Java – programowanie sieciowe

Podstawowe pojęcia dotyczące sieci

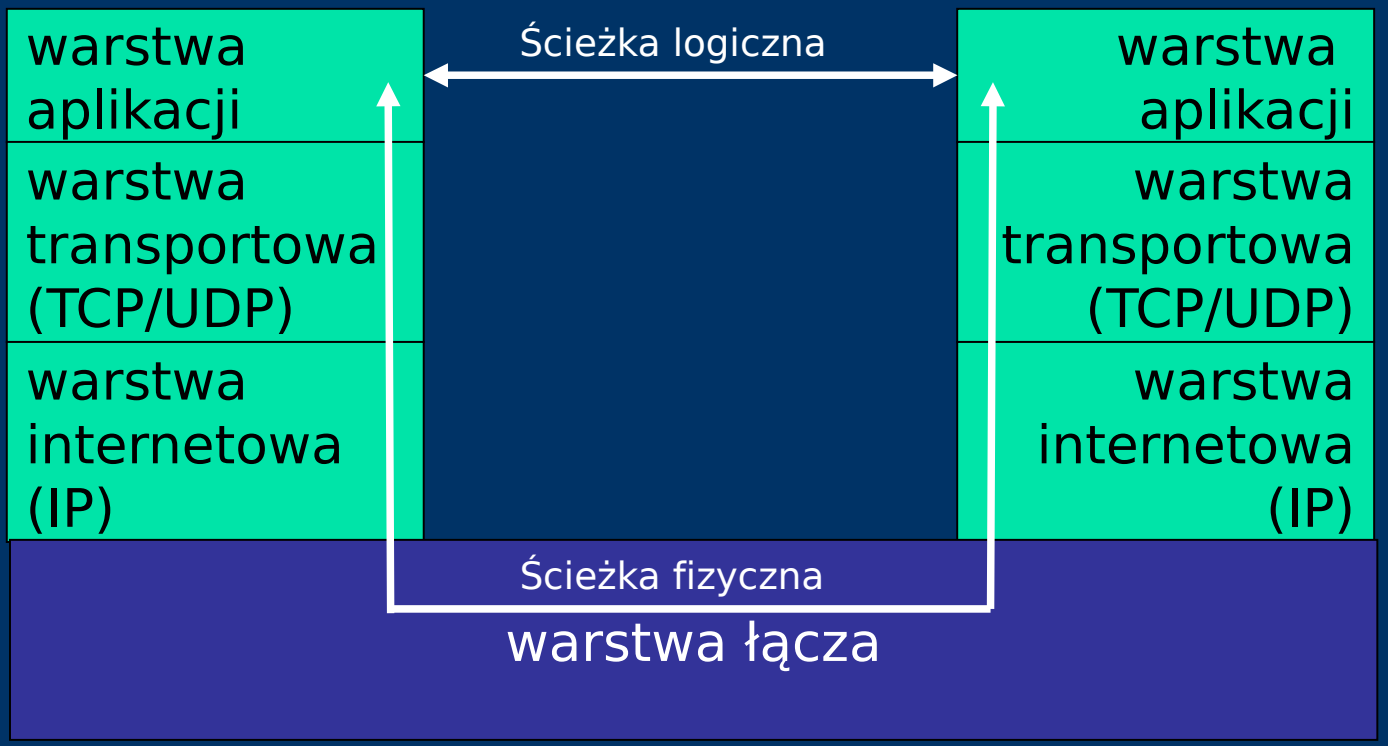
- Sieć to zbiór komputerów i innych urządzeń, które mogą się ze sobą komunikować w czasie rzeczywistym za pomocą transmisji danych.
- Każda maszyna (komputer, ruter, drukarka, terminal) znajdująca się w sieci nazywa się węzłem. Węzły, które są w pełni funkcjonalnymi komputerami nazywane są hostami.
- Każdy węzeł w sieci ma swój adres.

Podstawowe pojęcia dotyczące sieci

- Wszystkie współczesne sieci komputerowe są sieciami komutacji pakietów.
- Każdy pakiet oprócz fragmentu danych zawiera informację o tym kto i dokąd go wysłał.
- Zestaw reguł według których komputery i urządzenia komunikują się ze sobą nazywa się protokołem.

Warstwy sieci

- Przesyłanie danych przez sieć to skomplikowana operacja logistyczna.



Warstwy sieci

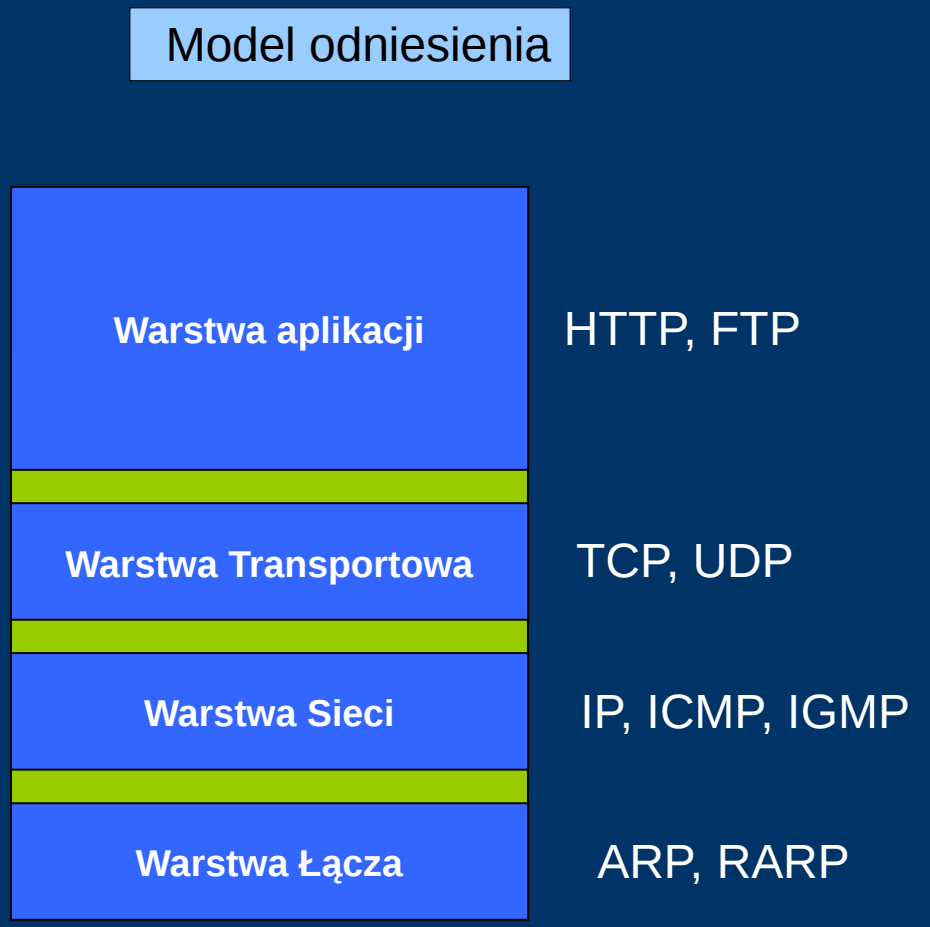
- Warstwa łącza definiuje konkretny interfejs sieciowy i przesyła datagramy IP fizycznym łączem (do sieci lokalnej i w świat) – Java nie ma dostępu do tej warstwy.
- Warstwa internetowa odpowiada za grupowanie danych w pakiety oraz za schemat adresowania, w którym różne maszyny mogą się odnaleźć – Java „zna” tylko protokół IP dla tej warstwy (jest on najpowszechniej stosowany).

Warstwy sieci

- Warstwa transportowa odpowiada za to, aby pakiety były odbierane w tej samej kolejności w jakiej zostały wysłane, oraz aby żaden z nich nie został uszkodzony ani zagubiony – Java umie obsłużyć dwa protokoły tej warstwy:
  - TCP (ang. Transmission Control Protocol) niezawodny,
  - UDP (ang. User Datagram Protocol) zawodny ale szybki.
- Warstwa aplikacji dostarcza dane użutkownikowi – znane protokoły tej warstwy to HTTP, SMTP, POP, IMAP, FTP, NFS, NNTP oraz wiele innych.

Java a warstwy sieci

- W Javie aplikacje sieciowe mogą być tworzone w warstwie aplikacji bez konieczności obsługi warstw UDP/TCP
- Obsługa sieci w Java jest niezależna od platformy

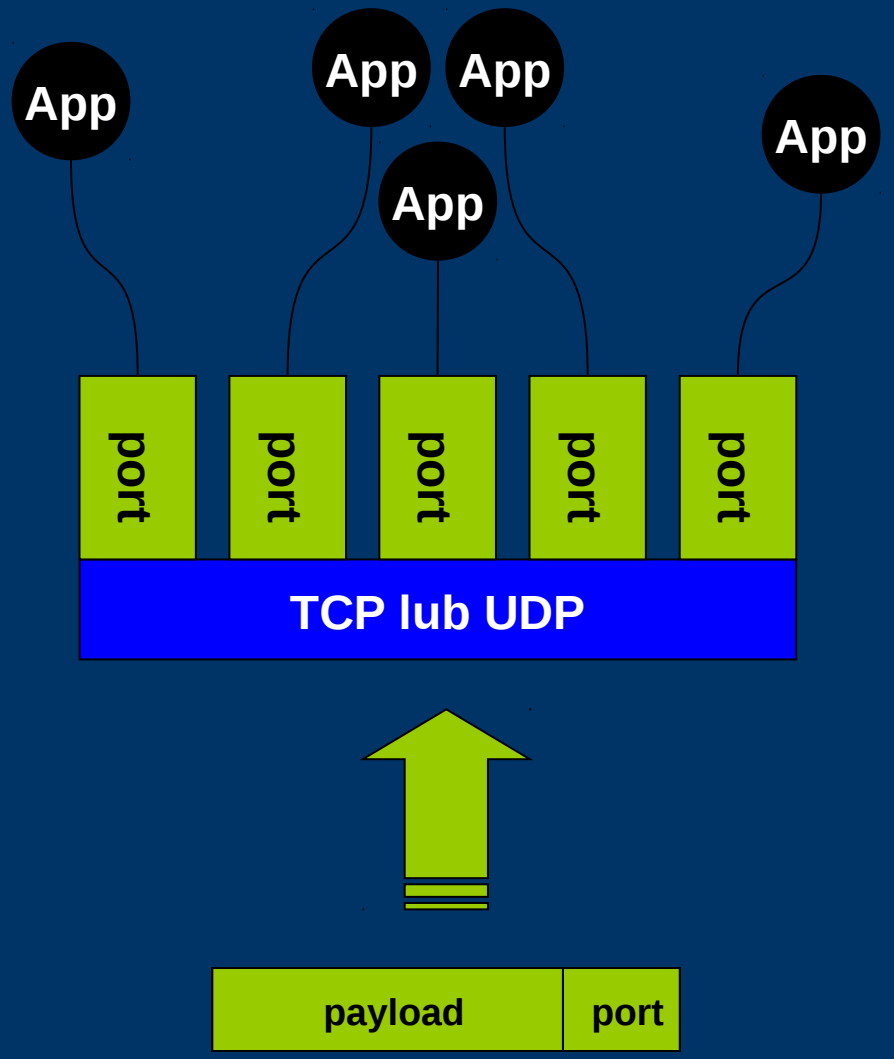


Adresy IP

- Protokół Internetu IP jest niezależny od platformy, automatycznie wyznacza trasę rutingu.
- Każdy komputer w sieci IP jest identyfikowany za pomocą swojego unikatowego 32-bitowego (IPv4) albo 128-bitowego (IPv6) adresu.
- DNS (ang. Domain Name System) to usługa, która tłumaczy nazwy mnemoniczne adresów na nazwy liczbowe.
- Pakiety, które przychodzą do określonego hosta mogą trafiać do różnych aplikacji czy serwisów dzięki portom. Jest ich 65535 dla protokołów TCP i UDP (porty 1-1023 są zarezerwowane dla usług standardowych).

Porty

- Przeznaczenie danych w Internecie określane jest:
  - adresem IP (32 bity),
  - numerem portu (16 bitów).
- Porty wykorzystywane są do kojarzenia aplikacji z danymi przychodzącymi na interfejs.



Adresy IP

- W pakiecie java.net jest zdefiniowana klasa InetAddress, która reprezentuje adres IP.
- Klasa InetAddress ma dwie podklasy Inet4Address i Inet6Address reprezentujące odpowiednio adresy protokołu internetowego w standardach IPv4 i IPv6.
- Klasa ta jest wykorzystywana przez inne klasy sieciowe: URL, Socket, ServerSocket, DatagramSocket, DatagramPacket.

Adresy IP

- Klasa InetAddress pozwala tworzyć obiekty tych klas za pomocą metod statycznych:
  - getByAddress (byte[] addr)
  - getByName (String host)
  - getLocalHost ()
- Z obiektu InetAddress można wydobyć szczegółowe informacje o adresie IP za pomocą metod:
  - getHostAddress ()
  - getHostName ()
  - getCanonicalHostName ()
  - toString ()
  - isAnyLocalAddress ()
  - isReachable (int timeout)

Adres URL (1)

- URL (ang. Uniform Resource Locator) to referencja do zasobu w Internecie.
- URL składa się z nazwy protokołu i nazwy zasobu, na przykład:

```
URL ul = new URL("http://www.koszalin.pl/");
```

- Pełna nazwa zasobu może składać się z nazwy hosta, ścieżki, pliku, portu, referencji i zapytania.

Adres URL (2)

- Metody do parsowania URL: getProtocol(), getAuthority(), getHost(), getPort(), getPath(), getQuery(), getFile(), getRef()
- Otwieranie strumienia na URL: openStream() – zwraca srumień we/wy
- Połączenie z zasobami z wykorzystaniem określonego protokołu dla URL: openConnection() zwraca URLConnection, istnieje możliwość rzutowania na konkretny protokół java.net.HttpURLConnection

Klasa URL

- Klasa URL reprezentuje adres URL w sieci WWW. Obiekt URL można utworzyć na kilka sposobów:
  new URL (String spec)
  new URL (String prot, String host, String file)
  new URL (String prot, String host, int port, String file)
  new URL (URL context, String spec)
- Podczas tworzenia obiektu URL może zostać zgłoszony wyjątek MalformedURLException.

Klasa URL

- Klasa URL udostępnia wiele metod odczytywania parametrów adresu URL:
  - getProtocol ()
  - getHost ()
  - getPort ()
  - getPath ()
  - getQuery ()
  - getRef ()

Czytanie z obiektu URL

Metoda *InputStream openStream ()* klasy URL potrafi nawiązać połączenie z podanym zasobem w sieci i otworzyć dla niego strumień do czytania:

```
URL url = new URL(URLName);
BufferedReader in = new BufferedReader(
        new InputStreamReader(
        url.openStream()));
String line;
while ((line=in.readLine())!=null)
{
// kolejne linie dokumentu są do dyspozycji
}
in.close();
```

Klasa URLConnection

- Klasa URLConnection ma zapewnić łatwiejszą w użyciu, wysokopoziomową abstrakcję połączenia sieciowego.
- Klasa URLConnection wykorzystuje klasę Socket do zapewnienia łączności sieciowej.
- Klasa URLConnection jest mocno związana z protokołem HTTP i zakłada, że każdy przesyłany plik jest poprzedzony nagłówkiem MIME.

Klasa URLConnection

Otwieranie połączeń URLConnection:

```
// konstrukcja URL
URL url = new URL("http://…");
// pozyskanie URLConnection
URLConnection uc = url.openConnection();
// konfiguracja URLConnection …
// odczytanie pól nagłówka …
// pobranie strumienia wejściowego …
// pobranie strumienia wyjściowego …
// zamknięcie połączenia …
```

Klasa URLConnection

Czytanie danych:

```
URL url = new URL("http://…");
URLConnection uc = url.openConnection();
InputStream is = uc.getInputStream();
```

Pisanie danych:

```
URL url = new URL("http://…");
URLConnection uc = url.openConnection();
uc.setDoOutput();
InputStream is = uc.getInputStream();
OutputStream is = uc.getOutputStream();
```

Klasa URLConnection

Serwery HTTP dostarczają sporo informacji w nagłówkach MIME. Klasa URLConnection posiada metody do odczytywania informacji z nagłówka MIME:

- getContentType()
- getContentLength()
- getContentEncoding()
- getDate()
- getExpiration()
- getLastModified()

Klasa URLConnection posiada też kilka ogólnych metod do odczytywania informacji z nagłówka MIME:

- getHeaderFieldKey(int)
- getHeaderField(int)
- getHeaderField(String)

Klasa URLConnection

Klasa URLConnection może konfigurować połączenie za pomocą metod:

- setDoInput(boolean)
- setDoOutput(boolean)
- setAllowUserInteraction(boolean)
- setUseCaches(boolean)
- setIfModifiedSince(long)

Klasa URLConnection może pobrać treść (obiekt Object) metodą getContent(). Dodatkowe metody przydatne do pracy z adresami URL typu http są w podklasie HttpURLConnection zawiera pewne.

Wyświetlanie zawartości strony na standardowe wyjście

```
import java.net.*;
import java.io.*;
public class URLReader {
   public static void main(String[ ] args) throws Exception {
      URL koszalin = new URL("http://www.koszalin.pl/");
      BufferedReader in = new BufferedReader(
        new InputStreamReader( koszalin.openStream()));
      String inputLine;
      while ((inputLine = in.readLine()) != null)
          System.out.println(inputLine);
          in.close();
      }
   }
```

Przykład: połączenie dla określonego protokołu

```
import java.net.*;
import java.io.*;
  public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL koszalin = new URL("http://www.koszalin.com/");
        URLConnection koszalinConnect = koszalin.openConnection();
        BufferedReader in = new BufferedReader(
             new InputStreamReader(koszalinConnect.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Pisanie pod wskazany URL

Kroki wymagane do stworzenia klienta piszącego pod wskazany URL:

1. utworzenie URL
2. pobranie obiektu URLConnection
3. ustawienie właściwości pisania do strumienia
4. otwarcie połączenia do zasobów
5. pobranie strumienia wyjściowego z obiektu połączenie
6. zapis do strumienia wyjściowego
7. zamknięcie strumienia wyjściowego

Zapis do strumienia wyjściowego

```
import java.io.*; import java.net.*;
public class Reverse {
    public static void main(String[ ] args) throws Exception {
      ...
      URL url = new URL(args[0]); // (1)
        URLConnection connection = url.openConnection(); //(2)
        connection.setDoOutput(true); //(3)
      OutputStreamWriter out = new OutputStreamWriter(
                          connection.getOutputStream()); //(4)
       out.write("string=" + stringToReverse); //(5)
       out.close();
      ...
    }
}
```

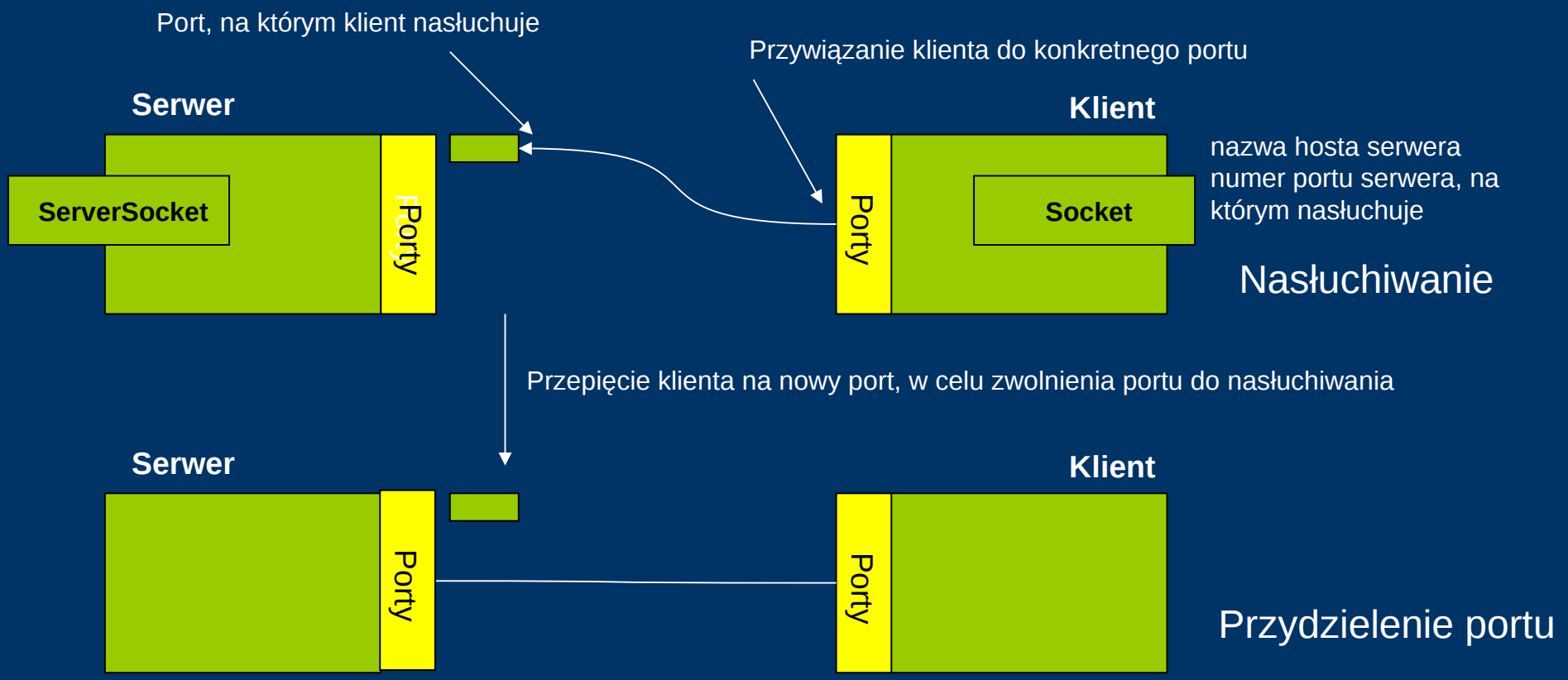
InetAddress

- Klasa reprezentuje 32 bitowy lub 128 bitowy adres IP na protokole transportowym UDP lub TCP
- Złożony jest z adresu IP oraz nazwy hosta
- Dwa typy adresów: multicast i unicast.
- Metody:
  - static getLocalHost() – zwraca obiekt klasy InetAddres opisujący lokalny komputer
  - String getHostAddress() – zwraca adres w postaci XXX.XXX.XXX.XXX
  - String getHostName() – zwraca adres w postaci maszyna.domena1.domena2 …
  - Metody pomocnicze, np. isMulticastAddress()

Model klient-serwer

- W modelu klient-serwer (ang. client-server) dane trzymane są na serwerze, interfejs użytkownika i logika przetwarzania danych są realizowane na kliencie.
- Zadaniem serwera jest przetwarzanie i analizowanie danych przed odesłaniem ich do klienta.
- Przykłady takich modeli: FTP, WWW.
- Przeciwieństwem tego modelu jest model równorzędny (ang. peer-to-peer) takie jak gry sieciowe, system telefoniczny, itp.

Architektura Klient-Serwer

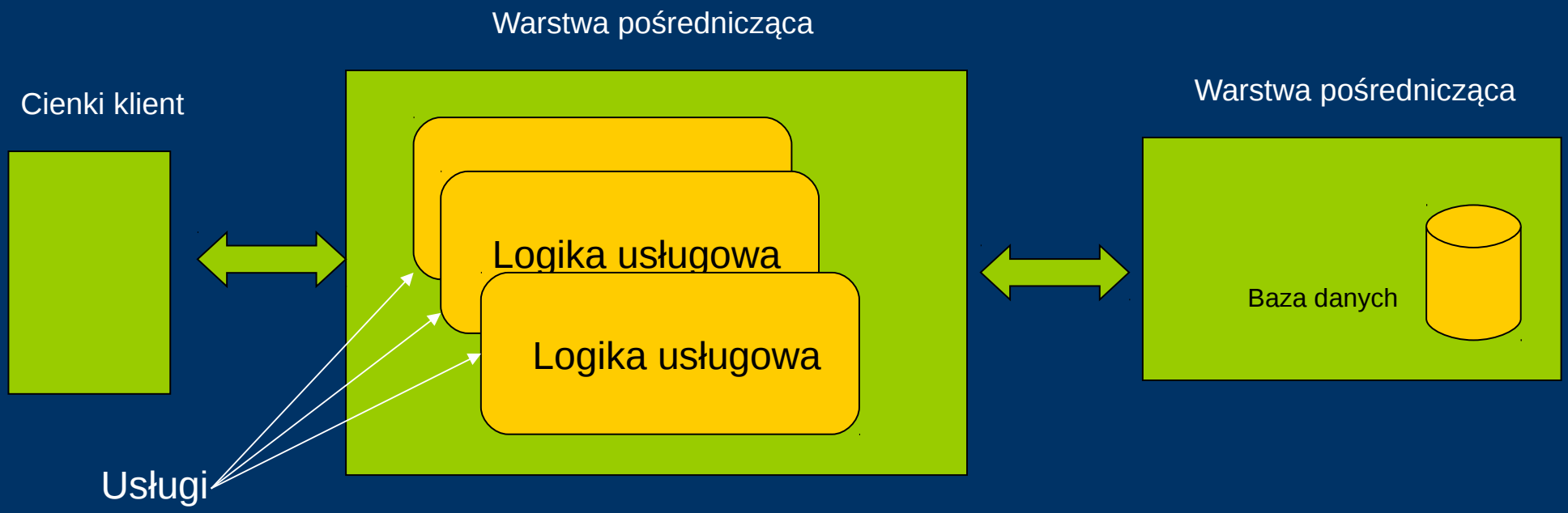


•Gniazdo (socket): jest punktem końcowym wykorzystywanym do zestawiania dwukierunkowego połączenia pomiędzy działającymi aplikacjami w sieci.
•Gniazdo jest przywiązane do wybranego portu, tak że warstwa TCP może identyfikować
•aplikację, do której powinny zostać dostarczone dane.

Architektura wielowarstwowa

Zalety architektury rozproszonej:
- Skalowalność aplikacji
- Dostępność
- Rozszerzalność aplikacji
- Bezpieczeństwo

Wady architektury rozproszonej:
- Dodatkowa złożoność aplikacji
- Potencjalnie wiele miejsc, w których mogą wystąpić błędy
- Ograniczenia związane z przepustowością łączy komunikacyjnych



Czytanie i zapis z gniazda

Klient

1) Utwórz gniazdo. Wymagana nazwa maszyny oraz numer portu.
2) Pobierz strumień wejściowy i wyjściowy PrintWriter i BufferedReader.
3) Czytaj linie po linii z BufferedReader i pisz do PrintWriter
4) Zamknij strumienie.
5) Zamknij połączenie.

Serwer

1) Utwórz gniazdo ServerSocekt na określonym porcie.
2) Zadeklaruj zamienną typu ClientSocket.
3) Użyj metody accept() w celu odebrania i ustanowienia przychodzącego połączenia.
4) Utwórz wątek do obsługi odebranego połączenia.
5) Pobierz strumień do pisania i czytania (PrintWriter i BufferedReader).
6) Pisz i czytaj ze strumieni.
7) Zamknij strumienie.
8) Zamknij połączenia.

Protokół

- Opisuje sposób komunikacji pomiędzy klientem i serwerem. Przeważnie realizowane jako automat.
- W przypadku protokołu tekstowego można wykorzystać tokenizer.

- Tokenizer:
  - utworzyć tokenizer z początkowego ciągu znaków;
  - odbierać kolejne części ciągu znaków (tokenów) za pomocą nextToken();
  - liczba tokenów pozostałych do odebrania: countTokens().

Przykład „Tokenzier”

```
public class TokenizerTest {
   public static void main(String[] args) {
      if (args.length == 1) {
         String input = args[0], delimiters = „;”;
         StringTokenizer token =
            new StringTokenizer(input, delimiters);
         while (token.hasMoreTokens()) {
            System.out.println(token.nextToken());
         }
      }else {
         System.out.println("Podaj ciag znakow„);
      }
   }
}
```

Gniazda

- Transmitowanie danych w pakietach to skomplikowana czynność – na szczęście berkeleyowskie gniazda pozwalają traktować połączenie sieciowe jak strumień.
- Gniazdo obsługuje połączenie między dwoma hostami:
  - łączy się ze zdalną maszyną > wysyła dane > odbiera dane > zamyka połączenie,
  - a także łączy się z portem, czeka na dane i odbiera połączenie o zdalnej maszyny na porcie granicznym.
- Klasa Socket jest wykorzystywana przez klienty i serwery, posiada metody pozwalające wykonywać pierwsze cztery z wymienionych operacji.

Klasa Socket

- Socket to podstawowa klasa do wykonywania operacji TCP po stronie klienta.
- W czasie konstrukcji obiektu klasy Socket od razu jest nawiązywane połączenie:

  Socket (String host, int port) throws UnknownHostException, IOException

  Socket (InetAddress host, int port) throws IOException
- Na końcu pracy z gniazdem należy zamknąć połączenie: socket.close()

http://www.cs.put.poznan.pl/dwawrzyniak/DPE/jsock.pdf

Klasa Socket

- Obiekt klasy Socket posiada kilka metod udostępniających informacje o gnieździe:
  getInetAddress()
  getPort()
  getLocalPort()
  isClosed()

Komunikacja z hostem za pomocą obiektu Socket

- Komunikacja z hostem jest realizowana za pomocą zwykłych strumieni bajtowych, do których dostęp uzyskuje się za pomocą metod:
  getInputStream()
  getOutputStream()
- Strumienie te są opakowywane:

```
Socket s = new Socket(…);
BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
PrintWriter out = new PrintWriter(new
OutputStreamWriter(s.getOutputStream()));
```

Komunikacja z hostem za pomocą obiektu Socket

- Metoda close() gniazda zamyka automatycznie jego strumienie do komunikacji.
- Gdy jeden ze strumieni do komunikacji zostanie zamknięty, całe gniazdo jest zamykane.
- Gdy w czasie pracy z gniazdem chcemy zamknąć tylko strumień do czytania lub pisania należy użyć jednej z metod:

  shutdownInput()
  shutdownOutput()

Opcje gniazda Socket

- Opcje gniazda określają, w jaki sposób gniazda wysyłają i odbierają dane:
  - TCP_NODELAY – wartość true wyłącza schemat buforowania;
  - SO_LINGER – określa, co należy zrobić z datagramami, które nie zostały wysłane przed zamknięciem gniazda;
  - SO_TIMEOUT – wartość wyrażona w milisekundach powoduje, że gniazdo nie zablokuje się na dłużej w trakcie czytania (zgłaszany jest wyjątek InterruptedException ale gniazdo nie jest zamykane);
  - SO_KEEPALIVE – wartość true włącza system kontrolowania bezczynnych połączeń (raz na dwie godziny).

Serwer

- Pisząc serwer zawsze trzeba opracować protokół rozmowy z klientami.
- Cykl życiowy serwera:
  - serwer zajmuje port;
  - w pętli: serwer czeka na klienta, prowadzi z nim rozmowę a na końcu zamyka połączenie z klientem;
  - serwer zwalnia port po zakończonej pracy.

Klasa ServerSocket

- Gniazdo serwera zajmuje lokalny port i czeka na nadchodzące połączenia TCP.
- W czasie konstrukcji obiektu klasy ServerSocket od razu jest zajmowany lokalny port:

ServerSocket (int port) throws IOException
ServerSocket (int port, int queue) throws IOException

Klasa ServerSocket

Przyjmowanie i zamykanie połączeń:

```
ServerSocket server = new ServerSocket(4848);
Socket s = server.accept();
// rozmowa z klientem
s.close();
```

Dostęp do strumieni za pomocą gniazda roboczego:

```
OutputStream os = s.getOutputStream();
InputStream is = s.getInputStream();
```

Klasa ServerSocket

Obiekt klasy ServerSocket posiada kilka metod udostępniających informacje o gnieździe:

- getInetAddress()
- getLocalPort()

Opcje gniazd serwera: SO_TIMEOUT – wartość wyrażona w milisekundach określa czas akceptacji nowego połączenia (wartość 0 oznacza brak limitu).