOutputStream());
          wyjscie.flush();
          wejscie = new ObjectInputStream(gniazdo_klienta.getInputStream());
          wiadomosc = (Wiadomosc) wejscie.readObject();
          System.out.println("Jaka wiadomosc odebral serwer od klienta:");
          wiadomosc.odczytajWiadomosc();
          wejscie.close();
          wyjscie.close();
          gniazdo_klienta.close();
        } catch (Exception e)
          { System.out.println("Wyjatek serwera "+e); }
      }
    }
```

```

private int sPort;
private ServerSocket serwer;
private Socket gniazdo_klienta;
private ObjectOutputStream wyjscie;
private ObjectInputStream wejscie;
private String host;
private Wiadomosc wiadomosc;

public serwer0(int port_, String host_)
{ sPort = port_; host=host_;
  try
  { serwer = new ServerSocket(sPort);
  } catch(IOException e)
  { System.out.println(e); }
}

public static void main(String args[]) throws Exception
{
  String host_ = InetAddress.getLocalHost().getHostName();
  int Port = 5000;
  serwer0 s = new serwer0(Port, host_);
  Thread t = new Thread(s);
  t.start();
}
}

```

1.2. Program klienta - ten program należy uruchomić jako drugi

```
public class klient0 implements Runnable
```

```
{
```

```
    private int port;
```

```
    private Socket gniazdo_klienta;
```

```
    private ObjectOutputStream wyjście;
```

```
    private ObjectInputStream wejście;
```

```
    private String host;
```

```
    private Wiadomosc wiadomosc;
```

```
klient0(String host_, int port_)
```

```
{    host = host_;  
    port = port_; }
```

```
public static void main(String args[]) throws Exception
```

```
{
```

```
    //String s = InetAddress.getLocalHost().getHostName();
```

```
    String s = InetAddress.getByName("kruk").getHostName();
```

```
    Klient0 k= new klient0(s, 5000);
```

```
    Thread t = new Thread(k);
```

```
    t.start();
```

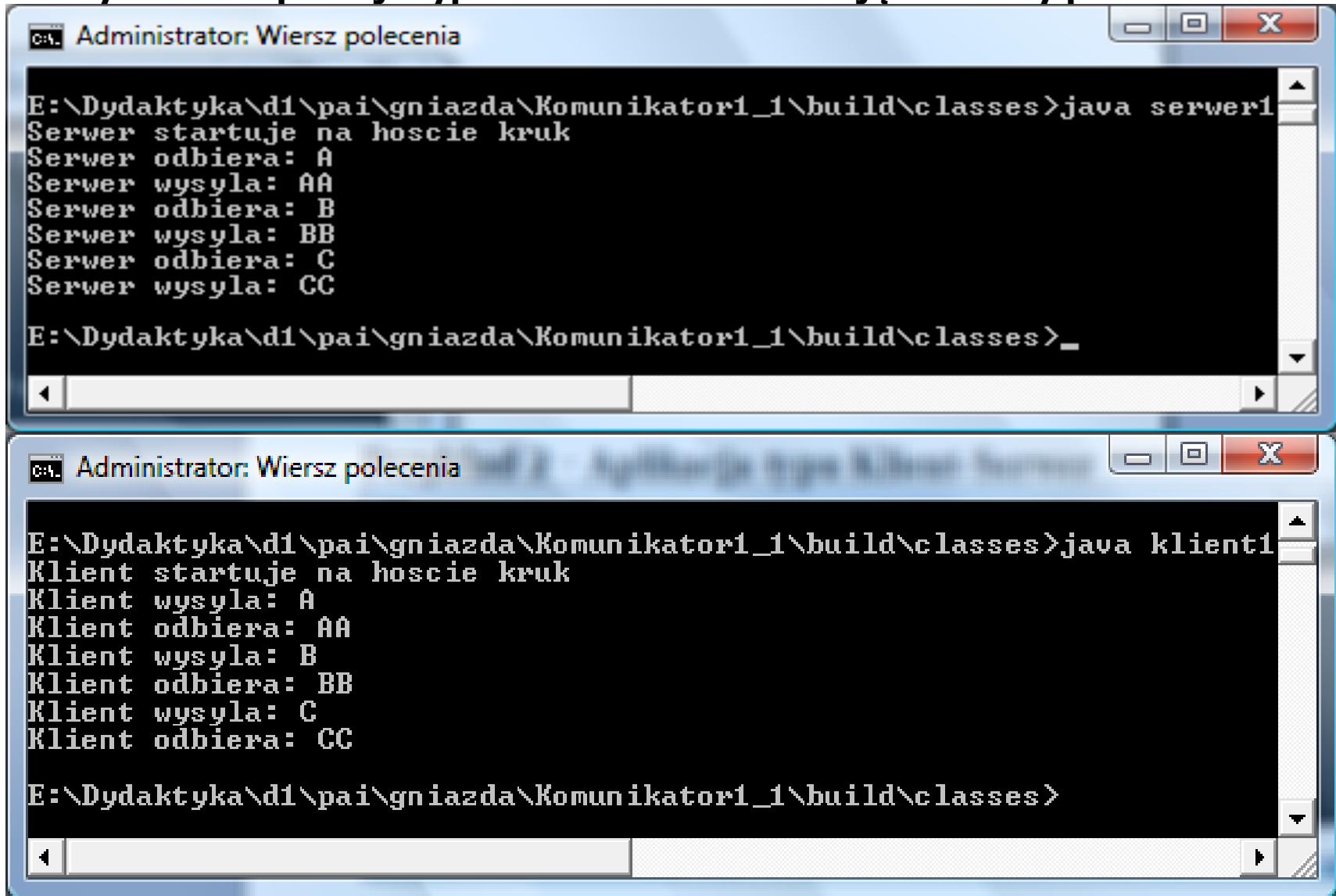
```
}
```

```

public void run()
{
    try
    {
        gniazdo_klienta = new Socket (host,port);
        wejscie = new ObjectInputStream(gniazdo_klienta.getInputStream());
        wyjscie = new ObjectOutputStream(gniazdo_klienta.getOutputStream());
        wyjscie.flush();
        System.out.println("Klient startuje na hoscie "+InetAddress.getLocalHost().getHostName());
        wiadomosc = new Wiadomosc();
        wiadomosc.zapiszWiadomosc();
        System.out.println("Co wysyla klient: ");
        wiadomosc.odczytajWiadomosc();
        wyjscie.writeObject(wiadomosc);
        gniazdo_klienta.close();
        wyjscie.close();
        wejscie.close();
    } catch (Exception e) {
        System.out.println("Wyjatek klienta "+e);
    }
}
}

```

Przykład 2 - Aplikacja typu Klient-Serwer realizująca zadany protokół

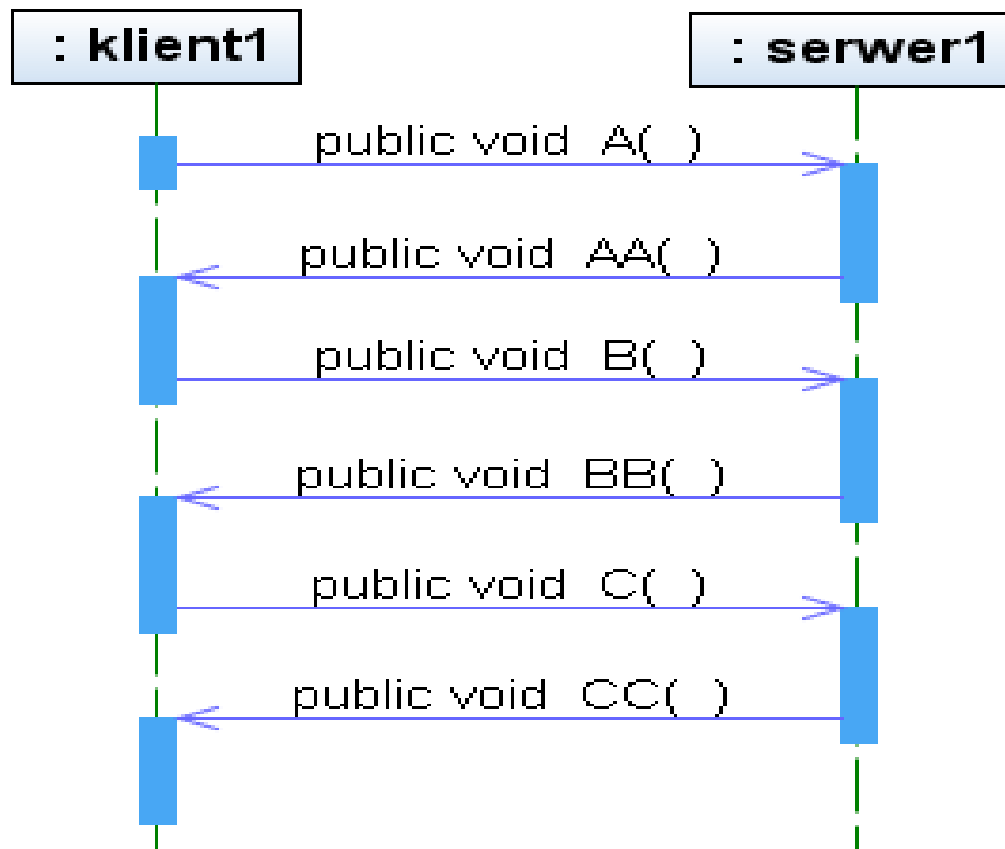


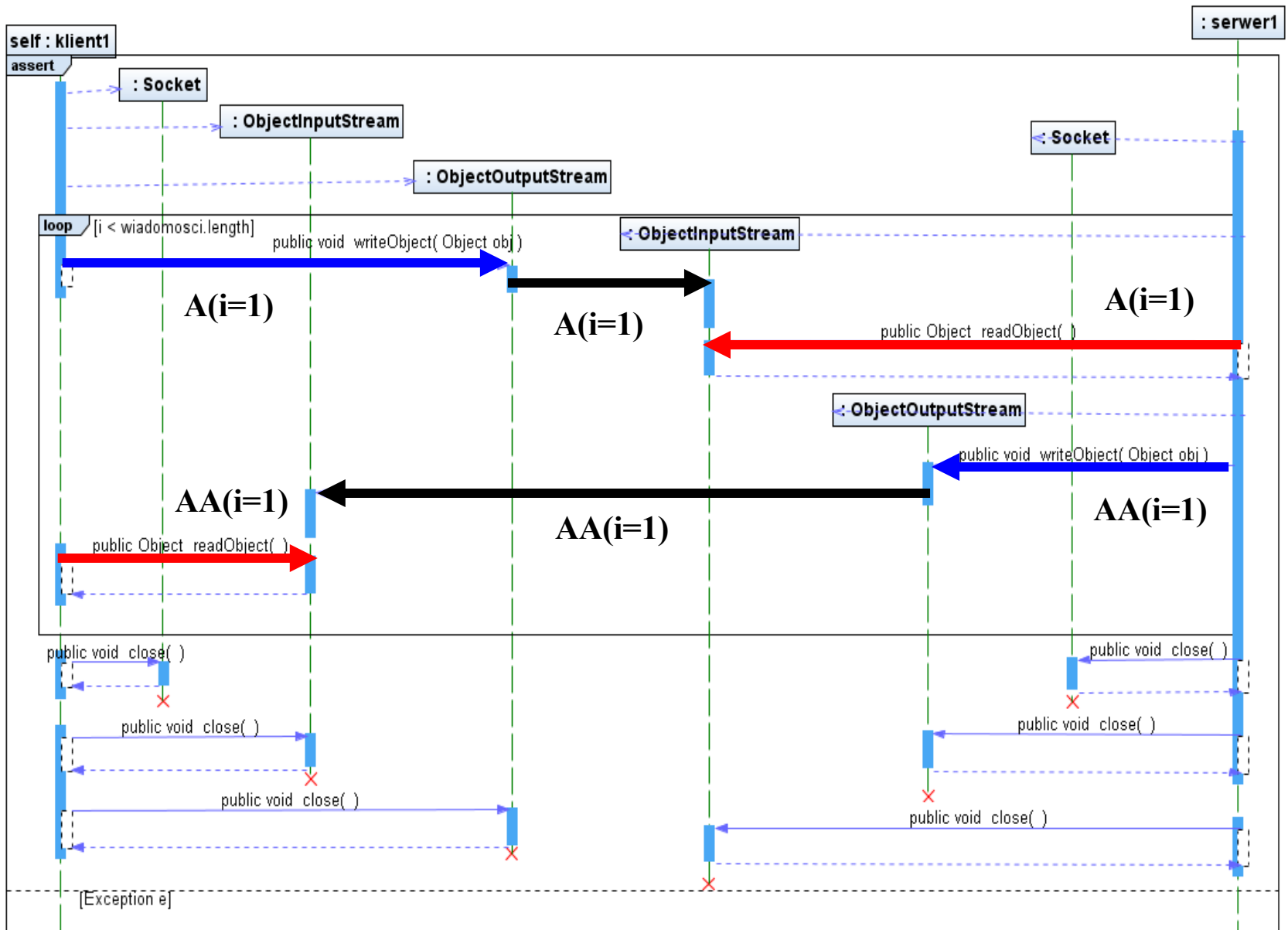
```
Administrator: Wiersz polecenia
E:\Dydaktyka\d1\pai\gniazda\Komunikator1_1\build\classes>java serwer1
Serwer startuje na hoscie kruk
Serwer odbiera: A
Serwer wysyla: AA
Serwer odbiera: B
Serwer wysyla: BB
Serwer odbiera: C
Serwer wysyla: CC

E:\Dydaktyka\d1\pai\gniazda\Komunikator1_1\build\classes>_

Administrator: Wiersz polecenia
E:\Dydaktyka\d1\pai\gniazda\Komunikator1_1\build\classes>java klient1
Klient startuje na hoscie kruk
Klient wysyla: A
Klient odbiera: AA
Klient wysyla: B
Klient odbiera: BB
Klient wysyla: C
Klient odbiera: CC

E:\Dydaktyka\d1\pai\gniazda\Komunikator1_1\build\classes>
```





2.1 Program serwera - ten program należy uruchomić jako pierwszy

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
```

```
public class serwer1 implements Runnable {
    private int sPort;
    private ServerSocket serwer;
    private Socket gniazdo_klienta;
    private ObjectOutputStream wyjscie;
    private ObjectInputStream wejscie;
    private String host, wiadomosc;
    private String wiadomosci []={"AA", "BB", "CC"};

    public serwer1(int port_, String host_) {
        sPort = port_;
        host = host_;
        try {
            serwer = new ServerSocket(sPort);
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```



```

public void run() {
    System.out.println("Serwer startuje na hoscie " + host);
    try {
        try {
            gniazdo_klienta = serwer.accept();
        } catch (IOException e) {
            System.out.println("Nie mozna polaczyc sie z klientem " + e);
            System.exit(1);
        }
        //przerwanie pracy serwera nie jest zalecane w praktyce
        wyjscie = new ObjectOutputStream(gniazdo_klienta.getOutputStream());
        wyjscie.flush();
        wejscie = new ObjectInputStream(gniazdo_klienta.getInputStream());
        for (int i = 0; i < wiadomosci.length; i++) {
            wiadomosc = (String) wejscie.readObject();
            System.out.println("Serwer odbiera: "+wiadomosc);
            wyjscie.writeObject(wiadomosci[i]);
            System.out.println("Serwer wysyla: "+wiadomosci[i]);
        }
        wejscie.close();
        wyjscie.close();
        gniazdo_klienta.close();
    } catch (Exception e) {
        System.out.println("Wyjatek serwera " + e);    }
}

```

```
public static void main(String args[]) throws Exception
{
    String host_ = InetAddress.getLocalHost().getHostName();
    int Port = 15000;
    serwer1 s2 = new serwer1(Port, host_);
    Thread t = new Thread(s2);
    t.start();
}
}
```

2.2. Program klienta – ten program należy wywołać jako drugi

```
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.net.InetAddress;  
import java.net.Socket;
```

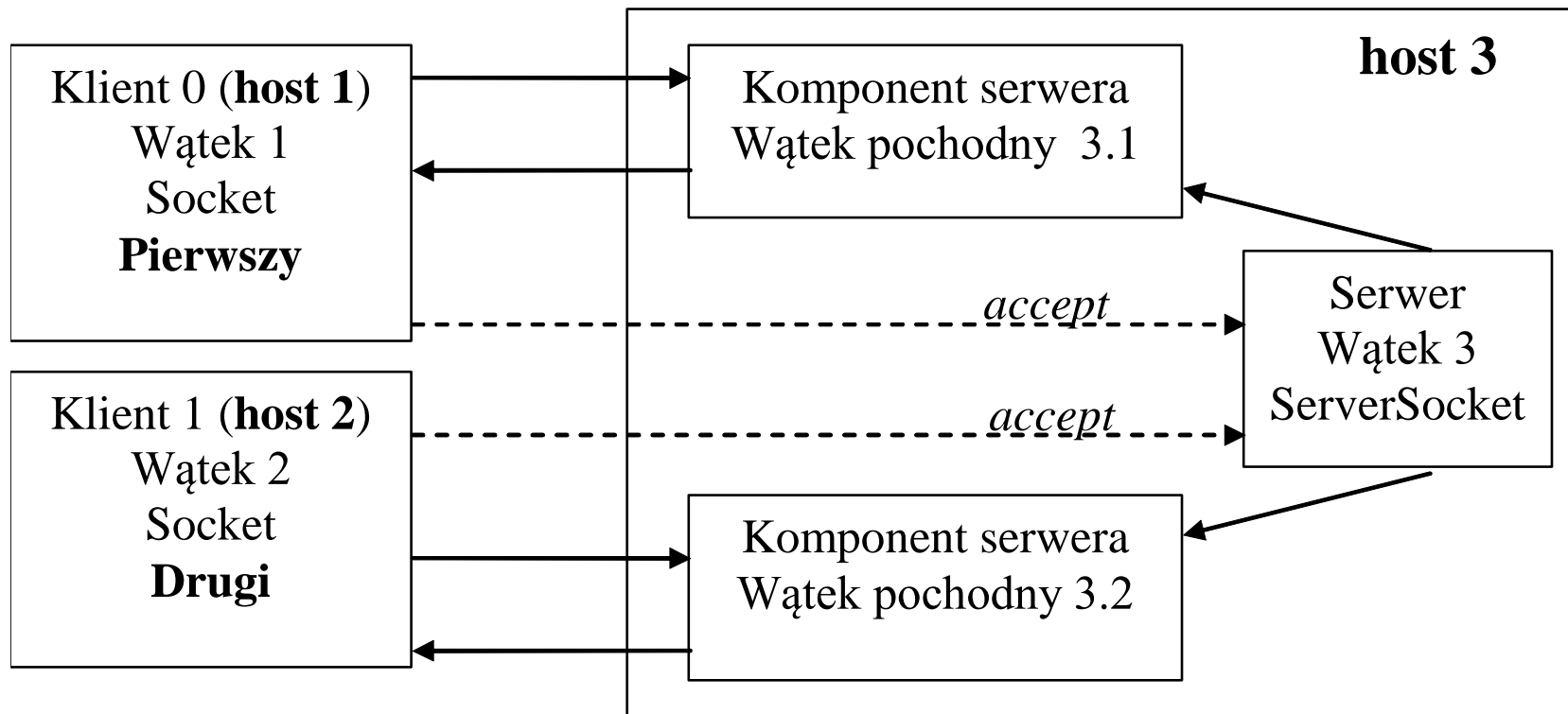
```
public class klient1 implements Runnable {  
public void run() {  
    try { gniazdo_klienta = new Socket(host, port);  
        wejscie = new ObjectInputStream(gniazdo_klienta.getInputStream());  
        wyjscie = new ObjectOutputStream(gniazdo_klienta.getOutputStream());  
        wyjscie.flush();  
        System.out.println("Klient startuje na hoscie " + host);  
        for (int i = 0; i < wiadomosci.length; i++) {  
            wyjscie.writeObject(wiadomosci[i]);  
            System.out.println("Klient wysyla: " + wiadomosci[i]);  
            wiadomosc = (String) wejscie.readObject();  
            System.out.println("Klient odbiera: " + wiadomosc);  
        }  
        gniazdo_klienta.close();  
        wyjscie.close();  
        wejscie.close();  
    } catch (Exception e) {  
        System.out.println("Wyjatek klienta " + e);    }  
    }  
}
```

```
private int port;  
private Socket gniazdo_klienta;  
private ObjectOutputStream wyjscie;  
private ObjectInputStream wejscie;  
private String host, wiadomosc;  
private String wiadomosci[] = {"A", "B", "C"};
```

```
klient1(String host_, int port_)  
{  
    host = host_;  
    port = port_;  
}
```

```
public static void main(String args[]) throws Exception  
{  
    String s = InetAddress.getLocalHost().getHostName();  
    //String s = InetAddress.getByName("kloc").getHostName();  
    klient1 k2 = new klient1(s, 15000);  
    Thread t = new Thread(k2);  
    t.start();  
}
```

Przykład 3 - Aplikacja typu Klient-Serwer. Program wielowątkowy.



3.1 Program serwera - ten program należy uruchomić jako pierwszy

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * @author kruczkiewicz
 */
```

```
public class serwer2 implements Runnable {
```

```
    public serwer2(int port_, String host_)
    {
        sPort = port_;
        host = host_;
        try {
            serwer = new ServerSocket(sPort);
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

```

public void run() {
    int numer = 0;
    Socket s = null;
    ObjectOutputStream output;
    ObjectInputStream input;
    System.out.println("Serwer startuje na hoscie " + host);
    while (true) {
        try {
            s = serwer.accept();
            if (s == null) continue;
            output = new ObjectOutputStream(s.getOutputStream());
            output.flush();
            input = new ObjectInputStream(s.getInputStream());
            komponent_klienta_po_stronie_serwera komp_klienta =
                new komponent_klienta_po_stronie_serwera(s,input,output);
            Thread watek_komponentu_klienta = new Thread(komp_klienta);
            watek_komponentu_klienta.start();
        } catch (IOException e) {
            System.out.println("Nie mozna polaczyc sie z klientem " + e);
            System.exit(1);
        } // przerwanie pracy serwera nie jest zalecane w praktyce
    }
}

```

```
private int sPort;  
private ServerSocket serwer;  
private String host;
```

```
public static void main(String args[]) throws Exception  
{  
    String host_ = InetAddress.getLocalHost().getHostName();  
    serwer s2 = new serwer2(5000, host_);  
    Thread t = new Thread(s2);  
    t.start();  
}  
}
```


3.2. Program komponentu klienta po stronie serwera – uruchamiany po wykonaniu metody accept

```
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.net.Socket;
```

```
public class komponent_klienta_po_stronie_serwera implements Runnable
```

```
{  
    private Socket s;  
    private ObjectOutputStream output;  
    private ObjectInputStream input;  
    private String nazwa;
```

```
public komponent_klienta_po_stronie_serwera(Socket s,  
        ObjectInputStream input, ObjectOutputStream output)
```

```
{  
    this.s = s;  
    this.input = input;  
    this.output = output;  
}
```

```

public void run()
{
    String m1, m2;
    try {
        nazwa = (String) input.readObject();
        m1 = nazwa;
        while (true)
        {
            if (m1.equals(nazwa + ": "+"czesc")) break;
            System.out.println("Odebrano wiadomosc od klienta: " + m1 + "\n");
            m2 = WEWY.weString("Podaj wiadomosc dla klienta "+ nazwa + ": ");
            output.writeObject(nazwa + ": " + m2);
            m1 = (String) input.readObject();
        }
        input.close();
        output.close();
        s.close();
    } catch (Exception e) {
        System.out.println("Wyjatek komponentu klienta po stronie serwera "+e);
    }
}
}

```

3.3. Program klienta – powinien być uruchomiony jako drugi

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.Socket;
public class klient2 implements Runnable {
    private int port;
    private Socket s;
    private ObjectOutputStream output;
    private ObjectInputStream input;
    private String host, nazwa;

    klient2(String host_, int port_) {
        host = host_;
        port = port_;
    }

    public static void main(String args[]) throws Exception
    {
        String s = InetAddress.getLocalHost().getHostName();
        klient2 k2 = new klient2(s, 5000);
        Thread t = new Thread(k2);
        t.start();
    }
}
```

```

public void run() {
    String m1, m2;
    try {
        s = new Socket(host, port);
        input = new ObjectInputStream(s.getInputStream());
        output = new ObjectOutputStream(s.getOutputStream());
        output.flush();
        System.out.println("Klient startuje na hoscie "+s.getLocalHost().getHostName());
        nazwa = WEWY.weString("Przedstaw sie: ");
        output.writeObject(nazwa);
        do {
            m1 = (String) input.readObject();
            System.out.println("\n" + "Dane odebrane od serwera: " + m1);
            m2 = WEWY.weString("Podaj wiadomosc dla serwera: ");
            output.writeObject(nazwa + ": " + m2);
        } while (!m2.equals("czesc"));
        output.close();
        input.close();
        s.close();
    } catch (Exception e) {
        System.out.println("Wyjatek klienta " + e);    }
}
}

```

3.4. Klasa realizująca wprowadzanie danych z klawiatury za pomocą strumieni

```
public class WEWY {  
  
    static String weString(String menu)  
    {  
        InputStreamReader wejście = new InputStreamReader( System.in );  
        BufferedReader bufor = new BufferedReader( wejście );  
        System.out.print(menu);  
        try  
        {  
            return bufor.readLine();  
        }  
        catch (IOException e)  
        {  
            System.err.println("Bład IO String");  
            return ""; }  
    }  
}
```

```
Administrator: Wiersz polecenia - java serwer2
E:\Dydaktyka\d1\pai\gniazda\komunikator2\build\classes>java serwer2
Serwer startuje na hoscie kruk
Odebrano wiadomosc od klienta: Pierwszy

Podaj wiadomosc dla klienta Pierwszy: Witaj Pierwszy
Odebrano wiadomosc od klienta: Drugi

Podaj wiadomosc dla klienta Drugi: Witaj Drugi
Odebrano wiadomosc od klienta: Pierwszy: Czy masz czas na rozmowe?

Podaj wiadomosc dla klienta Pierwszy: Mam malo czasu.
Odebrano wiadomosc od klienta: Drugi: Czy mozesz rozmawiac?

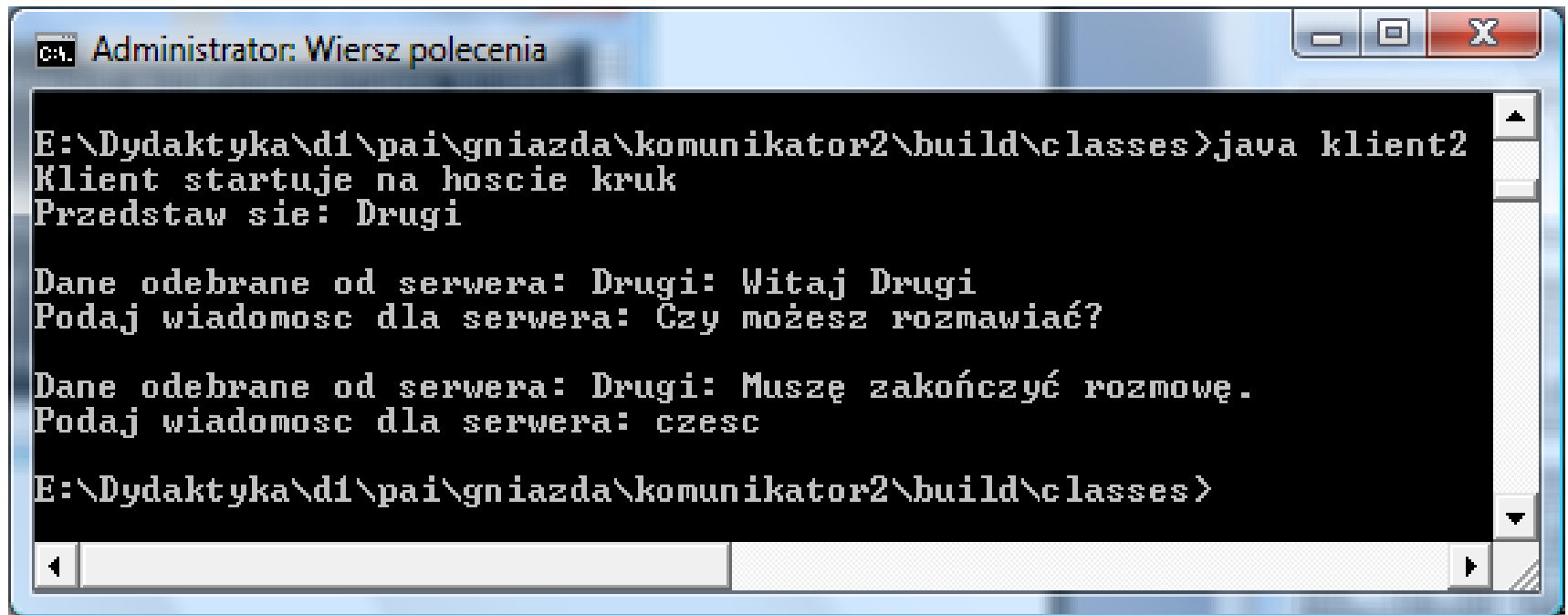
Podaj wiadomosc dla klienta Drugi: Musze zakonczyc rozmowe.
```

```
Administrator: Wiersz polecenia
E:\Dydaktyka\d1\pai\gniazda\komunikator2\build\classes>java klient2
Klient startuje na hoscie kruk
Przedstaw sie: Pierwszy

Dane odebrane od serwera: Pierwszy: Witaj Pierwszy
Podaj wiadomosc dla serwera: Czy masz czas na rozmowe?

Dane odebrane od serwera: Pierwszy: Mam malo czasu.
Podaj wiadomosc dla serwera: czesc

E:\Dydaktyka\d1\pai\gniazda\komunikator2\build\classes>_
```



```
Administrator: Wiersz polecenia
E:\Dydaktyka\d1\pai\gniazda\komunikator2\build\classes>java klient2
Klient startuje na hoscie kruk
Przedstaw sie: Drugi

Dane odebrane od serwera: Drugi: Witaj Drugi
Podaj wiadomosc dla serwera: Czy mozesz rozmawiac?

Dane odebrane od serwera: Drugi: Musze zakonczyc rozmowe.
Podaj wiadomosc dla serwera: czesc

E:\Dydaktyka\d1\pai\gniazda\komunikator2\build\classes>
```