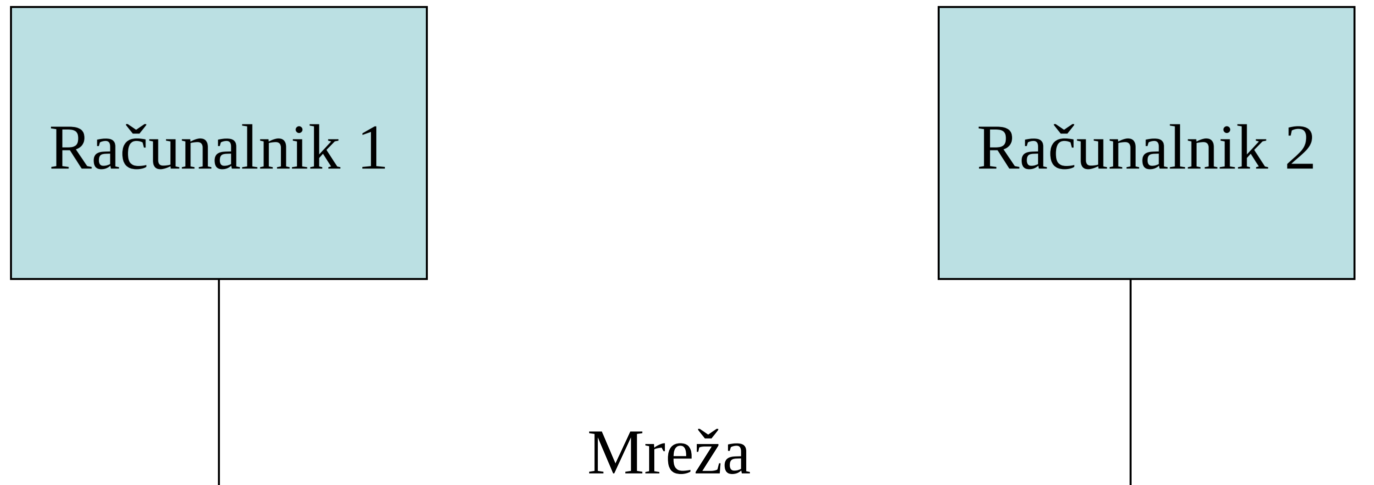
Mrežne aplikacije z Javo

- Java je prvi programski jezik z mislimi na omrežje
- Programiranje omrežnih aplikacij lajša paket java.net
- Ta paket vsebuje IP, vtičnice (sockets), rokovanje z DNS, URL

- Razumeti moramo 5 pojmov:

  1. Računalnik (Host)
  2. Vrata (Port)
  3. Osnovno zgradbo odjemalec-strežnik
  4. Protokol
  5. "Marshalling"

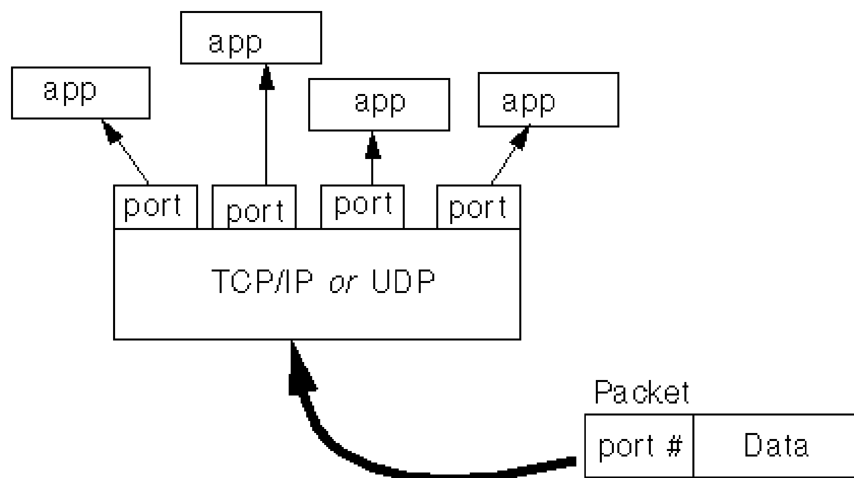
Računalniki (hosts)

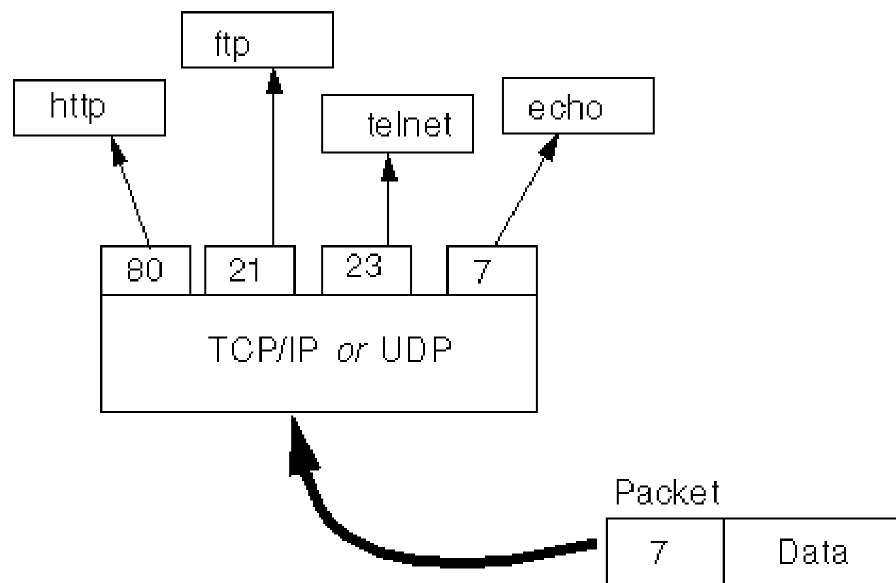


Računalnike identificiramo z njihovim imenom (Hostname) in IP naslovom

Vrata (Ports)

- Na računalnikih lahko istočasno teče več aplikacij
- Vrata (Port) delujejo za aplikacije kot "poštni nabiralniki
  - Vrata so oštevilčena od 0 do 65536
- Nekatera vrata so dobro znana, tako na primer telnet uporablja vrata 23.
- Druga vrata so alocirana dinamično
- Vsa obvestila imajo naslov <host-id, port>





Port Scanner

Psevdo koda

```
input host name
check that host exists
for port = 1 to 256 do
    try to open socket to port
    if socket opens, display port number
    close socket
endfor
```

PortScanner - Java

Po tem modelu imamo dve vrsti programov

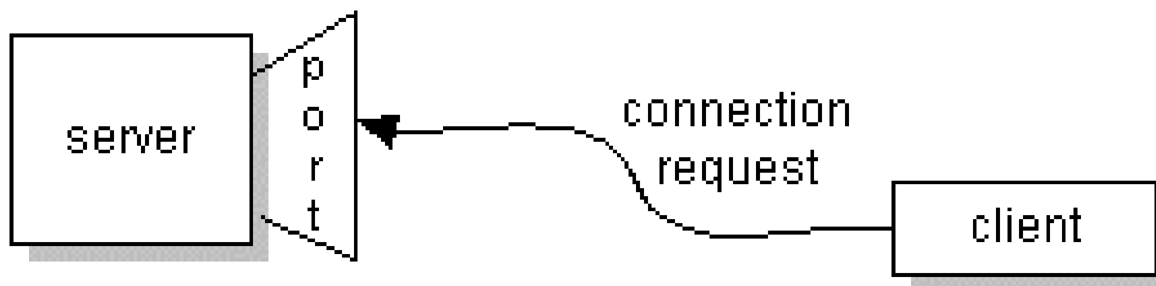
- Programi - strežniki čakajo druge programe (odjemalce), da zaprosijo za servis
- Programi – odjemalci poiščejo stik s strežniki in jih zaprosijo za servis

Primer: spletni brkljalniki

- Uporabnik klikne na povezavo, brkljalnik (odjemalec) se poveže s strežnikom na tej povezavi
- Strežnik poišče zahtevano datoteko in jo pošlje brkljalniku (odjemalcu)

Osnovna struktura strežnika (1)

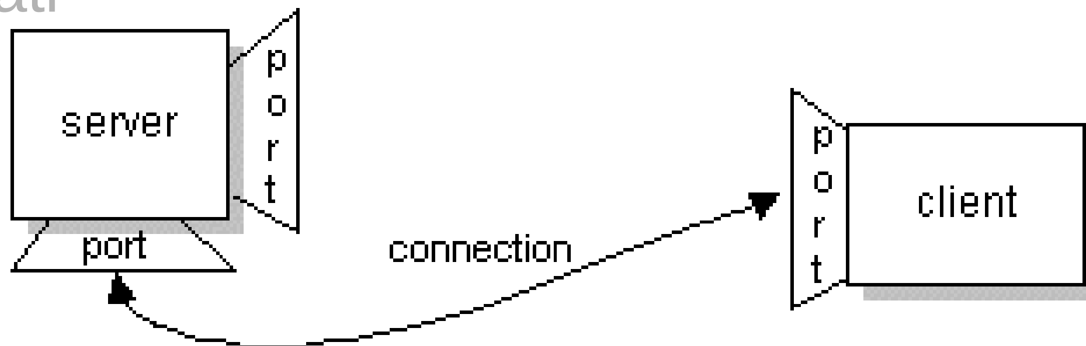
- Strežnik posluša (listen) na znanih vratih, če se kdo hoče povezati



- Ko pride do povezave, pove strežnik "programu-delavcu", da naj obravnava povezavo.
  - Delavec za to povezavo pogosto alocira nova vrata.
  - Povezava bo trajala dalj časa ali pa le za izvedbo neke operacije
  - Nekateri strežniki tvorijo za vsak nov zahtevek novega "delavca"

Osnovna struktura strežnika (2)

- Strežnik posluša (listen) na znanih vratih.če se kdo hoče povezati



- Ko pride do povezave, pove strežnik "programu-delavcu", da naj obravnava povezavo.
  - Delavec za to povezavo pogosto alocira nova vrata.
  - Povezava bo trajala dalj časa ali pa le za izvedbo neke operacije
  - Nekateri strežniki tvorijo za vsak nov zahtevek novega "delavca"

Osnovna struktura odjemalca

- Odjemalec najprej ugotovi naslov strežnika (host, port)
- Odjemalec se poveže s strežnikom
- Odjemalec se predstavi strežniku
- Odjemalec pošlje zahtevek strežniku
- Odjemalec čaka na odgovor strežnika
- Odjemalec zapre povezavo

Protokoli

- Protokoli definirajo pravilno interakcijo med stranema, ki komunicirata
  - Kako vzpostaviti povezavo
  - Format veljavnega zahtevka
  - Katere vrste zahtevkov so veljavne
  - Format odgovora
  - Ali imajo povezave večkratne zahtevke, itd.
- Primer: HTTP

Marshalling (urejanje, serializacija)

- Omrežja so zelo heterogena:
  - Računalniki imajo lahko različne operacijske sisteme
  - Računalniki imajo lahko različne podatkovne formate (na primer float)
  - Računalniki imajo lahko različna zaporedja bytov
  - Odjemalci in strežniki morda ne souporabljajo enak tip podatkov
- "Marshalling" je proces predelave preprostih in kompleksnih podatkovnih tipov v obliko, primerno za prenos. ("unmarshalling" je obratna predelava.
- To delo za nas opravi kar Java.
  - DataStreams in ObjectStreams

Razred InetAddress

```
public byte[] getAddress();
public static InetAddress[] getAllByName(String host);
public static InetAddress getByName(String host);
public String getHostName();
public static InetAddress getLocalHost();
```

Ta razred predstavlja naslov Internet Protocol (IP).

Aplikacije naj bi za tvorbo novih primerkov InetAddress uporabljale metode getLocalHost(), getByName(), ali getAllByName().

Če računalnika ne najdemo, pride do izjeme UnknownHostException

Primer uporabe

```
import java.net.*;

public class HostInfo {
  public static void main( String args[] ) {
    InetAddress ipAddr;

    try {
      ipAddr = InetAddress.getLocalHost();
      System.out.println( "To je" + ipAddr );
    }
    catch ( UnknownHostException ex ) {
      System.out.println( "Neznani računalnik" );
    }
    System.in();
}
```

Primer

Primer uporabe (2)

```
import java.net.*;

public class Resolver {
  public static void main( String args[ ] ) {
    InetAddress ipAddr;

    try {
      ipAddr = InetAddress.getByName( args[0] );
      System.out.print( "IP naslov = " + ipAddr + "\n " );
    }
    catch ( UnknownHostException ex ){
      System.out.println( "Neznani računalnik:" );
    }
  }

}
```

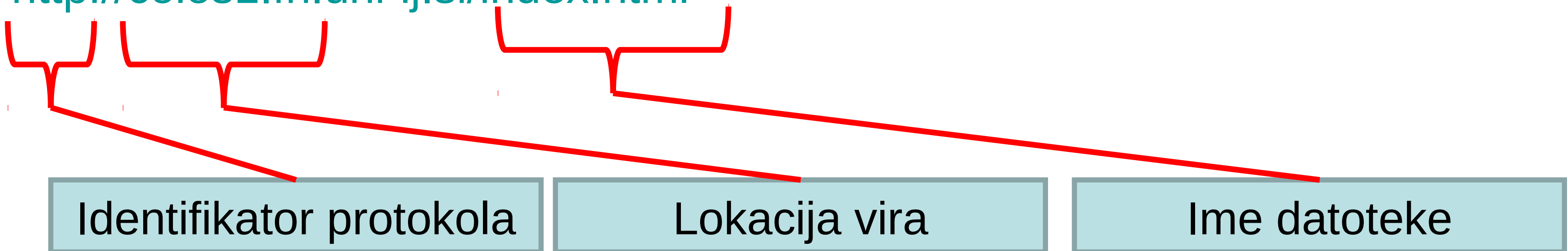
Primer

URL

URL je akronim za Uniform Resource Locator. To je naslov nekega vira na internetu.

URL ima obliko niza, ki opisuje, kako najdemo vir na Internetu. URL ima dve glavni komponenti: protokol, ki ga potrebujemo za dostop do vira in lokacijo vira. Primer:

http://colos1.fri.uni-lj.si/index.html



Javanski program , ki interaktira z internetom, lahko uporablja razred *URL* v paketu java.net.

Standardni protokoli so: http, ftp, mailto, news, gopher in file

Tvorba URL

- Z instanco iz razreda URL lahko v Javi predstavimo nek URL

URL colos = new URL(http://colos1.fri.uni-lj.si);

- Tako tvorjeni objekt URL predstavlja absolutni URL. Lahko pa tvorimo objekte URL s pomočjo relativnega naslova

URL ptt = new URL(colos, “ERI”);

- Vsi konstruktorji URL vrnejo MalformedURLException, če skušamo nasloviti neznani protokol.

Metode URL

Razred URL nudi več metod za povpraševanje objektov URL:

- Lahko za dani URL zvemo protokol, ime računalnika (host name), številko vrat (port number) in ime datoteke
- Vsi URL ne vsebujejo teh komponent!

```
public String getProtocol ( )
public String getHost( )
public String getPort( )
public String getFile ( )
```

Branje direktno iz URL

Ko uspešno tvorimo URL, lahko na primer beremo njegovo vsebino. Zato najprej pokličemo njegovo metodo openStream.

Metoda openStream() vrne objekt java.io.InputStream in vsebino lahko beremo kot vhodni tok.

Primer na naslednji strani bere iz ukazne vrstice vrsto imen računalnikov in URL-jev. S pomočjo vsakega argumenta ukazne vrstice skuša tvoriti URL, se povezati s specificiranim strežnikom, naložiti z njega podatke in jih izpisati na zaslon.

Primer branja iz URL

```
import java.net.*;
import java.io.*;
public class Webcat {

  public static void main(String[] args) {
     for (int i = 0; i < args.length; i++) {
         try {
            URL u = new URL(args[i]);
            InputStream is = u.openStream();
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            String theLine;
            while ((theLine = br.readLine()) != null)  System.out.println(theLine);
         }
         catch (MalformedURLException ex) { System.err.println(ex); }
         catch (IOException ex) { System.err.println(ex);   }
     }
  }

}
```

Primer bere iz ukazne vrstice vrsto imen računalnikov in URL-jev. S pomočjo vsakega argumenta ukazne vrstice skuša tvoriti URL, se povezati s specificiranim strežnikom, naložiti z njega podatke in jih izpisati na zaslon.

Povezava na URL

Po uspešni tvorbi URL objekta lahko uporabimo njegovo metodo openConnection() in se nanj navežemo.

Ko se povežemo na URL, vzpostavimo komunikacijsko povezavo med našim javanskim programom in URL na omrežju.

primer povezave s strežnikom FRI:

```
URL  friURL = new URL (http://www.fri.uni-lj.si/);
URLconnection friConnection = friURL.openConnection();
```

Tvorjeni objekt URLConnection je naša povezava. Če se tako uspemo povezati na URL, lahko beremo in pišemo v povezavo s pomočjo objekta URLConnection

Primer uporabe take povezave je na na naslednji strani

Primer: URLConnectionReader

```
import java.net.*;
import java.io.*;

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {

        URL googleURL = new URL(http://www.fri.uni-lj.si/);
        URLConnection friConnection = friURL.openConnection();

        BufferedReader in = new BufferedReader(new
                InputStreamReader(friConnection.getInputStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);

        in.close();
    }
}
```

Vzpostavimo povezavo s strežnikom FRI in beremo, kar dobimo z njega.

Zakaj URLConnection?

Primer z uporabo URLConnection opravlja natanko enako nalogo, kot primer, ko smo iz URL direktno brali, brez povezave.

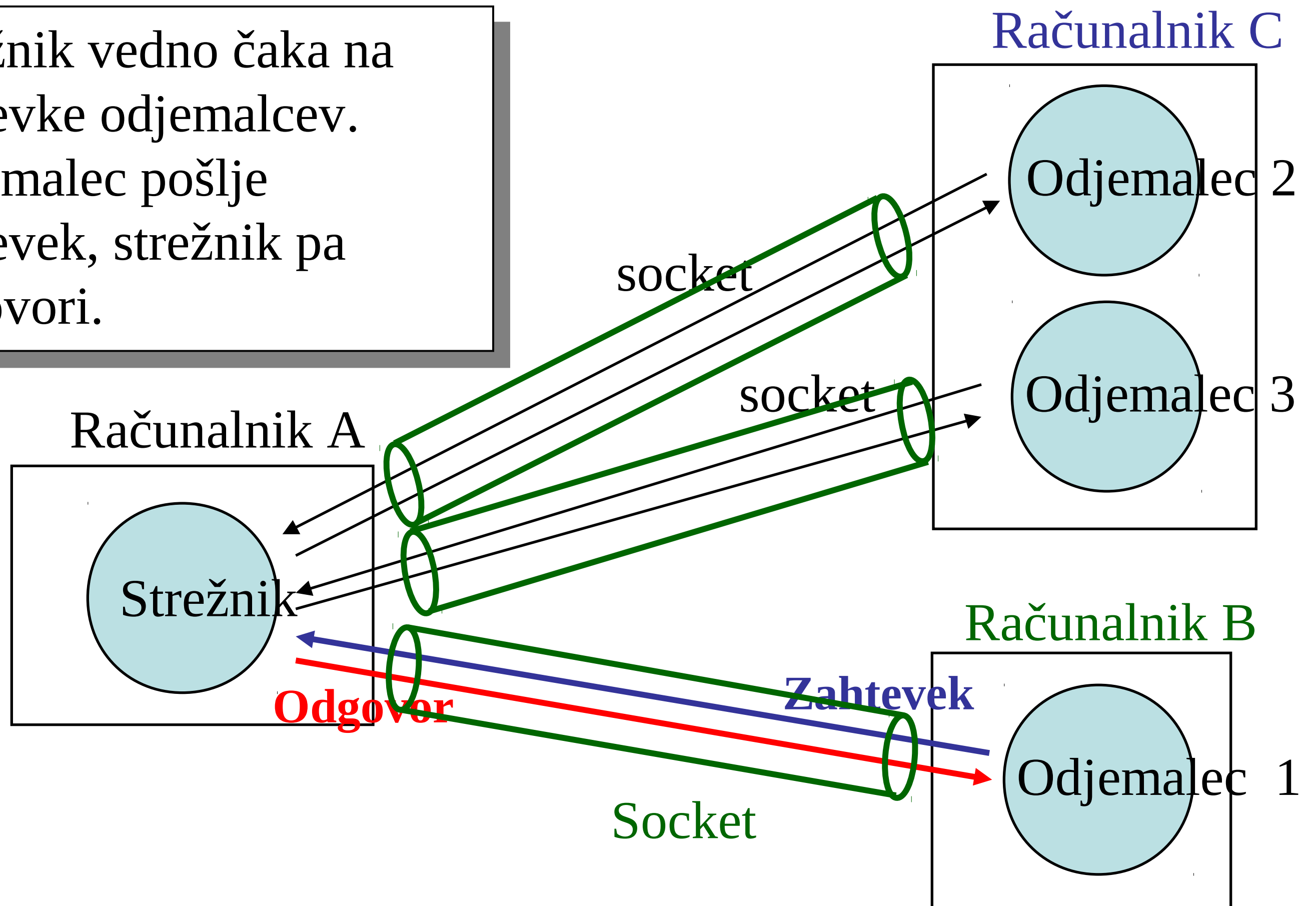
Vendar je uporaba URLConnection močnejša, saj jo lahko uporabimo tudi za druge naloge, na primer za istočasno pisanje na URL.

Pisanje v URLConnection

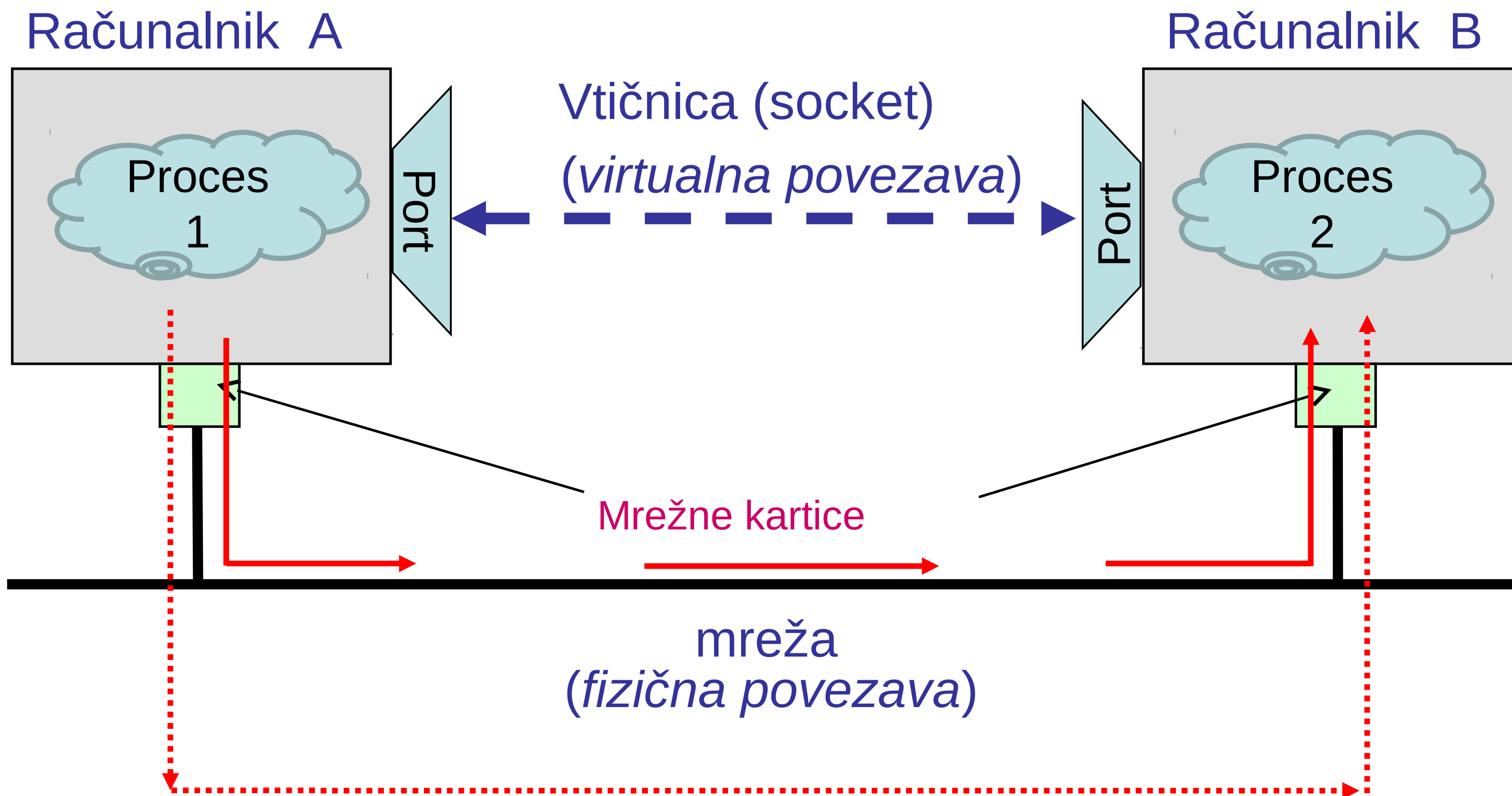
Javanski program mora opraviti naslednje operacije:

1. Tvoriti URL
2. Odpreti povezavo na URL
3. Na tej povezavi vzpostaviti zmožnost izhoda
4. Na povezavi pridobiti izhodni tok
5. Pisati v izhodni tok
6. Zapreti izhodni tok

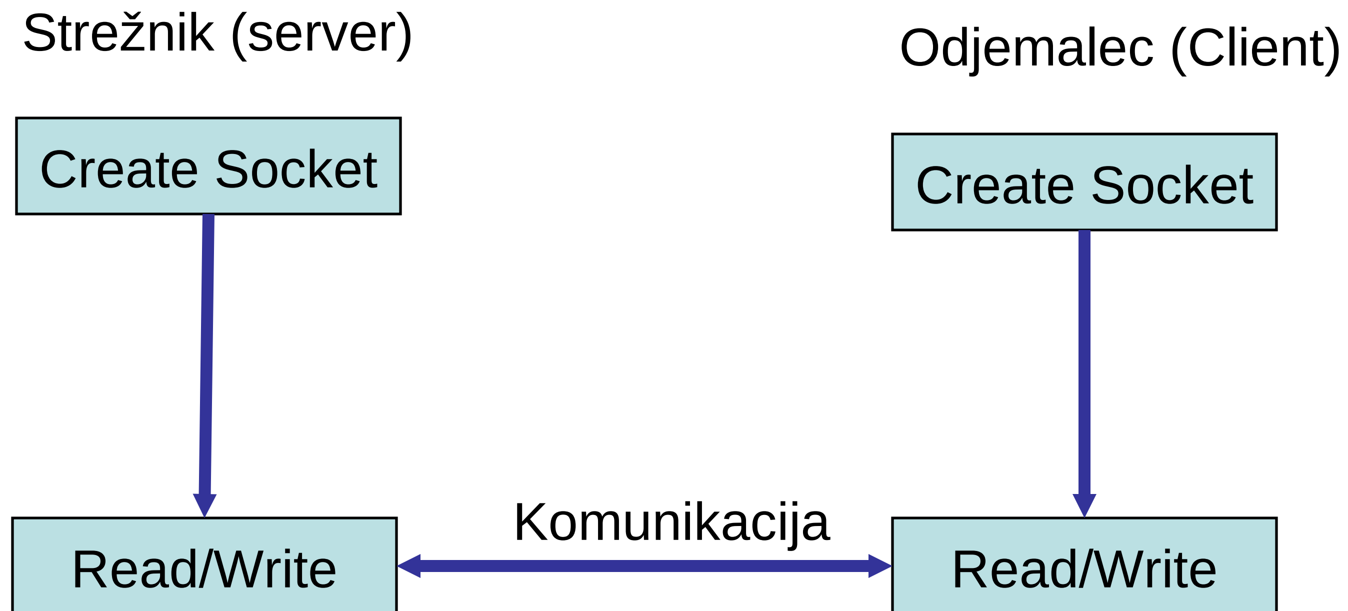
Vtičnice kot model odjemalec - strežnik



Komunikacija preko vtičnic



Komunikacija preko vtičnic (2)



Naloga vtičnic za odjemalce

Vtičnice (sockets) omogočajo programom komunikacijo z operacijskim sistemom in preko njega z omrežjem

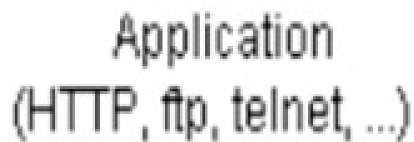
Osnovne operacije vtičnice:

- Povezava z oddaljenim računalnikom (host)
- Pošiljanje podatkov
- Prejemanje podatkov
- Zapiranje povezave
- Navezavo na vrata
- Poslušanje (listen) prihajajočih podatkov
- Sprejem povezav iz oddaljenih računalnikov na navezana vrata.

Vtičnice razumejo dve vrsti protokolov:

- TCP Transmission Control Protocol
- UDP User Datagram Protocol

Nekaj osnov račun. komunikacij



Računalniki na internetu komunicirajo med seboj ali s protokolom TCP (Transmission Control Protocol) ali s protokolom UDP (User Datagram Protocol).

Ko pišemo javanski program, ki naj bi komuniciral preko omrežja, programiramo aplikacijsko plast.

Tipično nam ni treba skrbeti o plasteh TCP oziroma UDP.

Uporabimo raje paket java.net. Njegovi razredi nudijo mrežno komunikacijo, neodvisno od sistema.

Vendar pa moramo za odločitev, katere razrede bomo uporabili, razumeti razliko med TCP in UDP.

Nekaj o UDP

UDP protokol nudi komunikacijo med dvema aplikacijama na omrežju, ki pa ni garantirana.

UDP je drugačen od TCP. Namesto povezave med dvema aplikacijama pošilja neodvisne pakete podatkov – datagrame.

Pošiljanje datagramov je podobno pošiljanju pisem preko poštnega servisa. Vrstni red posredovanja obvestil ni važen in ni garantiran. Vsako obvestilo je neodvisno od drugih obvestil.

Razred Socket

- Konstruktorji:
  - public Socket (String host, int port)
  - public Socket (InetAddress address, int port)

- Povpraševanje o lastnostih vtičnice:

  public InetAddress getInetAddress ( )
  Vrne naslov, na katerega je vtičnica povezana

  public int getPort( )
  Vrne oddaljena vrata, na katera je navezana vtičnica

  public int grtLocalPort( )
  Vrne lokalna vrata, na katera je vtičnica navezana

- Pretok podatkov preko vtičnice
  - public InputStream getInputStream( )
  - public OutputStream getOutputStream( )

- Zapiranje vtičnice
  - public void close( )

Izjeme vtičnic (Sockets Exceptions)

BindException

Do te izjeme pride, če poskuša vtičnica uporabiti lokalna vrata, ki so zasedena, ali pa nima dovolj pravic za uporabo teh vrat.

ConnectException

Ni povezave do oddaljenega računalnika. Na danih vratih ne teče noben strežni program.

NoRouteToHostException

Po omrežju ni poti do oddaljenega računalnika

SocketException

Nekaj je bilo z vtičnico narobe, pa to ni nobeden od prej navedenih vzrokov.

Vtičnice za strežnike

- Razlika med vtičnicami odjemalcev in vtičnicami strežnikov je, da vtičnice strežnikov čakajo na povezavo

- Osnoven življenski cikel strežnika:

  1. Tvorimo novo vtičnico ServerSocket
  2. Vtičnica posluša za zahtevke povezav tako, da je blokirana z uporabo stavka accept
  3. Strežnik interaktira z odjemalcem
  4. Odjemalec ali strežnik prekineta povezavo
  5. Strežnik se povrne na korak 2.

Metode vtičnic za strežnike

Tvorba vtičnice strežnika

```
public ServerSocket ( int port)
```

Tvorba vtičnice strežnika, ki posluša lokalna vrata:

```
public ServerSocket (int port, int backlog)
```

Sprejem in zapiranje povezav:

```
public Socket accept( )  // čaka na povezavo
public void close( )         //zapre vtičnico (socket)
```

Primer: Servis Daytime

Večina strežnikov UNIX izvaja na TCP vratih 13 storitev daytime .

```
sasa> telnet kiev.cs.rit.edu 13
Trying 129.21.38.145...
Connected to kiev.
Escape character is '^]'.
Fri Feb  6 08:33:44 1998
Connection closed by foreign host.
```

Javanski odjemalec za storitev daytime zlahka sprogramiramo. Program mora le vzpostaviti TCP povezavo z gostiteljskim računalnikom preko vrat 13 .

TakoTCP povezavo naredimo s pomočjo razreda Socket.

Primer odjemalca DayTime

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

public class DayTimeClient {
  static int dayTimePort = 13;

  public static void main(String args[]) {
    try {
      Socket sock = new Socket (args[0], dayTimePort);
      BufferedReader din = new BufferedReader(
        new InputStreamReader(sock.getInputStream()));
      String rTime = din.readLine();
      System.out.println(rTime);
      sock.close();
    }
    catch (Exception e) {}
  }
}
```

Primer

Strežnik DayTime

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

public class DayTimeServer {
  public static void main(String argv[]) {
    try {
      Date today = new Date();
      InetAddress localHost = InetAddress.getLocalHost();
      ServerSocket listen = new ServerSocket(0);
      System.out.println("Poslusam vrata:  "+listen.getLocalPort());

      for(;;) {
        Socket clnt = listen.accept();
        System.out.println(clnt.toString());
        PrintWriter out = new PrintWriter(clnt.getOutputStream(), true);
        out.println(today);
        clnt.close();
      }
    } catch(Exception e) {}}}
```

Primer

DayTimeServer v akciji

Izpis strežnika daytime :

```
kocka> java DayTimeServer
 Listening on port:  36109
 Socket[addr=cobalt/129.21.37.176,port=32875,localport=36109]
 Socket[addr=localhost/127.0.0.1,port=36112,localport=36109]
```

Izpis klijenta:

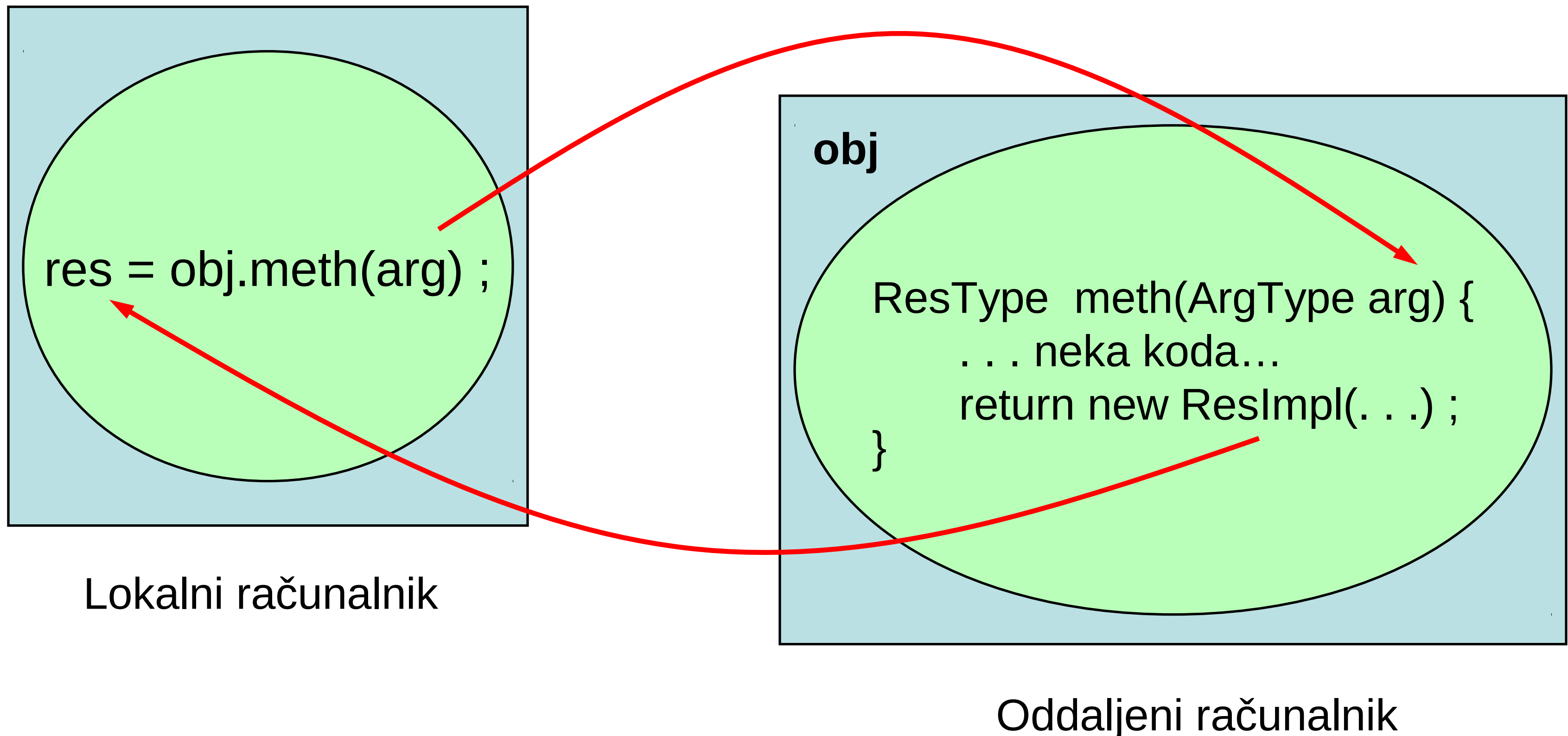
```
 sasa> telnet kocka 36109
Trying 129.21.38.145...
Connected to kocka.
Escape character is '^]'.
Fri Feb 06 09:53:00 EST 1998
Connection closed by foreign host.
```

Remote Method Invocation

- Java RMI je mehanizem, ki omogoča javanskemu programu, ki teče na enem računalniku uporabo metode objekta, ki živi na drugem računalniku.
- Klicanje oddaljene metode je podobno klicu navadne javanske metode.
  - pri klicu oddaljene metode posredujemo tej argumente, ki so izračunani v okolju lokalnega računalnika. Oddaljena metoda vrača vrednosti, ki so izračunane v okolju oddaljenega računalnika. Sistem RMI poskrbi za transparentno komunikacijo.

Slika porazdeljenega objekta

- Koda, ki teče na lokalnem računalniku, ima oddaljeni naslov na objekt **obj** na oddaljenem računalniku:



Oddaljeni vmesnik (Remote Interface)

- V RMI je minimum informacije, ki si jo delita "odjemalec" in "strežnik", skupni vmesnik. Ta določa visokonivojski "protokol" za komunikacijo med računalnikoma.
- Oddaljeni vmesnik je normalni javanski vmesnik, ki je izpeljan iz **java.rmi.Remote**.
  - Opomba: Do konstruktorjev, statičnih metod in spremenljivih polj oddaljenega objekta ne moremo priti.
- Vse metode v oddaljenem vmesniku morajo imeti deklarirano izjemo **java.rmi.RemoteException**.

Preprost primer

Imejmo datoteko **MessageWriter.java** z naslednjo definicijo vmesnik. Ta vmesnik definira eno samo metodo: writeMessage( ):

```
import java.rmi.* ;

public interface  MessageWriter  extends Remote {

      void  writeMessage(String s)  throws RemoteException ;
}
```

Vmesnik **java.rmi.Remote** sam po sebi ne *deklarira metod ali polj. Ko pa ga izpeljemo, pove sistemu RMI, da naj vmesnik obravnavamo kot oddaljeni vmesnik.*

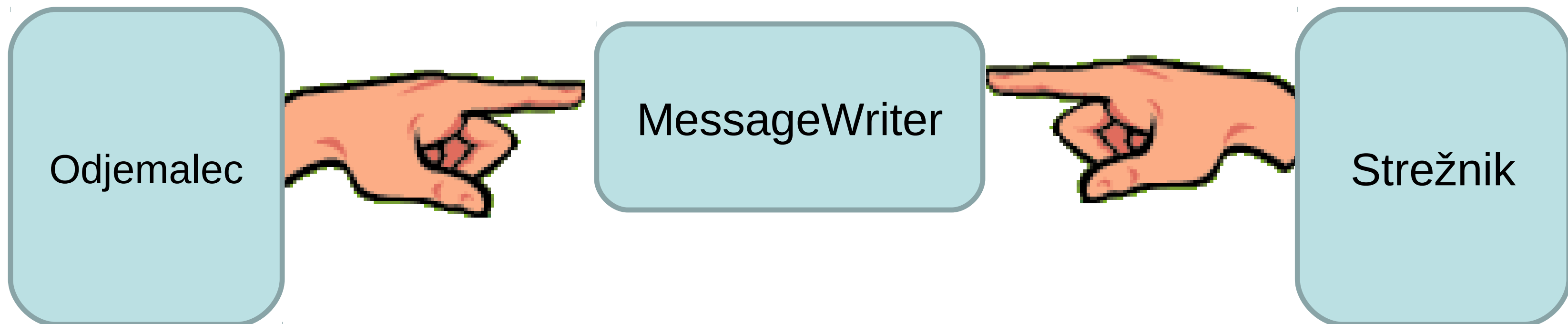
Oddaljeni objekt (Remote Object)

- *Oddaljeni objekt je instanca razreda, ki implementira oddaljeni vmesnik (remote interface).*
- *Po tihem dogovoru damo temu razredu isto ime, kot je ime oddaljenega vmesnika, dodamo pa mu na koncu "**Impl**".*

```
import java.rmi.* ;
import java.rmi.server.* ;

public class MessageWriterImpl extends UnicastRemoteObject
                                        implements MessageWriter {

    public MessageWriterImpl() throws RemoteException {
    }

    public void writeMessage(String s) throws RemoteException {
        System.out.println(s) ;
    }
}
```

Odjemalčev in strežni program

- Napisali smo datoteki za razred oddaljenega objekta, napisati moramo še program odjemalca in program strežnika, ki oba uporabljata ta razred.
- Običajno moramo opraviti še nekaj administracije z objavljanjem razredov in namestitvijo varnostnih upravnikov (security managers).
- Zaradi enostavnosti predpostavimo, da imata tako strežnik kot odjemalec kopije vseh datotek za razred **MessageWriter.**



Program - strežnik

- Imejmo datoteko **HelloServer.java** z deklaracijo naslednjega razreda:

```
import java.rmi.* ;

public class HelloServer {
    public static void main(String [ ] args) throws Exception {
        MessageWriter server = new MessageWriterImpl() ;

        Naming.rebind(“messageservice”, server) ;
    }
}
```

Komentar strežnega programa

- Ta program naredi sledeče:
  - Tvori oddaljeni objekt z lokalnim imenom **server**.
  - Objavi referenco na ta objekt z zunanjim imenom **“MessageWriter”**.
- Klic **Naming.rebind()** namesti referenco na **server** v register *RMI (RMI registry)* , ki teče na računalniku, na katerem teče tudi strežni program (**HelloServer**).
- Programi – odjemalci lahko dobijo to referenco tako, da pogledajo v ta register.

Program odjemalca

Predpostavimo, da ima datoteka **HelloClient.java** naslednjo deklaracijo razreda:

```
import java.rmi.* ;

public class HelloClient {
    public static void main(String [ ] args) throws Exception {
        MessageWriter server =
                (MessageWriter) Naming.lookup(
                        “rmi://kocka.fri.uni-
  lj.si/messageservice”) ;

        server.writeMessage(“Pozdravljen drugi svet!”) ;
    }
}
```

Komentar odjemalca

- Program naredi naslednje:
  - Poišče naslov oddaljenega objekta z zunanjim imenom **“MessageWriter”** in shrani vrnjeni naslov pod lokalnim imenom **server**.
  - Nato kliče oddaljeno metodo **writeMessage()**, ki se nahaja na lokaciji **server**.
- Klic Naming.lookup() išče v oddaljenem registru (remote registry) RMI. Njegov argument je *URL z oznako protokola* “rmi”.
- Ta primer predvideva, da živi oddaljeni objekt na računalniku “**kocka**” in da je bil registriran v privzetem RMI registru na tem računalniku.

Prevajanje primera

- programa **HelloServer** in**HelloClient** prevedemo na njunih računalnikih, na primer:

  kocka$ javac HelloServer

  sasa$ javac HelloClient

- Vse datoteke z imeni oblike **MessageWriter * .class** prepišemo v trenutni direktorij odjemalca.

Izvajanje strežnika in odjemalca

Terminal

Window Edit Options Help

```
sasa%java HelloClient
sasa%java HelloClient
sasa%
```

Terminal

Window Edit Options Help

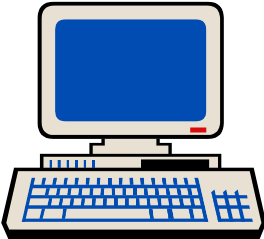
```
kocka% rmi registy &
[1] 4463
kocka% java HelloServer
Pozdravljen drugi svet!
Pozdravljen drugi svet!
```

Nekaj važnih delov RMI

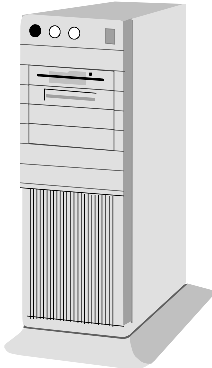
- **Štrclji (Stubs).**
  - Vsakemu oddaljenemu objektu je prirejen razred "stub", ki implementira isti oddaljeni vmesnik. Vsak odjemalec potrebuje svojo instanco "štrclja". Oddaljeni klici na strani odjemalca so v resnici lokalni klici štrclja.
- **Serializacija.**
  - Argumente in rezultate, ki naj bi bili poslani po mreži, mora sistem "serializirati" (marshalling). Po prejemu pa jih spet pretvori nazaj (unmarshalling). Vse to je za uporabnika nevidnoin se s tem ne ukvarja.
- **“Run-time System” na strani strežnika**
  - Ta je zadolžen za poslušanje zahtevkov na primernih IP vratih ter posredovanje teh zahtevkov objektu na (strežnem) računalniku.

Arhitektura

Odjemalec

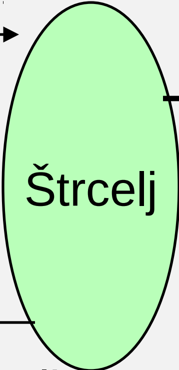


Internet



Strežnik

Lokalen klic metode štrclja

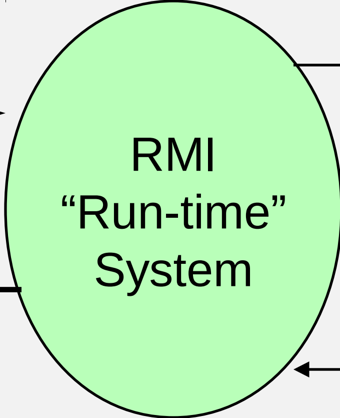


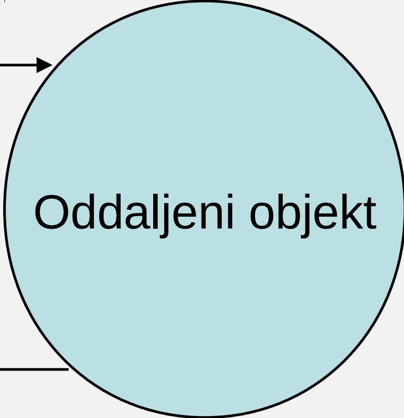
Vrne rezultat ali izjemo

Pošlje serializirane argumente

Pošlje serializirane rezultate ali izjeme

lokalno kliče metode objekta





Vrne vrednost ali izjemo