Modulo IIb
Java Networking Programming

Prof. Ismael H F Santos

Ementa

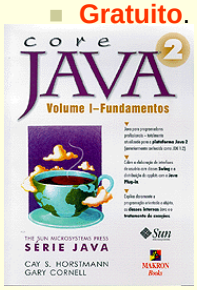
- Modulo IIb – Networking em JAVA
  - Networking

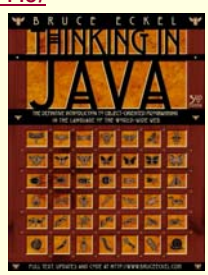
Bibliografia

- *Linguagem de Programação JAVA*
  - *Ismael H. F. Santos, Apostila UniverCidade, 2002*
- *The Java Tutorial: A practical guide for programmers*
  - *Tutorial on-line: http://java.sun.com/docs/books/tutorial*
- *Java in a Nutshell*
  - *David Flanagan, O´Reilly & Associates*
- *Just Java 2*
  - *Mark C. Chan, Steven W. Griffith e Anthony F. Iasi, Makron Books.*
- *Java 1.2*
  - *Laura Lemay & Rogers Cadenhead, Editora Campos*

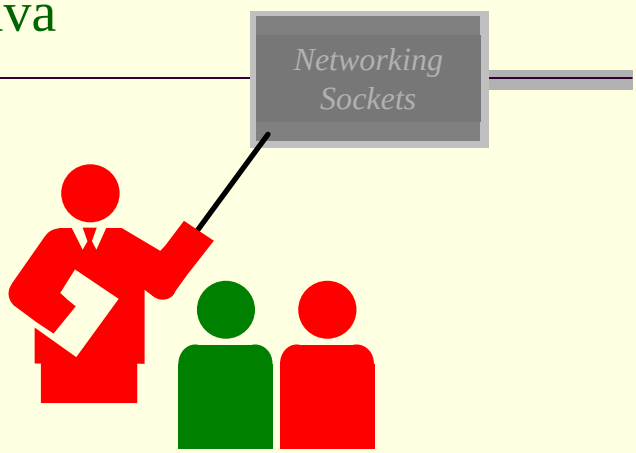
Livros

- **Core Java 2**, Cay S. Horstmann, Gary Cornell
  - Volume 1 (Fundamentos)
  - Volume 2 (Características Avançadas)
- **Java: Como Programar**, Deitel & Deitel
- **Thinking in Patterns with JAVA**, Bruce Eckel
  - **Gratuito**. http://www.mindview.net/Books/TIJ/





POO-Java



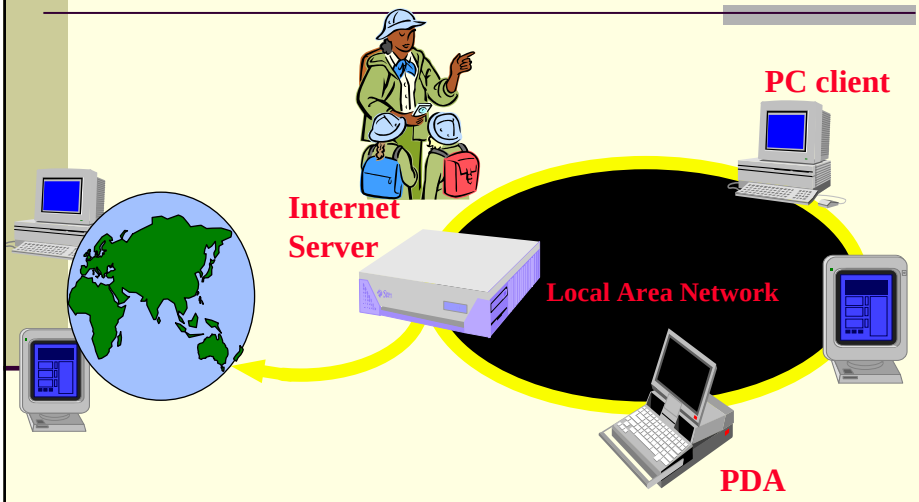
Network Summary

- Internet
  - Designed with multiple layers of abstraction
  - Underlying medium is unreliable, packet oriented
  - Provides two views
    - Reliable, connection oriented (TCP)
    - Unreliable, packet oriented (UDP)
- Java
  - Object-oriented classes & API
    - Sockets, URLs
    - Extensive networking support

Introduction

- Internet and WWW have emerged as global ubiquitous media for communication and changing the way we conduct science, engineering, and commerce.
- They also changing the way we learn, live, enjoy, communicate, interact, engage, etc. It appears like the modern life activities are getting completely centered around the Internet.

Internet Applications Serving Local and Remote Users



Internet & Web as a delivery Vehicle

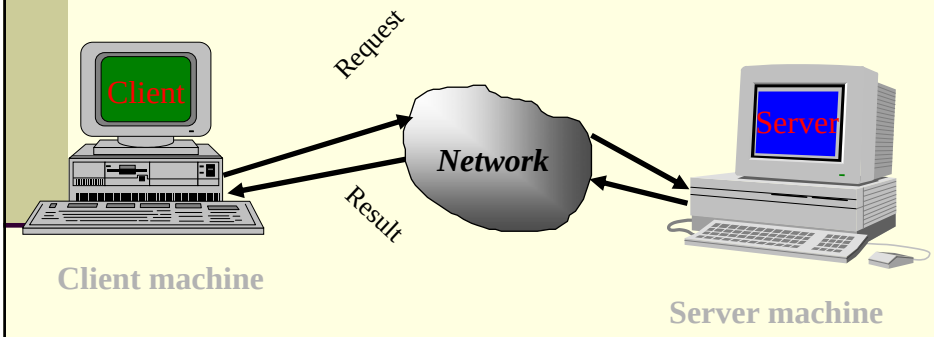
book reviews	captions to cartoons	fairy tales	flora/fauna report
food reviews	greeting cards or post cards	grocery lists	how-to pages
interviews	job descriptions	jokes	local menus
local legends / myths	local remedies	local folklore	movie critiques
newpapers	news analyses	problem solving	protest signs
puzzles	questionnaires	quotations	real estate notices
recipes	sayings	schedules	serialized stories
song lyrics	sports page	superstitions	traffic rules
TV reviews	used car descriptions	want ads	wanted posters

Increased demand for Internet applications

- To take advantage of opportunities presented by the Internet, businesses are continuously seeking new and innovative ways and means for offering their services via the Internet.
- This created a huge demand for software designers with skills to create new Internet-enabled applications or migrate existing/legacy applications on the Internet platform.
- Object-oriented Java technologies—Sockets, threads, RMI, clustering, Web services-- have emerged as leading solutions for creating portable, efficient, and maintainable large and complex Internet applications.

Elements of C-S Computing

a client, a server, and network



Client machine

Server machine

Networking Basics

- Applications Layer
  - Standard apps
    - HTTP
    - FTP
    - Telnet
  - User apps
- Transport Layer
  - TCP
  - UDP
  - Programming Interface:
    - Sockets
- Network Layer
  - IP
- Link Layer
  - Device drivers

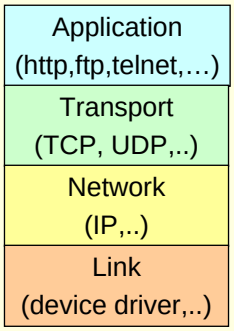
- TCP/IP Stack

Application (http,ftp,telnet,…)
Transport (TCP, UDP,..)
Network (IP,..)
Link (device driver,..)

Networking Basics

- TCP (Transport Control Protocol) is a connection-oriented protocol that provides a reliable flow of data between two computers.
- Example applications:
  - HTTP
  - FTP
  - Telnet

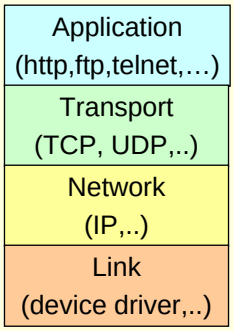
- TCP/IP Stack



Networking Basics

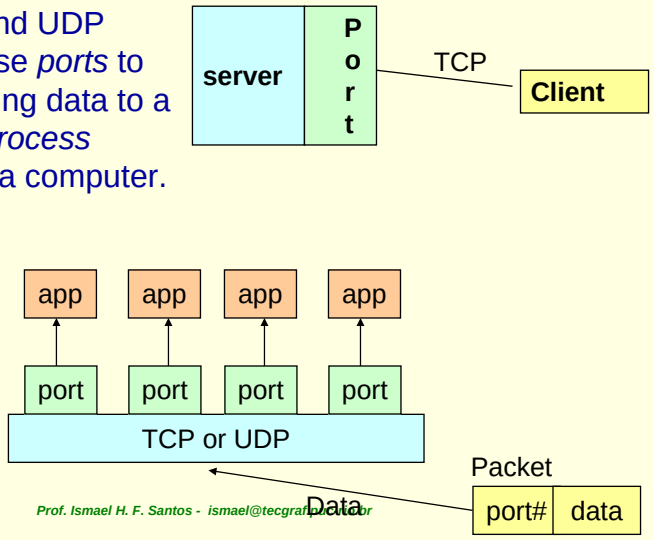
- UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called *datagrams*, from one computer to another with no guarantees about arrival.
- Example applications:
  - Clock server
  - Ping

- TCP/IP Stack



Understanding Ports

- The TCP and UDP protocols use *ports* to map incoming data to a particular *process* running on a computer.

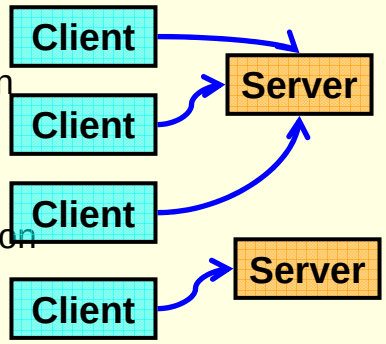


Understanding Ports

- Port is represented by a positive (16-bit) integer value
- Some ports have been reserved to support common/well known services:
  - ftp 21/tcp
  - telnet 23/tcp
  - smtp 25/tcp
  - login 513/tcp
- User level process/services generally use port number value >= 1024

Client / Server Model

- Relationship between two computer programs
- Client
  - Initiates communication
  - Requests services
- Server
  - Receives communication
  - Provides services
- Other models
  - Master / worker
  - Peer-to-peer (P2P)



Conceitos

- **Cliente:** Processo que solicita algo.
- **Servidor:** Processo que disponibiliza recursos.
- **Host:** Computador onde é executado o processo servidor.
- **Porta:** Porta do computador na qual o serviço é disponibilizado.
- **Socket:** Canal de comunicação entre um processo cliente e um processo servidor.
- **Conexão:** Identifica o par de sockets entre o processo cliente e o processo servidor.
- **Protocolo:** Padrão de comunicação entre os processos cliente e servidor.

Client Programming

- Basic steps
  1. Determine server location – IP address & port
  2. Open network connection to server
  3. Write data to server (request)
  4. Read data from server (response)
  5. Close network connection
  6. Stop client

Características

- **Cliente:** Precisa saber da existência e conhecer o endereço do servidor.
- **Servidor:** Não precisa saber o endereço do cliente, ou da sua existência, antes da conexão.

- **Sockets:** Cada um dos processos (Cliente e Servidor) constrói seu socket. Utiliza uma porta de comunicação.

Server Programming

- Basic steps
  1. Determine server location - port (& IP address)
  2. Create server to listen for connections
  3. Open network connection to client
  4. Read data from client (request)
  5. Write data to client (response)
  6. Close network connection to client
  7. Stop server

Server Programming

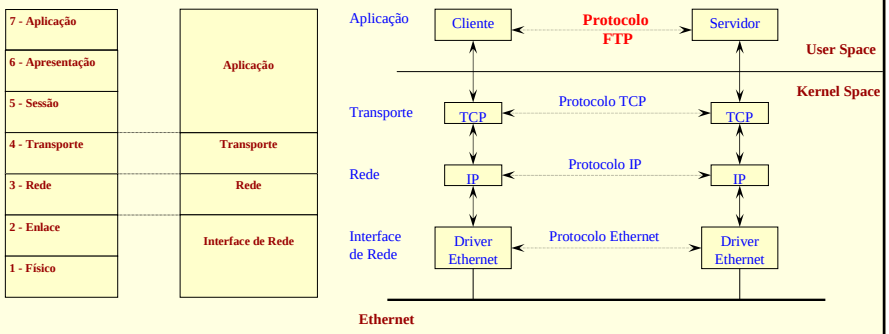
- Can support multiple connections / clients
- Loop
  - Handles multiple connections in order
- Multithreading
  - Allows multiple simultaneous connections

Client / Server Model Examples

Application	Client	Server
Web Browsing	Internet Explorer, Mozilla Firefox	Apache
Email	MS Outlook, Thunderbird	POP, IMAP, SMTP, Exchange
Streaming Music	Windows Media Player, iTunes	Internet Radio
Online Gaming	Half-Life, Everquest, PartyPoker	Game / Realm Servers

TCP/IP

- Pilha de Protocolos
  - Os serviços da pilha de protocolos TCP/IP podem ser arranjados de acordo com o esquema de camadas proposto pelo modelo RM/OSI da ISO, conforme figura abaixo.



Networking in Java

- Packages
  - java.net ⇒ Networking
  - java.io ⇒ I/O streams & utilities
  - java.rmi ⇒ Remote Method Invocation
  - java.security ⇒ Security policies
  - java.lang ⇒ Threading classes
- Support at multiple levels
  - Data transport ⇒ Socket classes
  - Network services ⇒ URL classes
  - Utilities & security

Sockets

- Dois processos para se comunicar precisam declarar o mesmo tipo de Socket e o mesmo domínio.
- Quando um socket é criado, o programa tem que especificar o tipo do socket e o domínio do endereço.
- Um socket pode ser de dois tipos: de fluxo ou de mensagens.
- O socket de fluxo trata as comunicações como um fluxo contínuo de caracteres.
  - O socket de mensagens têm que ler mensagens inteiras.

Domínio

Os domínios mais usados são o do UNIX e o da INTERNET

- DOMÍNIO DO UNIX – dois processos compartilham o mesmo sistema de arquivos. O endereço é uma string de caracteres que é uma entrada no sistema de arquivos.
- DOMÍNIO DA INTERNET – dois processos executando em hosts diferentes se comunicam. O endereço é o IP do host.

Protocolos de Comunicação

- Cada tipo de socket utiliza um protocolo de comunicação.

- O socket de fluxo utiliza o TCP (Transmission Control Protocol) é um protocolo orientado a conexão e seguro. Como exemplo pode ser citada uma ligação via telefone.
- O socket de mensagens utiliza o UDP (UNIX Datagram Protocol) que não é baseado em conexão e não é seguro. Como exemplo pode ser citado o envio de uma carta.

Usos

- Uma aplicação cliente que coleta informações de um usuário antes de enviá-las a um servidor central.
- Uma aplicação de servidor que funciona como um ponto de coleta central para dados de vários usuários.
- Chat.

Usos

- Uma aplicação cliente que coleta informações de um usuário antes de enviá-las a um servidor central.
- Uma aplicação de servidor que funciona como um ponto de coleta central para dados de vários usuários.
- Chat.

Networking in Java

- Packages
  - java.net ⇒ Networking
  - java.io ⇒ I/O streams & utilities
  - java.rmi ⇒ Remote Method Invocation
  - java.security ⇒ Security policies
  - java.lang ⇒ Threading classes
- Support at multiple levels
  - Data transport ⇒ Socket classes
  - Network services ⇒ URL classes
  - Utilities & security

Java Networking API

- Application Program Interface
  - Set of routines, protocols, tools
  - For building software applications
- Java networking API
  - Helps build network applications
  - Interfaces to sockets, network resources
  - Code implementing useful functionality
  - Includes classes for
    - Sockets
    - URLs

Java Networking Classes

- IP addresses
  - InetAddress
- Packets
  - DatagramPacket
- Sockets
  - Socket
  - ServerSocket
  - DatagramSocket
- URLs
  - URL

Java Sockets Programming

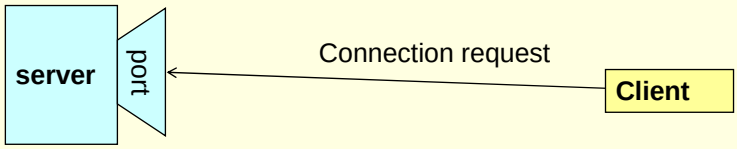
- Java uses BSD-style sockets to interface with TCP/IP services (`java.net` package)
- Java distinguishes between UDP, TCP server & TCP client sockets
- Behind-the-scenes classes do the actual work & can be updated or swapped out transparently

Sockets

- Sockets provide an interface for programming networks at the transport layer.
- Network communication using Sockets is very much similar to performing file I/O
  - In fact, socket handle is treated like file handle.
  - The streams used in file I/O operation are also applicable to socket-based I/O
- Socket-based communication is programming language independent.
  - That means, a socket program written in Java language can also communicate to a program written in Java or non-Java socket program.

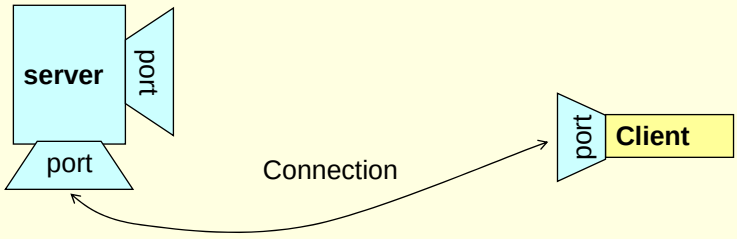
Socket Communication

- A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server waits and listens to the socket for a client to make a connection request.



Socket Communication

- If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bounds to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.



IP Addresses & Hostnames

- **`java.net.InetAddress`** class
- Represents a single IP address
- Factory class – no public constructor
- Performs transparent DNS lookups or reverse lookups
- **`java.net.UnkownHostException`** thrown if DNS system can't find IP address for specific host

java.net

O pacote java.net provê uma estrutura poderosa e flexível para rede. Muitas das classes deste pacote fazem parte de uma infra-estrutura de rede não utilizadas em aplicações normais, são mais complicadas e de difícil compreensão. Aqui vamos descrever somente as classes utilizadas normalmente em aplicações.

A classe URL representa uma localização na Internet. Ela provê uma interface bem simples de rede – o download de um objeto referenciado por uma URL pode ser feito através de uma simples chamada, ou podem ser abertos canais de leitura ou escrita neste objeto.

java.net

Em um nível mais complexo, o objeto URLConnection pode ser obtido a partir de um objeto URL fornecido. A classe URLConnection provê métodos adicionais que permitem trabalhar com URLs de maneiras mais sofisticadas.

Para fazer mais do que um simples download de um objeto referenciado por uma URL, o pacote java.net fornece a classe Socket. Esta classe permite conectar a uma determinada porta em um determinado host Internet e ler e escrever dados utilizando as classes InputStream e OutputStream do pacote java.io. Para implementar um servidor para aceitar conexões de clientes, é possível utilizar a classe ServerSocket. Tanto a classe Socket quanto a ServerSocket utilizam a classe InetAddress, que representa um endereço Internet.

java.net (cont.)

O pacote java.net fornece ainda classes que permitem operações de rede de baixo nível, através dos objetos DatagramPacket, que podem ser enviados e recebidos pela rede através do objeto DatagramSocket. O Java 1.1 incluiu neste pacote a classe MulticastSocket que suporta rede por difusão (multicast networking).

Uniform Resource Locator

- A classe **URL** modela URLs, permitindo a obtenção de informações e conteúdo de páginas na Web
- Essa classe é parte do pacote **java.net**

URL Class

- Provides high-level access to network data
- Abstracts the notion of a connection
- Constructor opens network connection
  - To resource named by URL

URL Constructors

- URL(fullURL)
  - URL("http://www.cs.umd.edu/class/index.html")
- URL(baseURL, relativeURL)
  - URL base = new URL("http://www.cs.umd.edu/");
  - URL class = new URL(base, "/class/index.html ");
- URL(protocol, baseURL, relativeURL)
  - URL("http", www.cs.umd.edu, "/class/index.html")
- URL(protocol, baseURL, port, relativeURL)
  - URL("http", www.cs.umd.edu, 80,"/class/index.html")

URL: Construtores

```
public URL(String spec)
   throws MalformedURLException

public URL(String protocol, String host,
           String file)
   throws MalformedURLException

public URL(String protocol, String host,
           int port, String file)
   throws MalformedURLException
```


URL Methods

- getProtocol( )
- getHost( )
- getPort( )
- getFile( )
- getContent( )
- openStream()
- openConnection()

URL: Métodos de Consulta

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

public String getUserInfo()
public String getPath()
public String getQuery()
```


URL Connection Classes

- High level description of network service
- Access resource named by URL
- Can define own protocols
- Examples
  - URLConnection ⇒ Reads resource
  - HttpURLConnection ⇒ Handles web page
  - JarURLConnection ⇒ Manipulates Java Archives
  - URLClassLoader ⇒ Loads class file into JVM

java.net (cont.)

java.net.URL

Esta classe representa uma URL (Uniform Resource Locator) e permite o download de dados referidos por uma URL.

java.net.URLConnection

Classe abstrata, define uma conexão de rede para um objeto especificado por uma *URL*. Deve ser utilizada quando se deseja mais controle sobre o download dos dados do que os oferecidos pelos métodos da classe *URL*.

java.net (cont.)

java.net.HttpURLConnection

Esta classe é uma especialização da classe *URLConnection*. Uma instância desta classe é retornada quando o método *openConnection()* é chamado por um objeto *URL* que utiliza o protocolo HTTP. As muitas constantes definidas por esta classe são os códigos de estado retornados pelos servidores HTTP. Por isto, para um maior entendimento desta classe, é necessária a compreensão em detalhes do protocolo HTTP.

java.net (cont.)

java.net.URLEncoder

Classe que define um método estático utilizado para converter uma string para a forma *URLencoded*, ou seja, espaços são convertidos para "+", e todos os caracteres não alfanuméricos (exceto "_") são convertidos por um caracter de percentagem ("%") seguindo por 2 dígitos hexadecimais. Este método é utilizado para permitir o envio da URL em ASCII, de forma que os caracteres especiais sejam corretamente interpretados por qualquer computador no mundo.

URL: Métodos de Acesso

```
public final InputStream openStream()
   throws IOException

public URLConnection openConnection()
   throws IOException
```

Exemplo de Uso de **URL**

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

public class HttpClient_URL {
  public static void main(String[] args) throws Exception {
    if (args.length == 0) {
      System.err.println("Forneça o endereço da página.");
      return;
    }
    URL url = new URL(args[0]);
    InputStream is = url.openStream();
    Reader r = new InputStreamReader(is);
    BufferedReader br = new BufferedReader(r);
    String l;
    while ((l = br.readLine()) != null) {
      System.out.println(l);
    }
  }
}
```


java.net (cont.)

java.net.InetAddress

Esta classe representa um endereço internet, e é utilizada na criação dos objetos *DatagramPacket* e *Socket*.

java.net.Socket

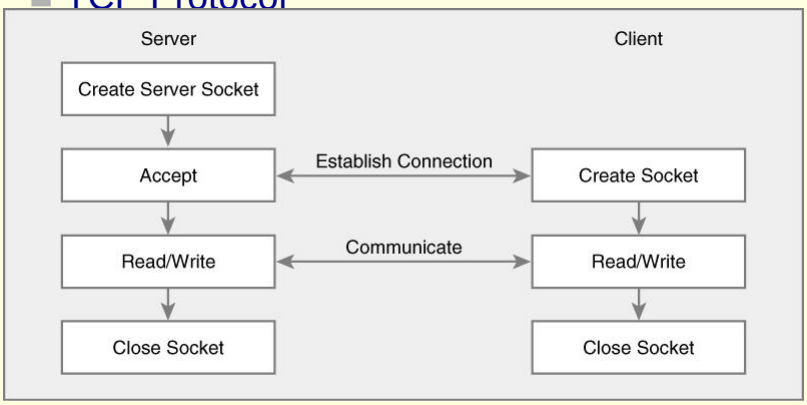
Esta classe implementa um socket para a comunicação entre processos através da rede. O método construtor cria um socket e o conecta ao host e porta especificados. Opcionalmente, também é possível especificar se a comunicação através do socket deve ser baseada em uma conexão confiável ou não. A conexão confiável é a padrão.

InetAddress Class

- Represents an IP address
- Can convert domain name to IP address
  - Performs DNS lookup
- Getting an InetAddress object
  - getLocalHost()
  - getByName(String host)
  - getByAddress(byte[] addr)

Connection Oriented

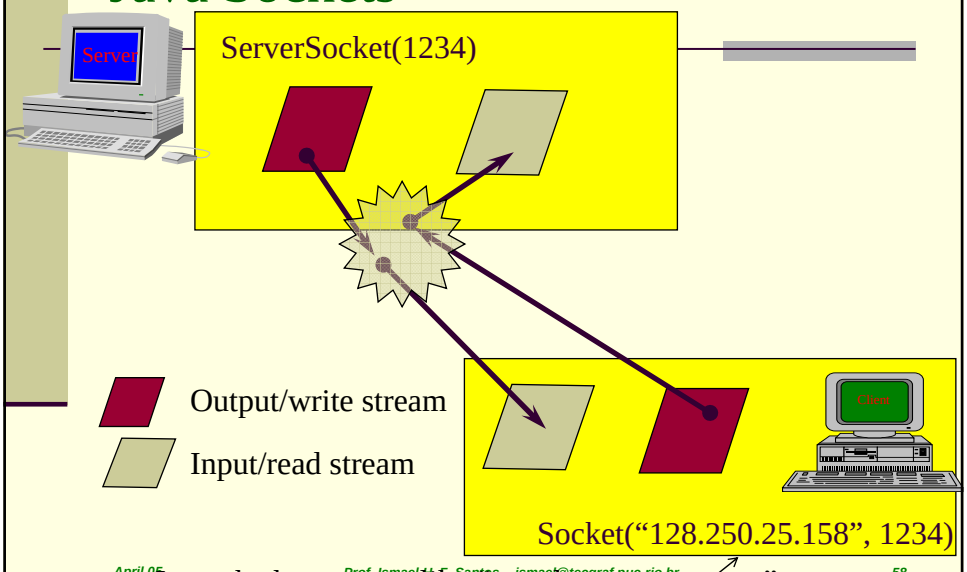
- TCP Protocol



Sockets and Java Socket Classes

- A socket is an endpoint of a two-way communication link between two programs running on the network.
- A socket is bound to a port number so that the TCP layer can identify the application that data destined to be sent.
- Java's .net package provides two classes:
  - Socket – for implementing a client
  - ServerSocket – for implementing a server

Java Sockets



It can be host_name like "mandroo.cs.mu.oz.au"

TCP Client Sockets

- `java.net.Socket` class
- Combines socket with socket options (timeout, linger, keep alive, no delay, etc)
- Encapsulates a `java.io.InputStream` and a `java.io.OutputStream` – can be retrieved for use in a layered I/O system

Socket Classes

- Provides interface to TCP, UDP sockets
- Socket
  - TCP client sockets
- ServerSocket
  - TCP server sockets
- DatagramSocket
  - UDP sockets (server or client)

Socket Class

- Creates socket for client
- Constructor connects to
  - Machine name or IP address
  - Port number
- Transfer data via streams
  - Similar to standard Java I/O streams

Socket Methods

- getInputStream()
- getOutputStream()
- close()
- getInetAddress()
- getPort()
- getLocalPort()

Implementing a Client

1. Create a Socket Object:
   ```
   client = new Socket( server, port_id );
   ```
2. Create I/O streams for communicating with the server.
   ```
   is = new DataInputStream(client.getInputStream() );
   os = new DataOutputStream( client.getOutputStream() );
   ```
3. Perform I/O or communication with the server:
   - Receive data from the server:
     ```
     String line = is.readLine();
     ```
   - Send data to the server:
     ```
     os.writeBytes("Hello\n");
     ```
4. Close the socket when done:
   ```
   client.close();
   ```


A simple client (simplified code)

```
// SimpleClient.java: a simple client program
import java.net.*;
import java.io.*;
public class SimpleClient {
  public static void main(String args[]) throws IOException {
    // Open your connection to a server, at port 1234
    Socket s1 = new Socket("mundroo.cs.mu.oz.au",1234);
    // Get an input file handle from the socket and read the input
    InputStream s1In = s1.getInputStream();
    DataInputStream dis = new DataInputStream(s1In);
    String st = new String (dis.readUTF());
    System.out.println(st);
    // When done, just close the connection and exit
    dis.close();
    s1In.close();
    s1.close();
  }
}
```

java.net (cont.)

java.net.ServerSocket

Esta classe é utilizada por servidores para "escutar" (listen) requisições de conexões de clientes. Quando um objeto *ServerSocket* é criado, é especificada a porta que o servidor irá escutar. O método *accept()* inicia a escuta nesta porta, e fica aguardando até que um cliente requisite uma conexão nesta porta. Quando isto ocorre, o método *accept()* aceita a conexão, criando e retornando um *Socket* que o servidor pode utilizar para se comunicar com o cliente.

TCP Server Sockets

- **`java.net.ServerSocket`** class
- Binds to a local port to listen for initial connections
- Can be bound to a local IP for multi-homed machines
- **`accept()`** method returns a **`java.net.Socket`**, not an integer descriptor

ServerSocket Class

- Create socket on server
- Constructor specifies local port
  - Server listens to port
- Usage
  - Begin waiting after invoking accept()
  - Listen for connection (from client socket)
  - Returns Socket for connection

ServerSocket Methods

- accept()
- close()
- getInetAddress()
- getLocalPort()

Implementing a Server

1. Open the Server Socket:
```
ServerSocket server;
DataOutputStream os;
DataInputStream is;
server = new ServerSocket( PORT );
```
2. Wait for the Client Request:
```
Socket client = server.accept();
```
3. Create I/O streams for communicating to the client
```
is = new DataInputStream( client.getInputStream() );
os = new DataOutputStream( client.getOutputStream() );
```
4. Perform communication with client
```
Receive from client: String line = is.readLine();
Send to client: os.writeBytes("Hello\n");
```
5. Close sockets: `client.close();`

For multithreaded server:

```
while(true) {
```
i. wait for client requests (step 2 above)

ii. create a thread with "client" socket as parameter (the thread creates streams (as in step (3) and does communication as stated in (4). Remove thread once service is provided.

```
}
```


A simple server (simplified code)

```
// SimpleServer.java: a simple server program
import java.net.*;
import java.io.*;
public class SimpleServer {
  public static void main(String args[]) throws IOException {
    // Register service on port 1234
    ServerSocket s = new ServerSocket(1234);
    Socket s1=s.accept(); // Wait and accept a connection
    // Get a communication stream associated with the socket
    OutputStream s1out = s1.getOutputStream();
    DataOutputStream dos = new DataOutputStream (s1out);
    // Send a string!
    dos.writeUTF("Hi there");
    // Close the connection, but not the server socket
    dos.close();
    s1out.close();
    s1.close();
  }
}
```

Run

- Run Server on mundroo.cs.mu.oz.au
  - [raj@mundroo] java SimpleServer &
- Run Client on any machine (including mundroo):
  - [raj@mundroo] java SimpleClient
    Hi there
- If you run client when server is not up:
  - [raj@mundroo] sockets [1:147] java SimpleClient

```
Exception in thread "main" java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:320)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:133)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:120)
        at java.net.Socket.<init>(Socket.java:273)
        at java.net.Socket.<init>(Socket.java:100)
        at SimpleClient.main(SimpleClient.java:6)
```


Socket Exceptions

```
try {
    Socket client = new Socket(host, port);
    handleConnection(client);
}
catch(UnknownHostException uhe) { System.out.println("Unknown
    host: " + host); uhe.printStackTrace();
}
catch(IOException ioe) {
System.out.println("IOException: " + ioe); ioe.printStackTrace();
}
```

ServerSocket & Exceptions

- public **ServerSocket**(int port) throws IOException
  - Creates a server socket on a specified port.
  - A port of 0 creates a socket on any free port. You can use **getLocalPort**() to identify the (assigned) port on which this socket is listening.
  - The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused.
- Throws:
  - IOException - if an I/O error occurs when opening the socket.
  - SecurityException - if a security manager exists and its checkListen method doesn't allow the operation.

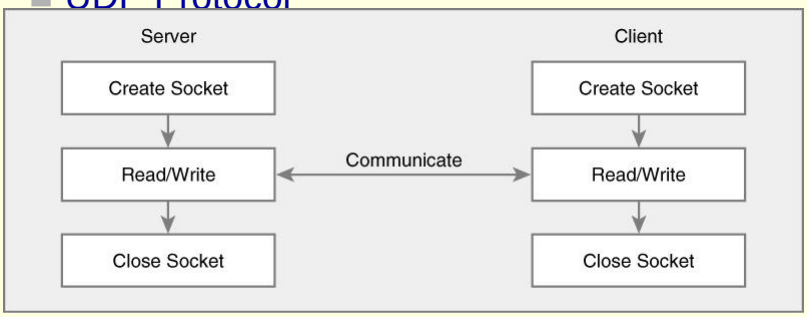
java.net (cont.)

java.net.DatagramSocket

Esta classe define um socket que pode receber e enviar pacotes de datagramas pela rede utilizando o protocolo UDP. Um datagrama é uma interface de rede de nível bem baixo: é simplesmente um vetor de bytes enviado através da rede. Um datagrama não implementa nenhum tipo de protocolo de comunicação baseado em stream, e não há conexão estabelecida entre o remetente e o receptor. Pacotes de datagramas são chamados não confiáveis porque o protocolo não faz nenhum esforço para garantir a chegada ou o reenvio do datagrama em caso de falha no envio. Assim, pacotes enviados através do *DatagramSocket* não tem garantia de chegada, muito menos garantia de chegada na ordem enviada. Por outro lado, este protocolo simples faz com que as transmissões de datagramas sejam bem rápidas.

Packet Oriented

- UDP Protocol



UDP Sockets

- `java.net.DatagramSocket` class
- Java makes no distinction between client/server for UDP sockets
- Connected mode UDP supported in Java 2
- Can be bound to both a local port & a local IP address – multi-homed support
- Supports some socket options (timeout, buffer size)

DatagramSocket Class

- Create UDP socket
  - Does not distinguish server / client sockets
- Constructor specifies InetAddress, port
- Set up UPD socket connection
- Send / receive DatagramPacket

DatagramSocket Methods

- close()
- getLocalAddress()
- getLocalPort()
- receive(DatagramPacket p)
- send(DatagramPacket p)
- setSoTimeout(int t)
- getSoTimeout()

java.net (cont.)

java.net.DatagramPacket

Esta classe implementa um pacote de dados que pode ser enviado ou recebido pela rede através de um *DatagramSocket*.

UDP Datagrams

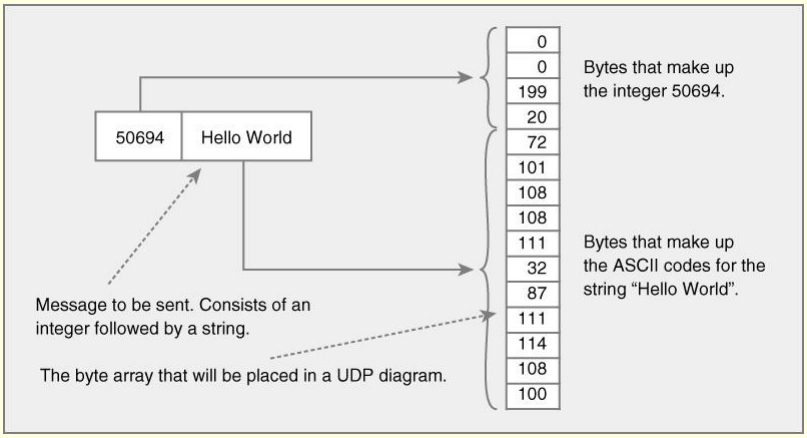
- **`java.net.DatagramPacket`** class
- Expects a byte array of data
- Address optional for connected-mode UDP
- This class is final – can't be extended!
- **`java.net.DatagramSocket`** instances can only send instances of **`java.net.DatagramPacket`**

DatagramPacket Class

- Each packet contains
  - InetAddress
  - Port of destination
  - Data

DatagramPacket Class

- Data in packet represented as byte array



DatagramPacket Methods

- getAddress()
- getData()
- getLength()
- getPort()
- setAddress()
- setData()
- setLength()
- setPort()

Exemplo de Programa

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

/* Este programa conecta a um WEB server e faz downloads de uma específica URL. Usa o
   protocolo HTTP. */

public class HttpClient {
    public static void main(String[] args) {
      try {
          // Verifica os argumentos
            if ((args.length != 1 ) && (args.length != 2))
                throw new IllegalArgumentException("Faltam argumentos");
          // Pega uma string para escrever o conteúdo da URL
            OutputStream to_file;
             if (args.length == 2) to_file = new FileOutputStream( args[1] );
                else to_file = System.out;
          // Agora usa uma class URL para analisar o que foi passado pelo
          // usuário como parâmetro: protocolo, host, porta, filename
```

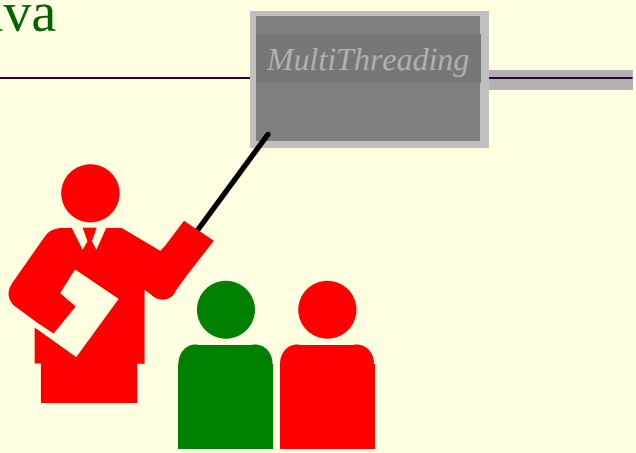
Exemplo de Programa (cont.)

```
        URL url = new URL( args[0] );
        String protocol = url.getProtocol();
        if (!protocol.equals("http"))
           throw new IllegalArgumentException("Usar o protocolo http");
        String host = url.getHost();
        int port = url.getPort();
        if (port == -1) port = 80; // Se não há porta,
                                   // use o default do http
        String filename = url.getFile();

      // Abrir uma conexão na rede com socket para a porta e o host
      // especificados
        Socket socket = new Socket(host,port);

      // Pega entrada e saída para o socket
        InputStream from_server = socket.getInputStream();
        PrintWriter to_server =  new PrintWriter
                ( new OutputStreamWriter(socket.getOutputStream() ));
```


Exemplo de Programa (cont.)

```
      // Envia o comando GET HTTP para o WEB server, especificando o
      // arquivo
          to_server.println("GET " + filename);
          to_server.flush(); // Envie agora

      // Le a resposta do server, e a escreve no arquivo
          byte[] buffer = new byte[4096];
          int bytes_read;
          while(( bytes_read = from_server.read(buffer)) != -1)
                    to_file.write(buffer, 0, bytes_read);
      // Quando o server fecha a conexão, nós fechamos nosso arquivo
          socket.close();
          to_file.close();
      }
      catch ( Exception e) { // Informe dos erros ocorridos
         System.err.println(e);
         System.err.println("Ao usar: java HttpClient <URL> <filename>]");
      }
   }
}
```

POO-Java



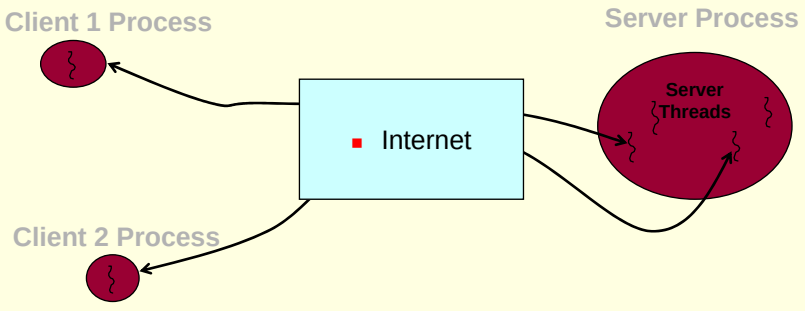
Threading

- Java doesn't support the notion of forking processes; how do we support concurrency?
  - Java was designed to support multi-threading!
  - In server environments we can spawn new threads to handle each client
  - Thread groups allow for collective control of many threads

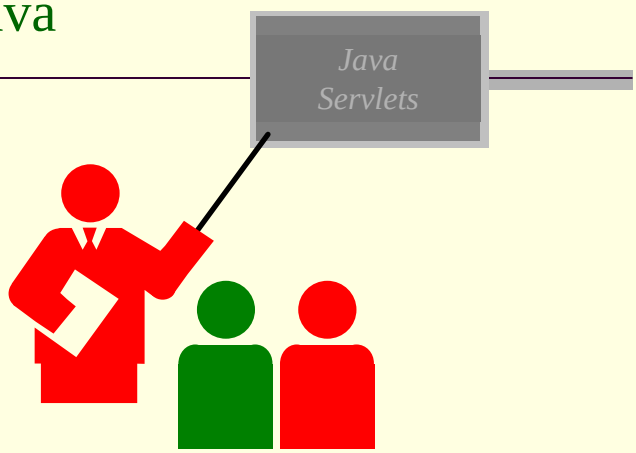
Server in Loop: Always up

```
// SimpleServerLoop.java: a simple server program that runs forever in a single thead
import java.net.*;
import java.io.*;
public class SimpleServerLoop {
  public static void main(String args[]) throws IOException {
    // Register service on port 1234
    ServerSocket s = new ServerSocket(1234);
    while(true)
    {
          Socket s1=s.accept(); // Wait and accept a connection
          // Get a communication stream associated with the socket
          OutputStream s1out = s1.getOutputStream();
          DataOutputStream dos = new DataOutputStream (s1out);
          // Send a string!
          dos.writeUTF("Hi there");
          // Close the connection, but not the server socket
          dos.close();
          s1out.close();
          s1.close();
    }
  }
}
```


Multithreaded Server: For Serving Multiple Clients Concurrently



POO-Java



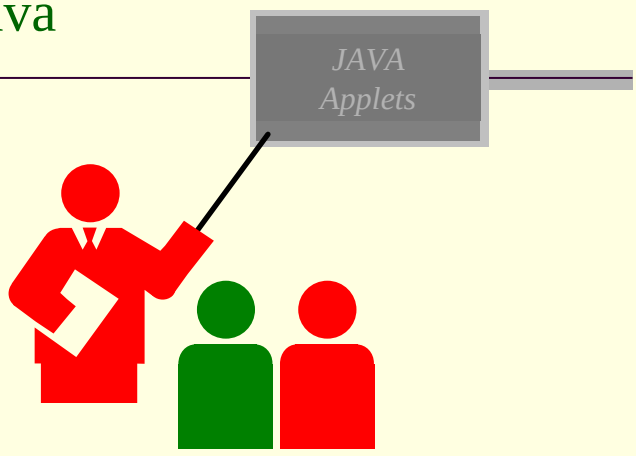
Java Servlets

- Servlets are the Java analog to CGI
- Advantages of servlets: full access to other Java APIs, persistence between invocations, guaranteed portability
- Servlets can be generic services or specific to HTTP

HTTP Servlets

- `javax.servlet.http.HttpServlet` class
- Uses HTTP to receive requests and generate responses
- Full support for all HTTP methods, cookies, sessions, persistent connections
- Servlets can be chained – example: de-blink servlet

POO-Java



Java Applets

- Applets are Java programs
  - Classes downloaded from network
  - Run in browser on client
- Applets have special security restrictions
  - Executed in applet sandbox
  - Controlled by java.lang.SecurityManager

Java Applets

- Client-side Java programs that run in a browser
- Applets have special security restrictions called the applet sandbox
- Only applets loaded over the network are subject to the applet sandbox
- The applet sandbox is controlled by a **`java.lang.SecurityManager`**

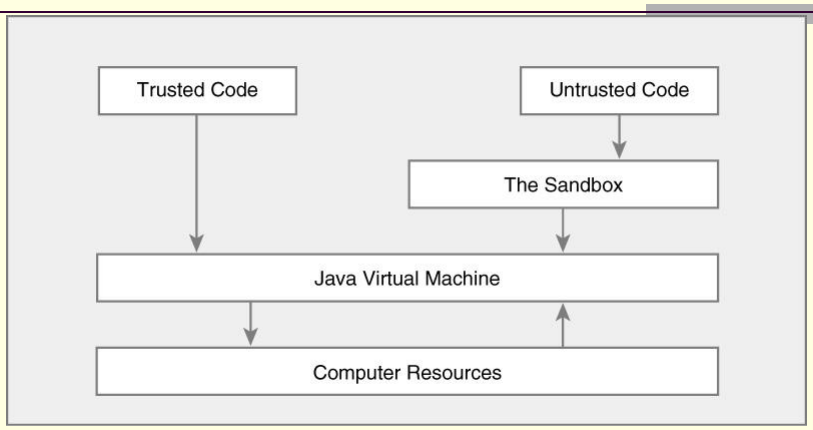
Applet Sandbox

- Prevents
  - Loading libraries
  - Defining native methods
  - Accessing local host file system
  - Running other programs (Runtime.exec())
  - Listening for connections
  - Opening sockets to new machines
    - Except for originating host
- Restricted access to system properties

Applet Sandbox

- Can't load libraries or define native methods
- Can't access local host filesystem
- Can't open sockets to hosts other than originating host
- Can't use **`Runtime.exec()`**
- Applet windows have a unique appearance
- Restricted access to certain system properties

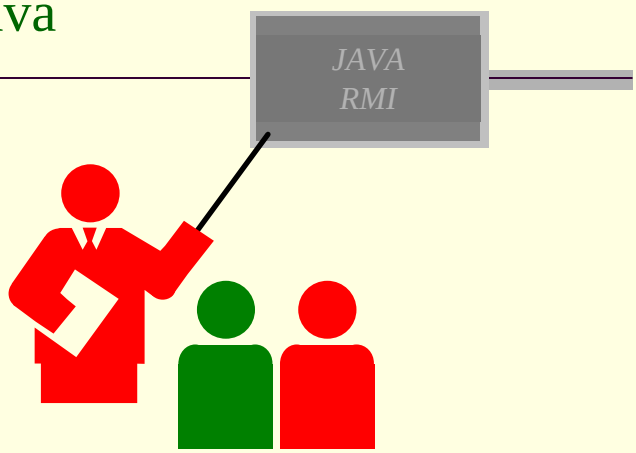
Applet Sandbox



Escaping the Applet Sandbox

- Browsers can define their own security policy via a new security manager
- Applets can be signed and executed as trusted content
- Security policies may vary from browser to browser, even for signed applets

POO-Java



Remote Method Invocation (RMI)

- RMI is the Java analog to RPC
- RMI servers use a naming service (rmiregistry) to register remote objects
- RMI servers use a special security policy implemented by **`RMISecurityManager`**
- The default RMI transport mechanism is via TCP sockets – this is transparent to RMI code!
- Any object transferred in an RMI call must implement the Serializable interface

Java Naming & Directory Interface (JNDI)

- JNDI provides a generic API that can be used to interface with any naming system
- JNDI uses SPIs (service provider interfaces) to access many different types of naming & directory services from the JNDI API
- Sun supplies JNDI SPIs for LDAP, NIS, COS (CORBA naming), RMI registry & local filesystem