

Zaawansowane Programowanie Obiektowe



Wykład 7a Programowanie sieciowe w Javie



dr inż. Wojciech Bieniecki

wbieniec@kis.p.lodz.pl

<http://wbieniec.kis.p.lodz.pl>

Adresowanie komputerów w sieci

`InetAddress` - klasa opisująca adres komputera w sieci poprzez nazwę/domenę, oraz poprzez numer IP.

Istnieje szereg metod statycznych klasy `InetAddress` wykorzystywanych do tworzenia obiektu klasy, brak jest bowiem konstruktorów.

Podstawowe metody wykorzystywane do tworzenia obiektów to:

`InetAddress.getByName(String name)` throws `UnknownHostException`;
tworzy obiekt klasy bazując na podanej nazwie komputera lub adresie

`InetAddress.getAllByName(String name)` throws `UnknownHostException`;
wykorzystywana wówczas kiedy komputer o danej nazwie ma wiele adresów IP. Zwracana jest wówczas tablica obiektów

`InetAddress.getLocalhost()` throws `UnknownHostException`;
wykorzystywana do uzyskania obiektu reprezentującego adres komputera lokalnego

Adresowanie komputerów w sieci

Adresowanie localhosta

```
import java.net.*;
. . .
try{
    InetAddress a0 = InetAddress.getLocalHost();
    System.out.println("Host address"+a0.getHostName()+" is: " +a0);
}
catch (UnknownHostException he){he.printStackTrace();}
```

Host odległy

```
InetAddress address = InetAddress.getByName("www.oreilly.com");
InetAddress address2 = InetAddress.getByName("212.191.89.2");
```

Adresowanie komputerów w sieci

Adresowanie wielu hostów

```
import java.net.*;
public class AllAddressesOfMicrosoft
{
    public static void main (String[] args) {
        try {
            InetAddress[] addresses =
                InetAddress.getAllByName("www.microsoft.com");
            for(int i = 0; i < addresses.length; i++) {
                System.out.println(addresses[i]);
            }
        }
        catch (UnknownHostException e) {
            System.out.println("Could not find www.microsoft.com");
        }
    }
}
```

*Uwaga – nie uwierzytelniony aplet nie może wykonywać operacji
getByName i getAllByName*

Metody klasy InetAddress

getHostName

```
try {  
    InetAddress ia = InetAddress.getByName("152.2.22.3");  
    System.out.println(ia.getHostName());  
}  
catch (Exception e) {  
    System.
```

getHostAddress

```
try {  
    InetAddress me = InetAddress.getLocalHost();  
    String dottedQuad = me.getHostAddress();  
}
```

Metody klasy InetAddress

getAddress

```
import java.net.*;
public class AddressTests {
    public static int getVersion(InetAddress ia) {
        byte[] address = ia.getAddress( );
        if (address.length == 4) return 4;
        else if (address.length == 16) return 6;
        else return -1;
    }
    public static char getClass(InetAddress ia) {
        byte[] address = ia.getAddress( );
        if (address.length != 4) {
            throw new IllegalArgumentException("No IPv6 addresses!");
        }
        int firstByte = address[0];
        if ((firstByte & 0x80) == 0) return 'A';
        else if ((firstByte & 0xC0) == 0x80) return 'B';
        else if ((firstByte & 0xE0) == 0xC0) return 'C';
        else if ((firstByte & 0xF0) == 0xE0) return 'D';
        else if ((firstByte & 0xF8) == 0xF0) return 'E';
        else return 'F';
    }
}
```

Zadanie: napisz program nslookup

Klasa URL

Klasa URL zawiera szereg metod umożliwiających filtrację adresu

Przykładowe konstruktory mają postać:

```
URL(String addressee) ,  
URL(String protocol, String host, int port, String file)
```

```
getProtocol()
```

Wyłuskuje nazwę protokołu

```
getHost()
```

Wyłuskuje nazwę komputera

```
getFile()
```

Wyłuskuje ścieżkę pliku

```
getPort()
```

Wyłuskuje numer portu

```
openStream()
```

Otwarcie strumienia pliku

Klasa URL

Testowanie protokołów sieciowych

```
private static void testProtocol(String url) {
    try {
        URL u = new URL(url);
        System.out.println(u.getProtocol( ) + " is supported");
    }
    catch (MalformedURLException e) {
        String protocol = url.substring(0, url.indexOf(':'));
        System.out.println(protocol + " is not supported");
    }
}
```

```
testProtocol("http://www.adc.org");
testProtocol("https://www.amazon.com/exec/obidos/order2/");
testProtocol("ftp://metalab.unc.edu/pub/languages/java/");
testProtocol("mailto:elharo@metalab.unc.edu");
testProtocol("telnet://dibner.poly.edu/");
testProtocol("file:///etc/passwd");
testProtocol(
"ldap://ldap.itd.umich.edu/o=UM,c=US?postalAddress");
```


Klasa URL

Tworzenie obiektu URL

```
URL u = new URL(  
    "http",  
    "www.eff.org",  
    "/blueribbon.html#intro");
```

```
URL u = new URL(  
    "http",  
    "lcsaxp.lcs.psu.edu",  
    1212,  
    "/%3b&db=psu");
```

Adresowanie względne

```
URL u1 = new URL("http://metalab.unc.edu/javafaq/index.html");  
URL u2 = new URL (u1, "mailinglists.html");  
URL relative = new URL(this.getDocumentBase( ),  
    "mailinglists.html");
```

Aplet testujący protokoły

```
import java.net.*;
import java.applet.*;
import java.awt.*;
public class ProtocolTesterApplet extends Applet {
    TextArea results = new TextArea( );
    public void init( ) {
        this.setLayout(new BorderLayout( ));
        this.add("Center", results);
    }
    public void start( ) {
        String host = "www.kis.p.lodz.pl";
        String file = "/index.html";
        String[] schemes = {"http", "https", "ftp", "mailto", "telnet"};
        for (int i = 0; i < schemes.length; i++) {
            try {
                URL u = new URL(schemes[i], host, file);
                results.append(schemes[i] + " is supported\r\n");
            }
            catch (MalformedURLException e) {results.append(schemes[i] + "
is not supported\r\n");}
        }
    }
}
```

Przetwarzanie łańcucha URL

Składniki URL

- Protokół (Protocol)
- Adres serwera (Server address)
- Ścieżka (pathname)
- Sekcja pliku (File section)
- Zapytanie (Query)

```
http://metalab.unc.edu/javafaq/books/jnp/index.html?isbn=1565922069#toc
```

```
http://admin@www.blackstar.com:8080/
```

Przetwarzanie łańcucha URL

public String getProtocol()

```
URL page = this.getCodeBase( );  
System.out.println(" aplet accessed by "  
+ page.getProtocol( ));
```

public String getHost()

```
URL u = new  
URL("ftp://anonymous:anonymous@wuarchive.wustl.edu/");  
String host = u.getHost( );
```

Co zwróci funkcja?

public int getPort()

```
URL u = new URL("http://www.kis.p.lodz.pl/");  
System.out.println("The port port of " + u + " is " +  
u.getPort( ));
```

Co zwróci funkcja?

Przetwarzanie łańcucha URL

public String getFile() i public String getPath()

http://metalab.unc.edu/javafaq/books/jnp/index.html?isbn=1565922069#toc

http://www.kis.p.lodz.pl/

Co zwróćą funkcje?

public String getRef()

public String getQuery()

public String getUserInfo()

public String getAuthority()

Strumienie - przypomnienie

Klasyfikacja strumieni

	Wejściowe	Wyjściowe
Strumienie bajtowe	InputStream	OutputStream
Strumienie znakowe	Reader	Writer

Tabela: Klasy (interfejsy) bazowe dla strumieni w Javie

Metody podstawowe dla strumieni wejściowych

read – czyta bajt / znak, lub tablicę bajtów / znaków

reset – cofa wskaźnik pliku do początku

skip – przesuwa wskaźnik pliku o określoną liczbę bajtów/znaków

close – zamyka strumień

Metody podstawowe dla strumieni wyjściowych

write – zapisuje bajt / znak, lub tablicę bajtów / znaków / String

flush – wymusza zapis bufora do strumienia

append – (tylko dla znakowych) – zapisuje znak do strumienia

close – zamyka strumień

Łączenie strumienia ze obiektem źródłowym lub docelowym

Nadajnik / odbiornik	Strumienie znakowe	Strumienie bajtowe
Pamięć	CharArrayReader CharArrayWriter	ByteArrayInputStream ByteArrayOutputStream
	StringReader StringWriter	(nie używane byte-to-character conversion)
Łączy (mechanizmy IPC)	PipedReader PipedWriter	PipedInputStream PipedOutputStream
Plik	FileReader FileWriter	FileInputStream FileOutputStream

Table: Subject classes (associated with a particular source / receiver)

Przykład – kopiowanie bajt po bajcie

```
import java.io.*;

class CopyFile
{
    static public void main(String[] args)
    {
        FileInputStream inp;
        FileOutputStream outp;
        try {
            inp = new FileInputStream(args[0]);
            outp = new FileOutputStream(args[1]);
            int c;
            while ((c=inp.read()) != -1) outp.write(c); // actual copying
        } catch(ArrayIndexOutOfBoundsException e) { // no arguments
            System.out.println("Syntax: CopyFile input output");
            System.exit(1); // error code
        } catch(FileNotFoundException e) { // unknown file
            System.out.println("File " + args[0] + " or file " +
                args[1] + " doesn't exist.\n");
            System.exit(2); // another error
        } catch(IOException e) { // another error, ie. disk full
            System.out.println(e.toString());
            System.exit(3);
        }
    }
}
```

Uwaga na obsługę
wyjątków

Łączenie funkcjonalności strumieni

FileInputStream oraz **FileOutputStream** mogą czytać lub zapisywać tylko **bajty**.

Strumienie filtrujące pozwalają obsługiwać inne formaty proste takie jak int, double

```
FileInputStream fin = new FileInputStream("numbers.dat");  
DataInputStream din = new DataInputStream(fin);  
double x = din.readDouble();
```

Strumienie filtrujące oraz transformujące są przykładem dekoratorów

Klasy przetwarzania

Klasy przetwarzające są niezależne od źródła i celu strumienia

Sposób przetwarzania	Strumień znakowe	Strumień bajtowe
Buforowanie	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
Filtrowanie – klasy bazowe, które definiują metody dla klas finalnych	FilterReader FilterWriter	FilterInputStream FilterOutputStream
Filtrowanie – klasy finalne		
Konwersja bajt -> znak	InputStreamReader OutputStreamWriter	
Konkatenacja		SequenceInputStream
Serializacja obiektów		ObjectInputStream ObjectOutputStream
Konwersja danych		DataInputStream DataOutputStream
Liczenie linii	LineNumberReader	LineNumberInputStream
Przeglądanie	PushbackReader	PushbackInputStream
Drukowanie	PrintWriter	PrintStream

Text I/O

Używamy klas `FileReader`, `FileWriter`.

FileReader: konwersja z domyślnej strony kodowej na Unicode.

FileWriter: konwersja w odwrotną stronę

```
import java.io.*;
public class ReadByReader {
    public static void main(String[] args) {
        StringBuffer cont = new StringBuffer();
        try {
            FileReader inp = new FileReader(args[0]);
            int c;
            while ((c=inp.read()) != -1)
                cont.append((char)c);
            inp.close();
        } catch (Exception e) { System.exit(1); }
        String s = cont.toString();
        System.out.println(s);
    }
}
```

Niestandardowe kodowanie znaków

W tym przypadku nie da się wykorzystać **FileReader**.

Używamy klas czytających / piszących bajtowo: `InputStreamReader`, `OutputStreamWriter`



```
FileInputStream fis = new FileInputStream("test.txt");  
InputStreamReader isr = new InputStreamReader(fis, "UTF8");
```

Odczyt pliku znakowego (tekstowego)

Metoda `read()`, z klasy `InputStreamReader` (czyta `FileReader`), czyta znak/znaki

Większe możliwości daje klasa `Ljava.util.Scanner`

<code>String</code>	<code>next()</code> Finds and returns the next complete token from this scanner.
<code>String</code>	<code>next(Pattern pattern)</code> Returns the next token if it matches the specified pattern.
<code>String</code>	<code>next(String pattern)</code> Returns the next token if it matches the pattern constructed from the specified string.
<code>BigDecimal</code>	<code>nextBigDecimal()</code> Scans the next token of the input as a <code>BigDecimal</code> .
<code>BigInteger</code>	<code>nextBigInteger()</code> Scans the next token of the input as a <code>BigInteger</code> .
<code>BigInteger</code>	<code>nextBigInteger(int radix)</code> Scans the next token of the input as a <code>BigInteger</code> .
<code>boolean</code>	<code>nextBoolean()</code> Scans the next token of the input into a boolean value and returns that value.
<code>byte</code>	<code>nextByte()</code> Scans the next token of the input as a byte.
<code>byte</code>	<code>nextByte(int radix)</code> Scans the next token of the input as a byte.
<code>double</code>	<code>nextDouble()</code> Scans the next token of the input as a double.

PrintWriter jest właściwą klasą do zapisu plików tekstowych

<code>void</code>	<code>print</code> (boolean b) Prints a boolean value.
<code>void</code>	<code>print</code> (char c) Prints a character.
<code>void</code>	<code>print</code> (char[] s) Prints an array of characters.
<code>void</code>	<code>print</code> (double d) Prints a double-precision floating-point number.
<code>void</code>	<code>print</code> (float f) Prints a floating-point number.
<code>void</code>	<code>print</code> (int i) Prints an integer.
<code>void</code>	<code>print</code> (long l) Prints a long integer.
<code>void</code>	<code>print</code> (Object obj) Prints an object.
<code>void</code>	<code>print</code> (String s) Prints a string.
<code>PrintWriter</code>	<code>printf</code> (Locale l, String format, Object ... args) A convenience method to write a formatted string to this writer using

Buforowanie

*Buforowanie ogranicza liczbę odwołań do urządzeń fizycznych.
Czytając duży plik należy unikać korzystania wyłącznie z klasy `FileReader`*

Użycie `BufferedReader` powinno przyspieszyć efektywność działania programu. Należy jej użyć jako klasy dekorującej `FileReader` połączonego do fizycznego urządzenia

```
FileReader fr = new FileReader("plik.txt");  
BufferedReader br = new BufferedReader(fr);  
String line;  
while ((line = br. readLine()) != null){  
    // processing a line  
}
```


Buforowanie szukania słów w pliku

```
class Search{
    public boolean hasAnyWord(String fname, Hashtable wordtab)
    {
        boolean result = false;
        try{
            String line;
            FileReader fr = new FileReader(fname);
            BufferedReader br = new BufferedReader(fr);

search:
            while ((line = br.readLine()) != null ){
                StringTokenizer st = new StringTokenizer(line, " ,.:;()\\t\\r\\n");
                while (st.hasMoreTokens()){
                    String word = st.nextToken();
                    if (wordtab.get(word) != null){
                        result = true;
                        break search;
                    }
                }
            }
            br.close();
        }
        catch (IOException e){ System.err.println(e); }
        return result;
    }
}
```

Buforowanie szukania słów w pliku

```
public class Szukaj{
    public static void main(String[] args){
        /* argumenty: nazwa_pliku slowo1 slowo2 ... */
        if (args.length < 2)
            System.exit(1);

        Object dummy = new Object();
        Hashtable words = new Hashtable();
        for (int i = 1; i<args.length; i++)
            words.put(args[i], dummy);

        Search srch = new Search();
        boolean result = srch.hasAnyWord(args[0], words);
        String msg = " nie zawiera zadnego ze slow:";
        if (result) msg = " zawiera ktoreś ze slow:";
        System.out.println("Plik "+args[0]+msg);

        Enumeration en = words.keys();// uzyskujemy wszystkie klucze
        tablicy
        while (en.hasMoreElements()){
            // ... i przebiegamy je po kolei
            String word = (String) en.nextElement();
            System.out.println(word);
        }
    }
}
```

Kodowanie pliku

Java używa 16 bitowego Unicodu. Jeżeli nie ustawimy sposobu konwersji, wykorzystana zostanie konwersja domyślna.

```
public class DefaultEncoding
{
    public static void main(String args[])
    {
        String p = System.getProperty("file.encoding") ;
        System.out.println(p) ;
    }
}
```

Przykładowe wyniki: ISO8859_1, ibm-852, Cp852 (Latin 2) , Cp1252 (Windows Western Europe/Latin-1).

Strumienie znakowe pozwalają na przezroczystą dla programisty konwersję źródłowego strumienia bajtowego na znaki Unicode i na odwrót. Są to wspomniane już klasy `InputStreamReader` oraz `OutputStreamWriter`

Przykład – konwersja HTML

```
import java.io.*;
public class Convert
{
    public static void main(String[] args)
    {
        String infile = args[0], // plik we
        in_enc = args[1],         // strona kodowa wejścia
        outfile = args[2],        // plik wynikowy
        out_enc = args[3];        // strona kodowa wyjścia
        try{
            FileInputStream fis = new FileInputStream(infile);
            InputStreamReader in = new InputStreamReader(fis, in_enc);
            FileOutputStream fos = new FileOutputStream(outfile);
            OutputStreamWriter out = new OutputStreamWriter(fos, out_enc);
            int c;
            while ((c = in.read()) != -1) out.write(c);
            in.close();
            out.close();
        }
        catch (IOException e){ System.err.println(e);}
    }
}
```

Przetwarzanie strumienia wejściowego z sieci

```
public final InputStream openStream( ) throws IOException
```

Metoda dokonuje połączenia z adresem URL, wykonuje handshaking i zwraca otwarty strumień bajtów. Strumień ten jest surowy. Nie zawiera nagłówka HTTP.

```
URLConnection openConnection( ) throws IOException
```

Metoda otwiera socket do zasobu i zwraca obiekt URLConnection, który reprezentuje połączenie do zasobu. Pozwala na bezpośredni dostęp do danych surowych a także nagłówków HTTP, umożliwia odczyt i zapis.

```
public final Object getContent( ) throws IOException
```

Przykład na getContent

```
try
{
    URL u = new URL(args[0]);
    Object o = u.getContent( );
    System.out.println("I got a " + o.getClass().getName( ));
} // end try
catch (IOException e) {
    System.err.println(e);
}
catch (MalformedURLException e)
{
    System.err.println(args[0] + " is not a parseable URL");
}
```

*JVM próbuje przekształcić pobrany strumień na obiekt.
Wynik takiej zamiany może być nieprzewidywalny.*

Przkład na getContent

```
URL u = new URL("http://www.kis.p.lodz.pl");
Class[] types = new Class[3];
types[0] = String.class;
types[1] = Reader.class;
types[2] = InputStream.class;
Object o = u.getContent(types);
if (o instanceof String) {System.out.println(o);}
else if (o instanceof Reader){
    int c;
    Reader r = (Reader) o;
    while ((c = r.read( )) != -1) System.out.print((char) c);
}
else if (o instanceof InputStream) {
    int c;
    InputStream in = (InputStream) o;
    while ((c = in.read( )) != -1) System.out.write(c);
}
else {
    System.out.println("Error: unexpected type " + o.getClass( ));
}
```

Gniazda

...umożliwiają niskopoziomową komunikację sieciową. Gniazdo to jeden z końców dwukierunkowego połączenia pomiędzy klientem a serwerem.

Serwer (ang. server) to program, który otwiera gniazdo (zwane zazwyczaj standardowym portem (ang. well-known port) i czeka, aż podłączy się klient (ang. client).

Klient łączy się z jakiegoś niezajętego portu (nazywanego *portem tymczasowym* (ang. *ephemeral port*)).

Kiedy tylko klient i serwer się połączą, serwer zazwyczaj proponuje, żeby przenieść rozmowę na inny port. Dzięki temu standardowy port się zwalnia i może czekać na kolejne połączenie.

Porty

Service name	TCP port	UDP port	RFC document
echo	7	7	862
discard	9	9	863
daytime	13	13	867
chargen	19	19	864
time	37	37	868

Tabela. Najczęściej spotykane usługi internetowe i odpowiadające im numery portów

Otwieranie gniazda

*Przykład zastosowania: Połączyć się z serwerem echo i używać tego serwisu, na przykład otrzymywać wysyłany tekst (do momentu dopóki nie wyślemy serwerowi tekstu „Bye”).
Aby stworzyć gniazdo, należy zainicjować klasę Socket z pakietu java.net:*

1. stworzyć gniazdo(ustanowić połączenie)

```
s = new Socket("zly.kis.p.lodz.pl", 7);
```

2. odczytać tekst użytkownika np. poprzez utworzenie strumienia BufferedReader (w którym zawarta jest metoda readLine()), utworzonej na podstawie strumienia System.in (odczyt z klawiatury)

```
BufferedReader kr =  
new BufferedReader(new InputStreamReader(System.in));
```

3. zbudować strumień wysyłający tekst do serwera i odczytujący z niego

```
PrintWriter pw=new PrintWriter(s.getOutputStream(),true);  
BufferedReader br=  
new BufferedReader(new InputStreamReader(s.getInputStream()));
```

Otwieranie gniazda

4. zaimplementować procedury komunikacyjne (otrzymują wysłany tekst do momentu w którym wyślemy "BYE"):

```
while( !line.equals("bye") )  
{  
    line=kr.readLine();  
    pw.println(line);  
    line= br.readLine();  
    System.out.println("Line received: " + line);  
}
```

5. zamknąć strumienie

```
br.close();  
s.close();
```

Gniazdo serwerowe

1. Tworzymy obiekt gniazda serwerowego

```
try
{
    fServerSocket = new ServerSocket(kPortNumber);
    System.out.println("TServer started...");
}
catch (IOException e)
{ ... }
```

2. Czekamy na połączenie klienta

```
Socket theClientSocket;
while (true)
{
    if (fServerSocket == null)
        return;
    try
    {
        theClientSocket = fServerSocket.accept();
```

Gniazdo serwerowe

3. zaakceptuj połączenie klienta i wyślij komunikat do klienta

```
PrintWriter theWriter = new PrintWriter(  
    new OutputStreamWriter(theClientSocket.getOutputStream())) ;  
theWriter.println(new java.util.Date().toString()) ;  
theWriter.flush();
```

4. przerwij połączenie

```
    theWriter.close() ;  
    theClientSocket.close() ;  
}  
catch (IOException e)  
{  
    System.err.println("Ugly exception: " + e.getMessage ());  
    System.exit(1) ;  
}  
}
```

Gniazda bezpołączeniowe

Gniazda bezpołączeniowe (ang. datagram sockets) umożliwiają przesyłanie i odbieranie datagramów. Najczęściej są to pakiety UDP.

Przy tym połączeniu nie rozróżniamy klienta i serwera, ponieważ połączenie nie jest zestawiane

Pakiety są wysyłane „w świat” i nie ma gwarancji że w ogóle dotrą do odbiorcy i będą ułożone w odpowiedniej kolejności

Konstruktory

Pozwalają na utworzenie połączenia do gniazda i przydzielenie go maszyny lokalnej lub odległej, dla portu domyślnego lub określonego

```
DatagramSocket()
```

```
DatagramSocket(int port)
```

```
DatagramSocket(int port, InetAddress laddr)
```

```
DatagramSocket(SocketAddress bindaddr)
```

Gniazda bezpołączeniowe

Przetwarzanie obiektów gniazda bezpołączeniowego

void bind(SocketAddress addr)

bind the socket to a given address

void close()

close the socket

void connect(InetAddress address, int port),

void connect(SocketAddress addr)

bind the socket with a given remote address and the port (the communication is limited only to this address).

void disconnect()

disconnecting a socket.

Metody komunikacyjne gniazda bezpołączeniowego

send(DatagramPacket data)

Enables to send a single packet.

receive(DatagramPacket data) .

holds the current thread until receive a message.

Datagram packet

Pakiet datagramowu (zwykle UDP) jest reprezentowany przez klasę **DatagramPacket**.

Obiekt tej klasy posiada bufory wejściowy i wyjściowy.

Pozwala na konfigurację niektórych parametrów pakietu UDP i edycję pola danych.

Lista najważniejszych pól klasy DatagramPacket:

buf – a byte array for transmitted data,

length – count of data bytes;

offset – data shift from the beginning of the array;

address – an address of the source or destination socket

Datagram packet

Konstrukcja obiektu DatagramPacket

`DatagramPacket(byte[] buf, int length)`

`DatagramPacket(byte[] buf, int length, InetAddress address, int port)`

`DatagramPacket(byte[] buf, int offset, int length)`

`DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)`

`DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)`

`DatagramPacket(byte[] buf, int length, SocketAddress address)`

Gdzie :

buf – a byte array for transmitted data,

length – count of data bytes;

offset – data shift from the beginning of the array;

address – an address of the source or destination socket

Gniazdo Multicast

Klasa **MulticastSocket** została zaprojektowana do wysyłania i odbioru multicastowych pakietów datagramowych.

Połączenie multicastowe nie może być routowane. Przypomina UDP.

Dla pakietów multicastowych adresy źródłowy i docelowy nie są adresami hostów lecz grup multicastowych, do których hosty mogą należeć

Adresy IP grup multicastowych są klasy D i mają standardowe numery portów

Zakres klasy D to <224.0.0.0 239.255.255.255 >

Adres 244.0.0.0 jest zarezerwowany i nie może być używany do standardowej komunikacji

Gniazda Multicast zachowują się tak jak zwykłe gniazda, mają jednak pewne dodatkowe metody

```
joinGroup(InetAddress adr)  
leaveGroup(InetAddress adr)
```

Metody te pozwalają dołączyć hostowi do grupy lub ją opuścić. Członek grupy może wysyłać oraz odbierać pakiety

Multicast - przykład

```
String msg = "Hello";  
InetAddress group = InetAddress.getByName("228.5.6.7");  
MulticastSocket s = new MulticastSocket(6789);  
s.joinGroup(group);  
DatagramPacket hi =  
new DatagramPacket(msg.getBytes(),  
msg.length(), group, 6789);  
s.send(hi); // get their responses!  
byte[] buf = new byte[1000];  
DatagramPacket recv = new DatagramPacket(buf, buf.length);  
s.receive(recv);  
// OK, I'm done talking - leave the group...  
s.leaveGroup(group);
```