Obsługa sieci

URL

- Budowa URL (ang. Uniform Resource Locator):

http://java.sun.com:80/docs/tutorial/index.html?name=networking#DOWNLOADING

- protokół = http
- authority = java.sun.com:80
- host = java.sun.com
- port = 80
- ścieżka = /docs/books/tutorial/index.html
- zapytanie = name=networking
- Nazwa pliku = /docs/books/tutorial/index.html?name=networking
- ref = DOWNLOADING

Klasa URL

- URL url1 = new URL("http://www.gamelan.com/");
  URL(String str);
- URL url2 = new URL("http://www.gamelan.com/pages/");
  URL url21 = new URL(url2, "Gamelan.game.html");
  URL(URL baseURL, String relativeURL)
- URL("http","www.gamelan.com","/pages/Gamelan.net.html"
)URL(String protocol, String adres, String AdresWzględny)

Znaki specjalne a URL

http://foo.com/hello world/

Taki adres nie może być bezpośrednio zapisany bo znaki specjalne jak spacja wymagają specjalnego zakodowania:

```
URL url = new URL("http://foo.com/hello%20world");
```

Wówczas korzystamy z kasy URI

```
URI uri = new URI("http", "foo.com", "/hello world/", "");
```

A następnie konwertujemy URI do URL.

```
URL url = uri.toURL();
```

Wyjątki

Jeśli nieznany protokół lub jeśli dane w konstruktorze wskazują na *null*

```
try {
    URL myURL = new URL(. . .)
} catch (MalformedURLException e) {
    . . .
    // exception handler code here
    . . .
}
```

Uwaga na URL

Uwaga:

- Błąd w implementacji URL – metoda **equals** porównuje jedynie adresy IP a nie nazwy domenowe, stąd

```
public class URITest {
    public static void main(String[] args) throws MalformedURLException {
        URL url1 = new URL("http://kzi.polsl.pl");
        URL url2 = new URL("http://mblachnik.pl");
        if (url1.equals(url2)){
            System.out.println("Adresy sa identyczne");
        } else {
            System.out.println("Adresy są rozne")
        }
    }
}
```

```
Usages | Output - Wyklady (run) | Search Results
run:
Adresy sa identyczne
BUILD SUCCESSFUL (total time: 0 seconds)
```

Parsowanie URL

- *getProtocol* - Zwraca identyfikator protokołu zawartego w URL
- *getAuthority* - Returns the authority component of the URL.
- *getHost* - Zwraca nazwę hosta zawartą w URL.
- *getPort* - Zwraca numer portu zawartego w adresie URL w postaci *int* lub -1 jeśli port nie zdefiniowany
- *getPath* - Zwraca ścieżkę adresu URL.
- *getQuery* - Zwraca zapytanie zawarte w URL.
- *getFile* - Zwraca nazwę pliku zawartą w URL
- *getRef* - Zwraca referencję do danego obiektu w pliku URL.

Odczytywanie z URL

- Jeżeli poprawnie stworzymy URL to możemy z niego czytać dane poprzez metodę *openStream()* która tworzy obiekt klasy java.io.InputStreamobject z którego możemy czytać jak z każdego innego strumienia danych

TCP

Definicja

Socket jest jednym z punktów końcowych dwukierunkowego połączenia komunikacyjnego pomiędzy dwoma programami działającymi w sieci. Socket jest ograniczony do numeru portu tak by warstwa TCP mogła zidentyfikować aplikację docelową do której dane zostały wysłane. Sockety działają na TCP!!!

Java a TCP czyli gniazda TCP

- Implementacja gniazd w Javie jest (jak cała Java) niezależna od platformy dzięki czemu zawsze działa i pozwala na niskopoziomową komunikację
- Java.net.Socket – klasa implementująca gniazda po stronie klienta – podczepia się do danego portu na danym serwerze
- Java.net.ServerSocket – klasa implementująca gniazda po stronie servera – podczepia się pod określony port oraz nasłuchuje czy nie zostało wysłane jakiekolwiek zapytanie.
- Klasa URL i korzystające z niej URLConnection, URLEncoder również wewnętrznie bazują na gniazdach, jednakże w przypadku korzystania z ze standardowych protokołów (http,ftp itp) są one bardziej odpowiednie (czytamy jak ze strumienia)

Obsługa gniazda na kliencie

- Socket – metody:
  - Socket(String nazwa, int port) – inicjalizacja gniazda – podłączenie do hosta o nazwie *nazwa* na dany port
  - Socket(InetAddress ipAddr, int port) – inicjalizacja gniazda – podłączenie do hosta o danym adresie ip na dany port
  - getOutputStream() – zapis do gniazda jako źródła strumienia danych, zwraca obiekt klasy OutputStream
  - getInputStream() - czytanie z gniazda jako źródła strumienia danych, zwraca obiekt klasy InputStream
  - Pozostała pbsługa gniazd jak każdego strumienia

Ogólne zasady obsługi klienta

- Otwórz gniazdo
- Otwórz strumień wejścia i wyjścia do gniazda
- Czytaj i pisz do strumienia w zależności od protokołu serwera
- Zamknij strumienie wejścia wyjścia
- Zamknij gniazdo

```java
public class EchoKlient {
    public static void main(String[] args) throws IOException {
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        String ip = "192.168.1.1";
        try {
            echoSocket = new Socket(ip, Integer.parseInt("4444"));
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(
                    new InputStreamReader(echoSocket.getInputStream()));
        } catch (UnknownHostException e) { System.exit(1);
        } catch (IOException e) { System.exit(1);
        }
        BufferedReader stdIn = new BufferedReader(
                new InputStreamReader(System.in));
        String userInput;
        while ((userInput = stdIn.readLine()) != null) {
            out.println(userInput);
            System.out.println("echo: " + in.readLine());
        }
        if (out!=null) out.close();
        if (in!=null) in.close();
        if (stdIn!=null) stdIn.close();
        if (echoSocket!=null) echoSocket.close();
    }
}
```

Obsługa gniazda na serwerze

- ServerSocket – działa jako usługa i rezerwuje dojście do portu
  - ServerSocket(int port) – podłączenie się do danego portu.
  - Działanie: Metoda accept() nasłuchuje czy nie nadeszło połączenie z jakiegoś hosta. Jeśli połączenie nadeszło tworzony jest Obiekt klasy Socket, wówczas postępujemy tak jak na kliencie obsługując strumień danych.

```java
public class EchoServer {

    public static void main(String[] args) throws IOException {
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        int port = 4444;
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(port);
        } catch (IOException e) { System.exit(1); }
        while (true) {
            try {
                echoSocket = serverSocket.accept();
            } catch (IOException e) { System.exit(1); }
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(
                    new InputStreamReader(echoSocket.getInputStream()));
            String input;
            while ((input = in.readLine()) != null) {
                out.println(input);
                System.out.println("echo: " + input);
            }
            if (out!=null) out.close();
            if (in!=null) in.close();
            if (echoSocket!=null) echoSocket.close();
        }
    }
}
```

Uwaga Serwer obsługuje tylko jeden wątek!!!
W praktyce konieczna wielowątkowość (Patrz UDP)

Obsługa protokołu UDP

UDP

Protokół UDP jest:

- Bezpołączeniowy
- Bezstanowy

Ale za to jest szybki!!!

Java + UDP

Klasy do obsługi protokołu UDP

- DatagramSocket – tworzy socket do obsługi ruchu UDP.
  - Przypina się na lokalnej maszynie do określonego portu zdefiniowanego jako argument konstruktora lub też port przydzielany jest przez system.
- Posiada dwie metody b. ważne metody
  - send – metoda służąca do wysyłania pakietów
  - receive – metoda służąca do odbierania pakietów

Pakiety reprezentowane są przez klasę DatagramPacket

UDP klasa DatagramPacket

DatagramPacket – klasa obsługująca datagram który zostanie wysłany lub odebrany.

DatagramPacket() – konstruktor przy odbieraniu wymaga określenia bufora (tablicy bajtów) i długości pakietu nie większej od rozmiaru tablicy lub w przypadku wysyłania wiadomości dodatkowo określamy adres ip adresata i nr portu na który chcemy wysłać wiadomość.

Przy wysyłaniu wiadomości tekstowej String musimy go najpierw przekształcić na ciąg bajtów poprzez metodę zaimplementowaną w klasie String typu getBytes() – metoda ta dokonuje konwersji zależnego od platformy Stringa na ciąg bajtów.

Użyteczne metody:

- getAddress() – zwraca adres IP zdalnego hosta
- getPort() – zwraca port zdalnego hosta
- getData() – zwrócenie referencji do obiektu bufora danych
- getLength() – zwrócenie długości pakietu

```java
public class UDPSerwer implements Runnable {
    byte[] ip = {127, 0, 0, 1};
    int port = 7;
    DatagramSocket socket;

    public UDPSerwer() {
        try {
            socket = new DatagramSocket(port);
        } catch (SocketException e) {}
    }

    @Override public void run() {
        try {
            while (true) {
                byte[] buf = new byte[256];
                DatagramPacket packet = new DatagramPacket(buf, buf.length);
                socket.receive(packet);
                InetAddress remIP = packet.getAddress();
                int remPort = packet.getPort();
                String msg = new String(packet.getData(), 0, packet.getLength());
                System.out.println("Od: " + remIP + " port: " + remPort
                        + " wiadomość: " + msg);
                msg += " Odbite";
                packet.setData(msg.getBytes());
                socket.send(packet);
            }
        } catch (IOException e) {
        } finally { socket.close(); }
    }

    public static void main(String[] arg) {
        new Thread(new UDPSerwer()).start();
    }
}
```

Po stronie serwera socket przypinamy do portu

Tworzenie datagramu do odbioru

Bufor datagramu

Wysłanie danych

Odczyt danych z pakietu przekształcenie odebranej wiadomości na tekst

```
public class UDPClient {

    public static void main(String[] arg) {
        int port = 7;
        String msg = "Ala ma kota";
        byte[] ip = {127, 0, 0, 1};
        DatagramSocket socket = null;
        try {
            byte[] udpMsg = msg.getBytes();
            System.out.println("Wiadomość wysłana: " + msg);
            socket = new DatagramSocket();
            DatagramPacket packetSend = new DatagramPacket(udpMsg, udpMsg.length,
                    InetAddress.getByAddress(ip), port);
            socket.send(packetSend);
            byte[] udpMsgR = new byte[256];
            DatagramPacket packetReceive = new DatagramPacket(udpMsgR, udpMsgR.length);
            socket.receive(packetReceive);
            String msgRs = new String(packetReceive.getData(), 0, packetReceiv tLength());
            System.out.println("Wiadomość odebrana:"  msgRs);

        } catch (SocketException e) {
        } catch (UnknownHostException e) {
        } catch (IOException e) {
        } finally { if (socket!=null) socket.close();}
    }
}
```

Stworzenie socketu. System sam przydziela port

Stworzenie datagramu do wysyłki. Podajemy adres IP i port adresata

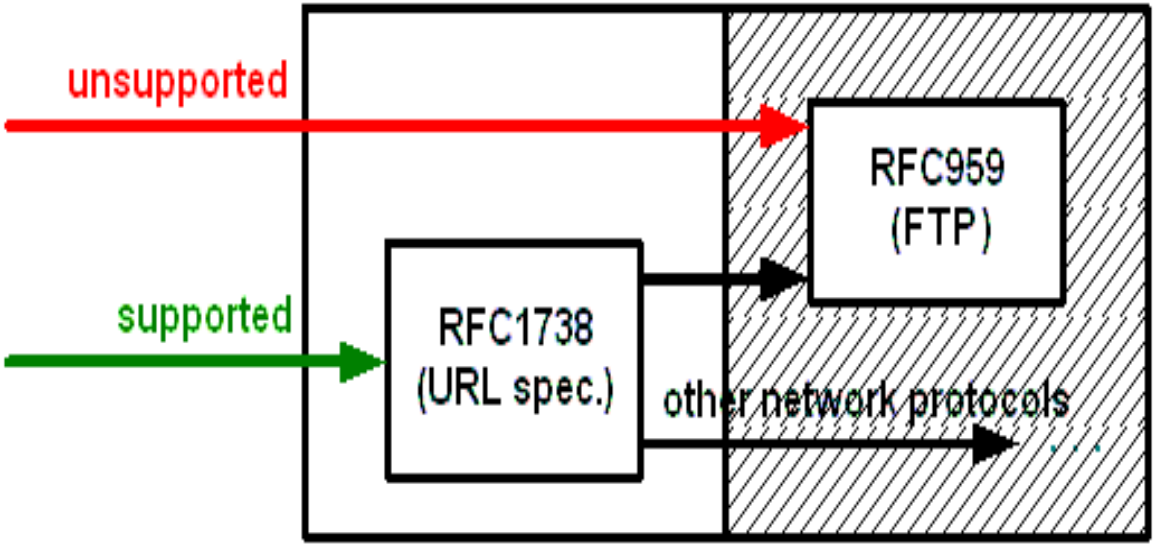
Odbiór wiadomości

Przygotowanie Datagramu do odbioru wiadomości

Konwersja odebranej wiadomości na string

Protokoły sieciowe

Rozwiązania SUN'a



1. Wykożystanie klasy URL do obsługi FTP.
2. Klasa URL umożliwia określenie protokołu FTP.
3. Na podstawie adresu URL tworzone jest połączenie – Klasa URLConnection
4. Klasa URLConnection umożliwiaia utworzenie strumieni wejściowego i wyjściowego

Przykład

```
public class FtpTest {

    public static void main(String[] args) {
        try {
            URI uri = new URI("ftp://user01:pass1234@ftp.foo.com/README.txt");
            URL url = uri.toURL();
            URLConnection urlc = url.openConnection();
            InputStream is = urlc.getInputStream();
            OutputStream os = urlc.getOutputStream();
            BufferedReader strIn = new BufferedReader(new InputStreamReader(is));
            String str;
            while ((str = strIn.readLine()) != null) {
                System.out.println(str + "\n");
            }
        } catch (URISyntaxException ex) {
        } catch (MalformedURLException e) {
        } catch (IOException e) {
        }
    }
}
```

Projekt Apache Jakarta

Implementacje różnych protokołów sieciowych w tym:

- FTP/FTPS
- NNTP
- SMTP
- POP3
- Telnet
- TFTP
- Finger
- Whois
- rexec/rcmd/rlogin
- Time (rdate) and Daytime
- Echo
- Discard
- NTP/SNTP

Klasa FTPClient

Przydatne funkcje:

- connect(String server) – Połącz z serwerem
- disconnect() – rozłącz z serwerem
- login(String username, String password) - logowanie
- logout() – wylogowywanie z serwera
- changeWorkingDirectory(String pathname) – zmiana aktualnego katalogu na zdalnym serwerze ftp
- FTPFile[] listFiles() – Lista plików z aktualnego katalogu serwera FTP
- String[] listNames() – Lista nazw plików z aktualnego katalogu serwera FTP
- String[] listNames(String pathname) – jw.. Ale z określonego katalogu serwera
- makeDirectory(String pathname) – Stwórz katalog
- rename(String from, String to) – Zmiana nazwy pliku
- storeFile(String remoteName, InputStream local) – upload plik z określonego java.io.InputStreama'a, pod określoną nazwę wpisaną w polu remoteName);
- retrieveFile(java.lang.String remoteName, OutputStream local) – Pobranie pliku z serwera określając jego zdalną nazwę do określonego Java.io.OutputStreama.

Przykład

```
public class FtpJakarta {

    public static void main(String[] args) {
        try {
            String server = "ftp.pl";
            String username = "uzytkownik";
            String password = "haslo";
            FTPClient f = new FTPClient();
            f.connect(server);
            f.login(username, password);
            FTPFile[] files = f.listFiles();
            System.out.println("Listing files:");
            for (FTPFile file : files) {
                System.out.println(file.getName());
            }
            f.logout();
            f.disconnect();
        } catch (SocketException e) {
        } catch (IOException e) {
        }
    }
}
```

JavaMail

Pakiet javax.mail

Wysyłanie wiadomości emaili

Proces wysyłania wiadomości:

1. Stworzenie sesji z odpowiednimi właściwościami
2. Stworzenie wiadomości
3. Określenie parametrów wiadomości: Od, Do, Treść, Temat itp.
4. Wysłanie wiadomości

Protokół SMTP

- Otwarcie sesji i zdefiniowanie jej parametrów
- Metody: *Session.getDefaultInstance* lub *Session.getDefaultInstance* – Pozwalają na otwarcie sesji

  Session session = Session.getDefaultInstance(props, auth);

  props – obiekt reprezentujący właściwości sesji klasy *Properties*
  auth – obiekt reprezentujący autentykację klasy *Authenticator* czyli umożliwiający zwracanie nazwy użytkownika i hasła

- Stworzenie obiektu właściwości sesji metoda *System.getProperties()*:

  Properties props = System.getProperties(); //Stworzenie obiektu

- *Określenie parametrów:*

  props.put("mail.smtp.host", smtpServer); //Przypisanie właściwości – domyślnego adresu smtp

Autentykacja

- Domyślnie serwery SMTP nie obsługiwały autoryzacji co powodowało że każdy mógł wysłać maila podszywając się pod dowolnego użytkownika, dlatego w celu zabezpieczenia się coraz większą popularność zyskuje „podpisywanie” sesji co wymaga podania nazwy użytkownika i hasła.
- W Javie jest to reprezentowane przez obiekt klasy *Authenticator*
- W praktyce konieczne jest stworzenie własnej klasy rozszerzającej klasę *Authenticator* co w praktyce sprowadza się do przeciążenia metody *getPasswordAuthentication()* .

Utworzenie wiadomości

- Klasa *Message* lub *MimeMessage* – Wiadomość typu Mime

  *Message msg = **new** MimeMessage(session);*

- Określenie parametrów wiadomości:

Od kogo

msg.setFrom(new InternetAddress(from));

Do kogo

msg.setRecipients(Message.RecipientType.TO,InternetAddress.parse(to, false));

Stała Message.RecipientType.TO/CC/BCC określa typ odbiorców

Tytuł

msg.setSubject(subject);

Treść wiadomość

msg.setText(body);

Nagłówek wiadomości

msg.setHeader("X-Mailer", "LOTONtechEmail");

Data wiadomości

msg.setSentDate(new Date());

Wysłanie wiadomości

- Wysłanie wiadomości:

  Transport.send(msg);

```java
public class WysylanieMaili {

    public static void main(String args[])
            throws AddressException, MessagingException {
        String smtpServer = "x.pl";
        String to = "x@x.pl";
        String from = "x@x.pl";
        String subject = "Test";
        String body = "Tresc maila";
        final String userName = "";
        final String passwd = "";

        Properties props = System.getProperties();
        props.put("mail.smtp.host", smtpServer);
        Authenticator auth = new Authenticator() {
            @Override protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(userName, passwd);
            } };
        Session session = Session.getDefaultInstance(props, auth);
        Message msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress(from));
        msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse(to, false));
        msg.setSubject(subject);
        msg.setText(body);
        msg.setHeader("X-Mailer", "LOTONtechEmail");
        msg.setSentDate(new Date());
        Transport.send(msg);
        System.out.println("Message send OK.");
    }
}
```

Odczyt wiadomości – protokół POP3

1. Utworzenie sesji z odpowiednimi parametrami i autentykacją (patrz SMTP)
2. Określenie protokołu używanego do podpięcia się do składowiska wiadomości (POP3/IMAP).
3. Połączenie ze składowiskiem danych używając odpowiednich danych autoryzacyjnych jak użytkownik, hasło, nazwa serwera
4. Pobranie i podłączenie się do folderu zawierającego wiadomości
5. Otwarcie folderu
6. Pobranie z folderu odpowiednich wiadomości
7. Po skończonej pracy zamknięcie folderu i składowiska

Utworzenie sesji i podłączenie się do składowiska wiadomości

Stworzenie sesji

```
Properties props = System.getProperties();
Session session =
Session.getDefaultInstance(props, new
SomeAuthenticator(popUser,popPassword));
```

Podłączenie się do składowiska

```
store = session.getStore("pop3");
store.connect(popServer, popUser,
popPassword);
```

Pobranie i podłączenie się do folderu

Pobranie folderu

```
folder = store.getDefaultFolder();
```

Wybór folderu skrzynki odbiorczej

```
folder = folder.getFolder("INBOX");
```

Podłączenie się do folderu

```
folder.open(Folder.READ_ONLY);
```

Pobór wiadomości czyli obsługa folderu

Pobór wiadomości o folderze:

```
System.out.println(„Masz :" + folder.getMessageCount() + "
   wiadomości.");


System.out.println(„W tym :" + folder.getNewMessageCount() + "
   nowych wiadomości.");
System.out.println(„i :" + folder.getUnreadMessageCount() + "
   nieprzeczytanych wiadomości.");
```

Odczyt wiadomości

```
Message[] msgs = folder.getMessages(1,5);
for (int msgNum = 0; msgNum < msgs.length; msgNum++)
{
printMessage(msgs[msgNum]);
}
```

```
public class Pop {
    public static void main(String args[]) {
        final String user = "user", pass = "passwd", srv = "127.0.0.1";
        Store store = null; Folder folder = null;
        try {
            Properties props = System.getProperties();
            Authenticator auth = new Authenticator() {
                @Override protected PasswordAuthentication
                        getPasswordAuthentication() {
                    return new PasswordAuthentication(user, pass); } };
            Session session = Session.getDefaultInstance(props, auth);
            store = session.getStore("pop3");
            store.connect(srv, user, pass);
            folder = store.getDefaultFolder();
            if (folder == null) { throw new Exception("No default folder"); }
            folder = folder.getFolder("INBOX");
            if (folder == null) { throw new Exception("No POP3 INBOX"); }
            folder.open(Folder.READ_ONLY);

            Message[] msgs = folder.getMessages();
            for (int msgNum = 0; msgNum < msgs.length; msgNum++) {
                String tytul = msgs[msgNum].getSubject();
                System.out.println(tytul);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            try {
                if (folder != null) { folder.close(false); }
                if (store != null) { store.close(); }
            } catch (Exception ex2) { }
        }
    }
}
```