Java a dostęp do Internetu.

Robert A. Kłopotek
r.klopotek@uksw.edu.pl

Wydział Matematyczno-Przyrodniczy. Szkoła Nauk Ścisłych, UKSW

20.04.2017

Java a dostęp do Internetu

- Java Networking - jest koncepcją połączenia dwóch lub więcej urządzeń komputerowych razem, dzięki czemu możemy dzielić zasoby.
- klasy i interfejsy odpowiadające za obsługę połączeń sieciowych znajdują się w pakiecie `java.net`
- Programowanie gniazd (Java socket) zapewnia możliwość współużytkowania danych między różnymi urządzeniami komputerowymi.
- Gniazdo jest punktem końcowym dwukierunkowej komunikacji pomiędzy odległymi procesami.

Podstawowa terminologia (1/2)

- Adres IP - unikalny numer przypisany do węzła sieci, np. 192.168.0.1. Składa się on z oktetów, które mają zakres od 0 do 255. Jest to logiczny adres, który można zmienić.
- Protokół - jest zbiorem reguł zasadniczo stosowanych w komunikacji. Na przykład: TCP, FTP, Telnet, SMTP, POP itp.
- Port - numer portu służy do jednoznacznego identyfikowania różnych aplikacji. Działa jako punkt komunikacyjny pomiędzy aplikacjami. Numer portu jest skojarzony z adresem IP do komunikacji między dwoma aplikacjami.
- Adres MAC (Media Access Control) - jest unikalnym identyfikatorem NIC (kontroler interfejsu sieciowego - Network Interface Controller). Węzeł sieciowy może mieć wiele NIC, ale każdy z unikalnym MAC.

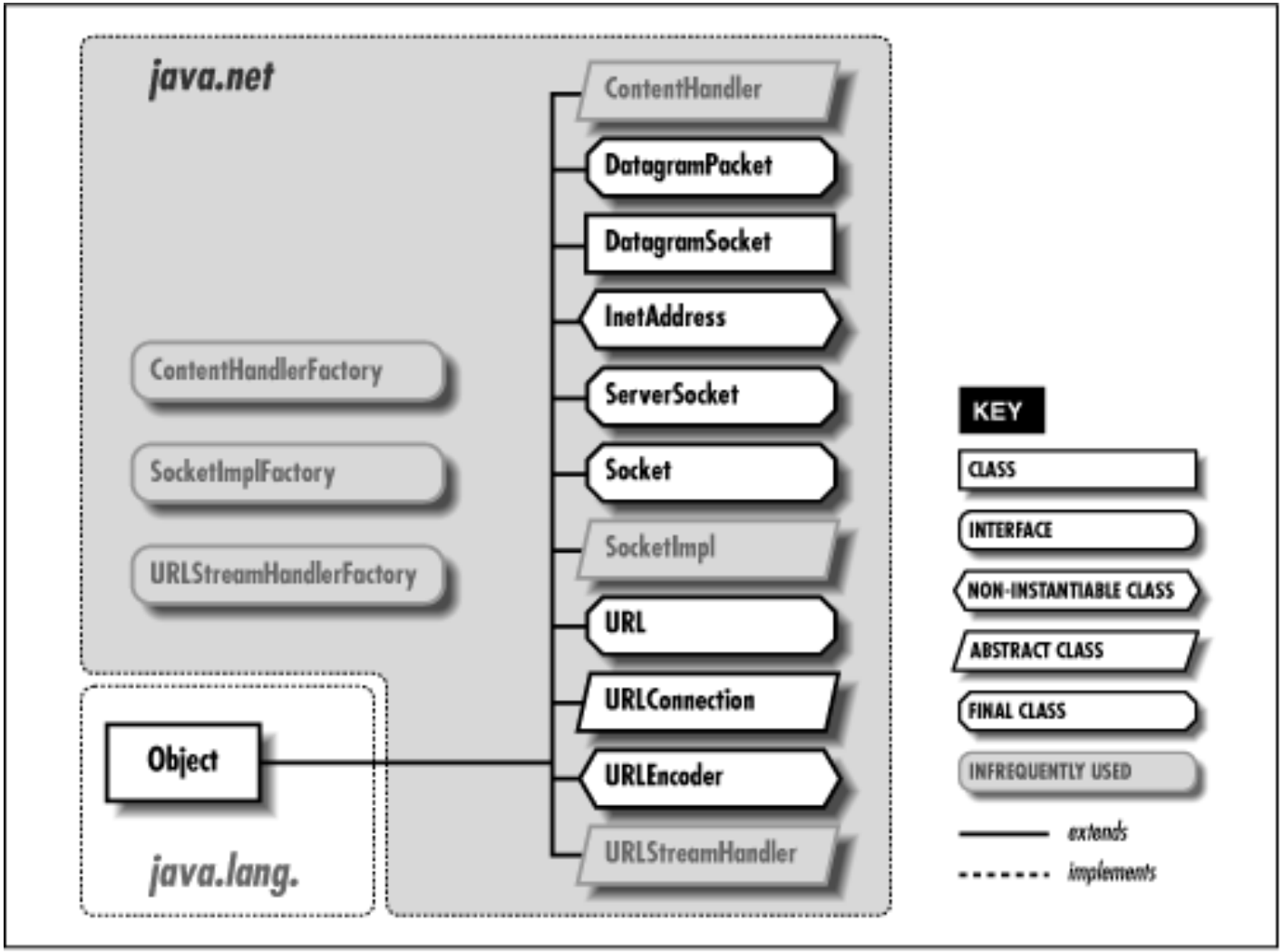
Podstawowa terminologia (2/2)

- Connection-oriented and connection-less protocol
  - W protokole zorientowanym na połączenie potwierdzenie jest wysyłane przez odbiornik. Jest więc niezawodny, ale powolny. Przykładem protokołu zorientowanego na połączenia jest TCP.
  - W protokole bez połączenia, potwierdzenie nie jest wysyłane przez odbiorcę. Nie jest to jednak wiarygodne, ale szybkie. Przykładem protokołu bez połączenia jest UDP
- Gniazdo (socket) - jest punktem końcowym komunikacji dwukierunkowej, wiąże adres IP i port.
- W Javie TCP i UDP obsługują gniazda 64K, od 0 do 65535. Numer gniazda (portu) między 0 a 1023 jest zarezerwowany dla popularnych protokołów aplikacji (np. port 80 dla protokołu HTTP, port 443 dla HTTPS, port 21 dla protokołu FTP, port 23 dla Telnet, port 25 dla SMTP, port 110 dla POP3 itp.)

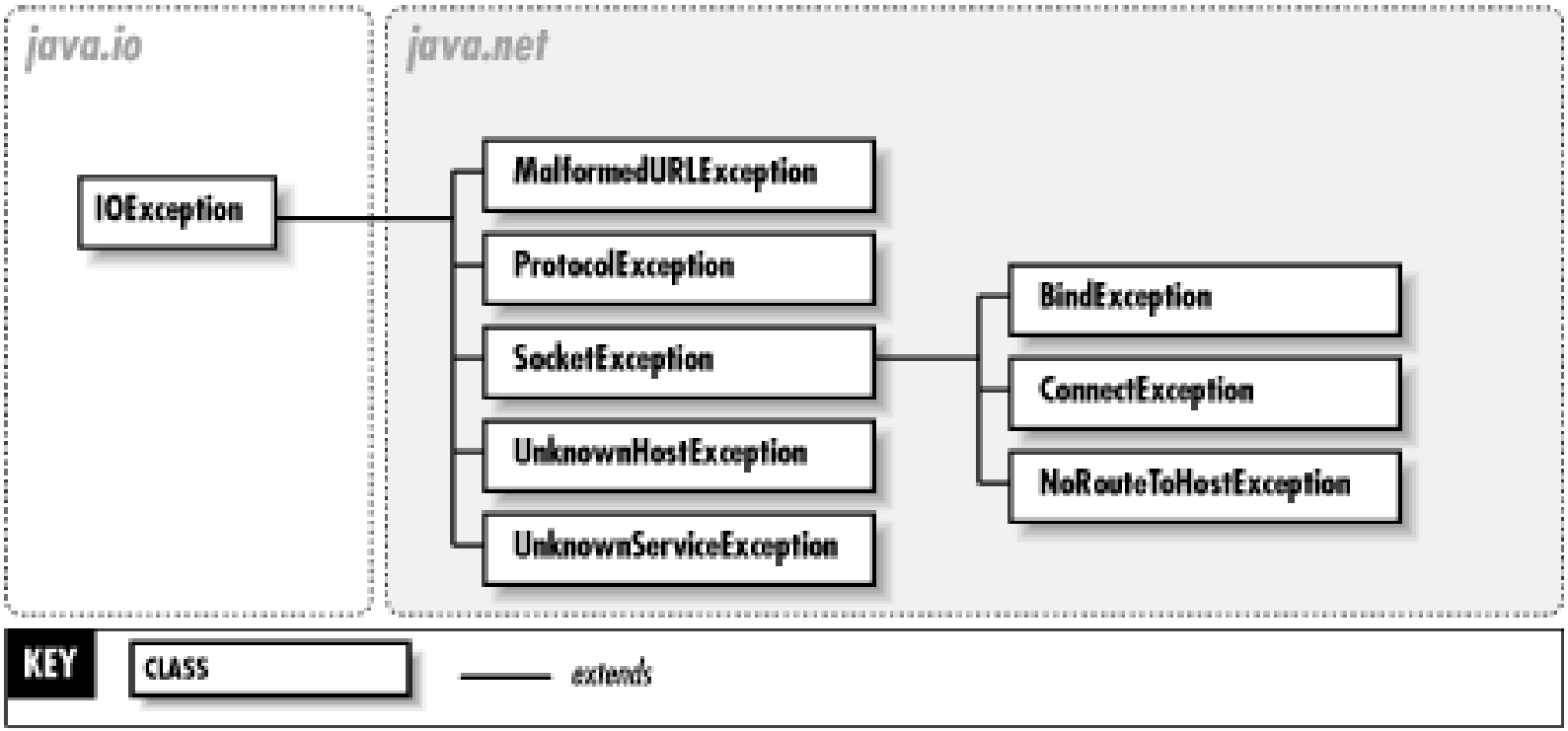
Programowanie gniazd

- Programowanie gniazd Java służy do komunikacji między aplikacjami działającymi w różnych środowiskach JRE.
- Programowanie w języku Java może być zorientowane na połączenia lub bez połączenia.
- Klasy typu Socket i ServerSocket są wykorzystywane w protokole zorientowanym na połączenie, a klasy DatagramSocket i DatagramPacket w protokole bez połączenia.
- Klient w programowaniu gniazd musi znać dwie informacje: adres IP serwera oraz numer portu

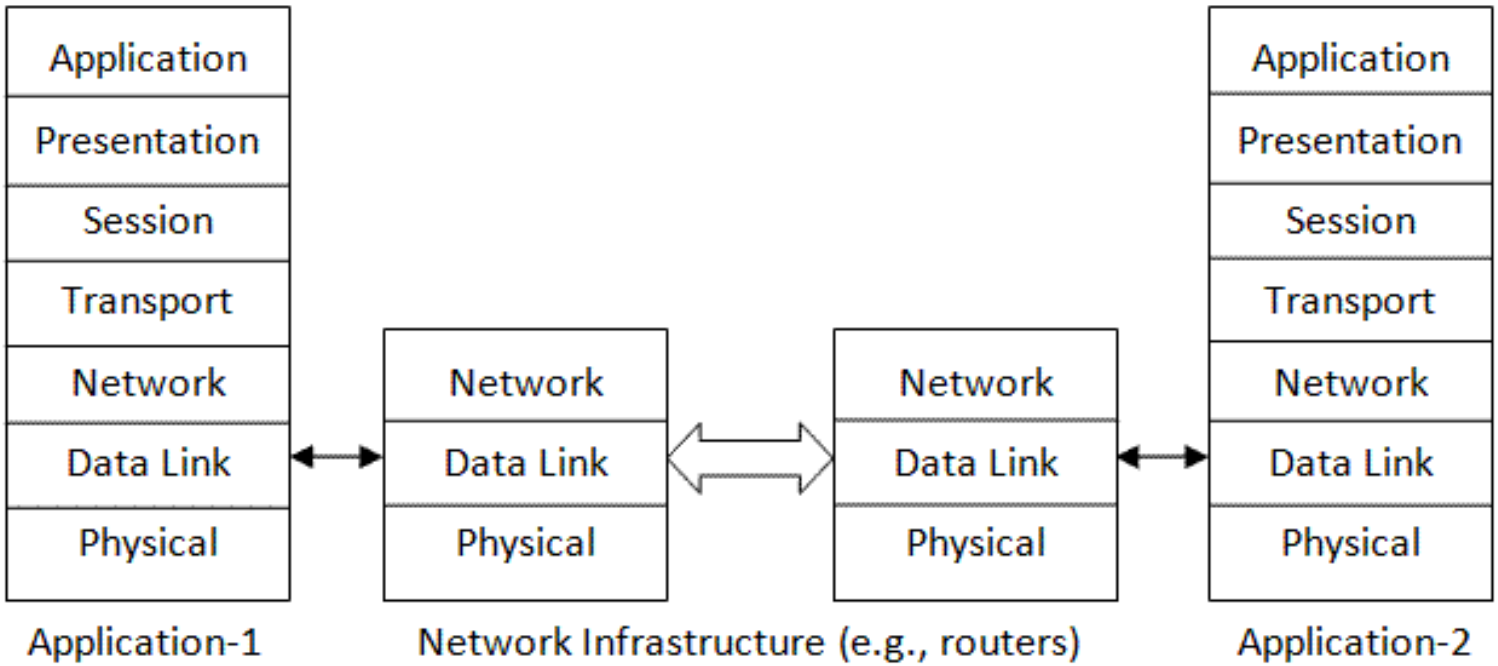
Pakiet java.net



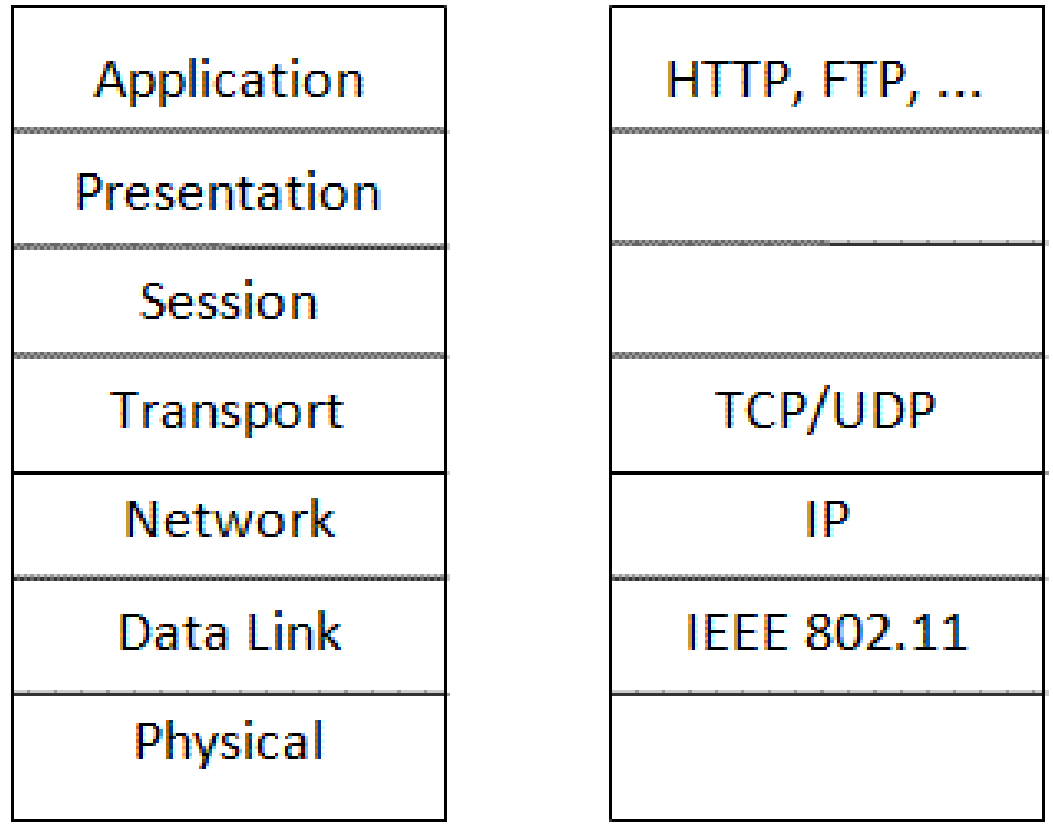
Wyjątki java.net



Architektura OSI



OSI vs TCP/IP



ISO/OSI 7-layer Networking Model

TCP/IP-based Network

Socket i ServerSocket - metody klas

- klasa Socket - jest po prostu punktem końcowym komunikacji między maszynami. Klasa Socket może być użyta do utworzenia gniazda.
  - public InputStream getInputStream()
  - public OutputStream getOutputStream()
  - public synchronized void close()
- klasa ServerSocket - może być użyta do utworzenia gniazda serwera. Obiekt ten służy do nawiązywania komunikacji z klientami.
  - ppublic Socket accept()
  - public synchronized void close()

Klient i serwer - przykład

```
//MyServer.java
import java.io.*;
import java.net.*;
public class MyServer {
 public static void main(
                    String[] args){
  try{
   ServerSocket ss =
            new ServerSocket(1234);
   Socket s = ss.accept();
   DataInputStream dis=
    new DataInputStream(
                s.getInputStream());
   String str =
             (String)dis.readUTF();
   System.out.println(
                  "message= "+str);
   ss.close();
  }catch(Exception e)
          {System.out.println(e);} }
 }
}
```

```
//MyClient.java
import java.io.*;
import java.net.*;
public class MyClient {
 public static void main(
                          String[] args) {
  try{
   Socket s =
     new Socket("localhost",1234);
   DataOutputStream dout =
    new DataOutputStream(
                s.getOutputStream());
   dout.writeUTF("Hello Server");
   dout.flush();
   dout.close();
   s.close();
  }catch(Exception e)
          {System.out.println(e);}
}
```

Serwer wielowątkowy - przykład

```
//MultiServer.java
import java.io.*;
import java.net.*;
public class MultiServer {
 public static void main(
                    String[] args){
  int number = 0;
  try{
   ServerSocket server_socket =
           new ServerSocket(6000);
   while(true) {
    Socket socket =
           server_socket.accept();
    System.out.println(
            "Zglosil sie klient");
    number++;
    new ServiceThread(
           socket, number).start();
   }
  }catch(Exception e) {
           {System.out.println(e);}
  }
 }
}
```

```
//ServiceThread.java
import java.io.*;
import java.net.*;
public class ServiceThread
   extends java.lang.Thread {
 private Socket socket;
 private int number;

 public ServiceThread(Socket s,
                      int n){
  socket = s;
  number = n;
 }
 public void run() {
  ...
 }
}
```

DatagramSocket i DatagramPacket

- klasa DatagramSocket - Klasa Java DatagramSocket reprezentuje gniazdo bez połączenia, służące do wysyłania i odbierania pakietów datagramowych. Datagram jest zasadniczo informacją, ale nie ma gwarancji jej zawartości, dostarczenia ani czasu dotarcia.
  - DatagramSocket() throws SocketEeption
  - DatagramSocket(int port) throws SocketEeption
  - DatagramSocket(int port, InetAddress address) throws SocketEeption:
- klasa DatagramPacket - to wiadomość, która może zostać wysłana lub odebrana. Jeśli wysyłasz wiele pakietów, mogą one docierać w dowolnej kolejności. Dodatkowo dostarczenie pakietów nie jest gwarantowane.
  - DatagramPacket(byte[] barr, int length)
  - DatagramPacket(byte[] barr, int length, InetAddress address, int port)

Wysyłanie i odbieranie pakietów - przykład

```
//DSender.java
import java.net.*;
public class DSender{
 public static void main(
                  String[] args)
              throws Exception {
  DatagramSocket ds =
           ew DatagramSocket();
  String str = "Welcome java";
  InetAddress ip =
     netAddress.getByName(
                  "127.0.0.1");

  DatagramPacket dp =
    new DatagramPacket(
      str.getBytes(),
      str.length(), ip, 3000);
  ds.send(dp);
  ds.close();
 }
}
```

```
//DReceiver.java
import java.net.*;
public class DReceiver{
 public static void main(
                    String[] args)
                throws Exception {
  DatagramSocket ds =
        new DatagramSocket(3000);
  byte[] buf = new byte[1024];
  DatagramPacket dp =
   new DatagramPacket(buf, 1024);
  ds.receive(dp);
  String str =
          new String(dp.getData(),
                0, dp.getLength());
  System.out.println(str);
  ds.close();
 }
}
```

Java URL

- Klasa URL reprezentuje adres URL. URL jest skrótem dla Uniform Resource Locator. Wskazuje zasoby w sieci World Wide Web.
- URL może zawierać takie informacje jak: protokół, nazwa serwera lub adres IP, numer portu, plik lub katalog
- Klasa URLConnection reprezentuje łącze komunikacyjne między adresem URL a aplikacją. Ta klasa może być używana do odczytu i zapisu danych do określonego zasobu, do którego odwołuje się URL.
- Klasa HttpURLConnection to specyficzny adres http dla URLConnection. Działa tylko dla protokołu HTTP. Dzięki klasie HttpURLConnection można uzyskać informacje o dowolnym adresie HTTP, takim jak informacje o nagłówku, kod statusu, kod odpowiedzi itp.
- java.net.HttpURLConnection jest podklasą klasy URLConnection.

URL - przykład

```
//URLDemo.java
import java.io.*;
import java.net.*;
public class URLDemo{
 public static void main(String[] args){
  try{
   URL url=new URL("http://rklopotek.blog.uksw.edu.pl/java");

   System.out.println("Protocol: "+url.getProtocol());
   System.out.println("Host Name: "+url.getHost());
   System.out.println("Port Number: "+url.getPort());
   System.out.println("File Name: "+url.getFile());

   }catch(Exception e){System.out.println(e);
  }
 }
}
```

URL - wydruk

```
Protocol: http
Host Name: rklopotek.blog.uksw.edu.pl
Port Number: -1
File Name: /java
```

URLConnecton - przykład

```
import java.io.*;
import java.net.*;
public class URLConnectionExample {
 public static void main(String[] args){
  try{
   URL url = new URL("http://rklopotek.blog.uksw.edu.pl/java");
   URLConnection urlcon = url.openConnection();
   InputStream stream = urlcon.getInputStream();
   int i;
   while((i=stream.read())!=-1){
    System.out.print((char)i);
   }
  }catch(Exception e){System.out.println(e);}
 }
}
```

HttpURLConnection - przykład

```
import java.io.*;
import java.net.*;
import java.util.Map;
public class HttpURLConnectionDemo{
 public static void main(String[] args){
  try{
   URL url=new URL("http://rklopotek.blog.uksw.edu.pl/java");
   HttpURLConnection huc=(HttpURLConnection)url.openConnection();
   for(int i=1;i<=9;i++){
    System.out.println(huc.getHeaderFieldKey(i)
     +"␣=␣"+huc.getHeaderField(i));
   }
   huc.disconnect();
  }catch(Exception e){System.out.println(e);}
 }
}
```

HttpURLConnection - wydruk

```
Date = Wed, 03 May 2017 22:18:57 GMT
Server = Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips
         PHP/5.6.30 mod_wsgi/3.4 Python/2.7.5
X-Powered-By = PHP/5.6.30
Link = <http://rklopotek.blog.uksw.edu.pl/wp-json/>;
       rel="https://api.w.org/"
Link = <http://rklopotek.blog.uksw.edu.pl/?p=6>;
       rel=shortlink
Keep-Alive = timeout=5, max=100
Connection = Keep-Alive
Transfer-Encoding = chunked
Content-Type = text/html; charset=UTF-8
```

Java URL

- Klasa InetAddress języka Java reprezentuje adres IP.
- Klasa java.net.InetAddress udostępnia metody uzyskiwania adresu IP dowolnej nazwy hosta, na przykład www.google.com, www.facebook.com itd.
- Metody:
  - public static InetAddress getByName(String host) throws UnknownHostException
  - public static InetAddress getLocalHost() throws UnknownHostException
  - public String getHostName()
  - public String getHostAddress()

Pytania?