INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Programmation Orientée Objet

Sami Yangui, Ph.D.
A. Prof and CNRS LAAS Researcher

Lecture 3: Socket Programming in Java

October 21, 2019

OUTLINE

- A Definition of Sockets
- Types of Sockets
- The Java.net Framework
- TCP classes
  - InetAddress class, ServerSocket class, Socket class
- UDP classes
  - DatagramSocket class, DatagramPacket class

SOCKETS - DEFINITION

- An interface between application and network
  1. The application creates a socket
  2. The socket *type* dictates the communication style
     - Reliable vs best effort
     - Connection-oriented vs connectionless
  3. The application configures the socket to:
     - Pass data (to the socket) for network transmission
     - Receive data (from the socket) transmitted through the network by some other entities

TYPE OF SOCKETS

- There are two essential types of sockets

STREAM	DATAGRAM
a.k.a TCP	a.k.a UDP
Reliable delivey	Unreliable delivery
In-order guaranteed	No order guarantees
Connection-oriented	No notion of « connection »
Bi-directional	Can send or receive

Transmission Control Protocol

With acknowledgement

User Datagram Protocol

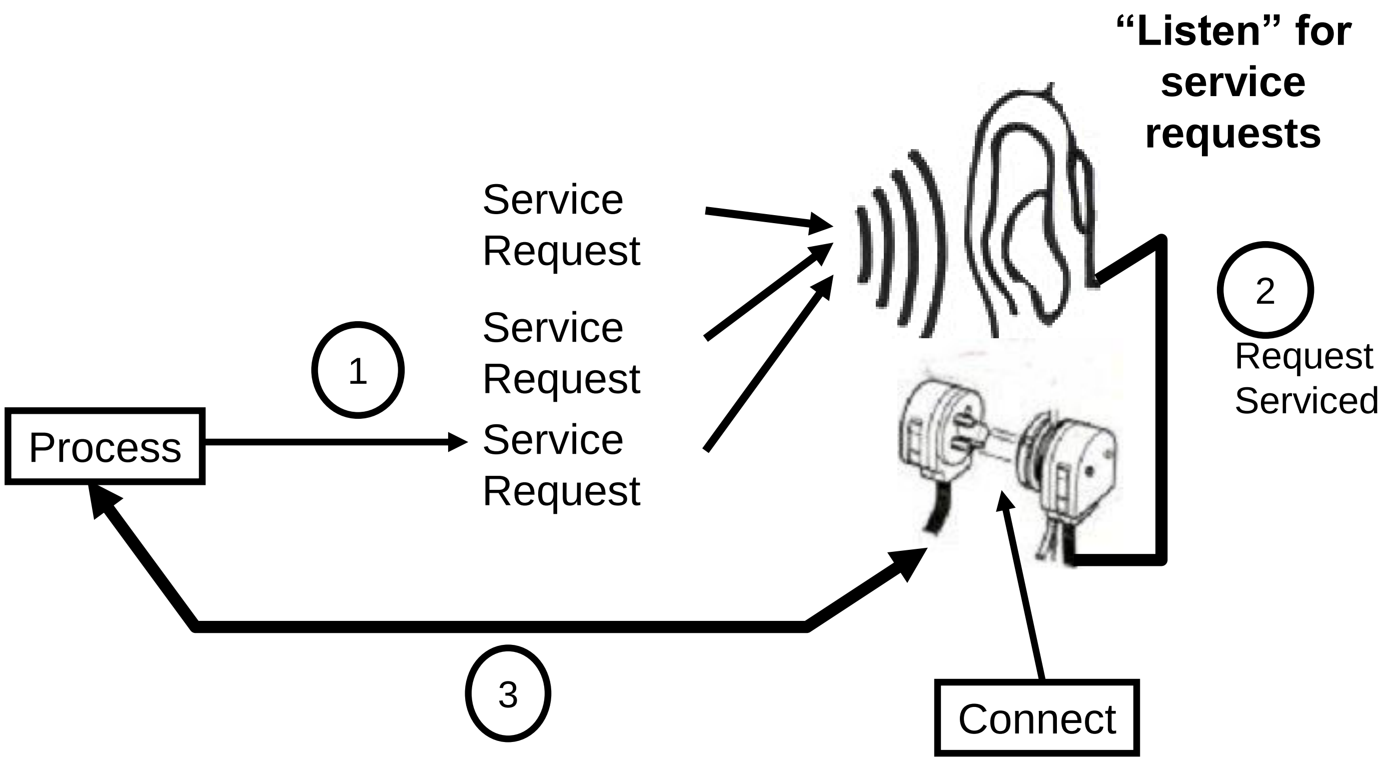
Without acknowledgement

Dest. is not aware

Application indicates dest. for each packet

TYPE OF SOCKETS

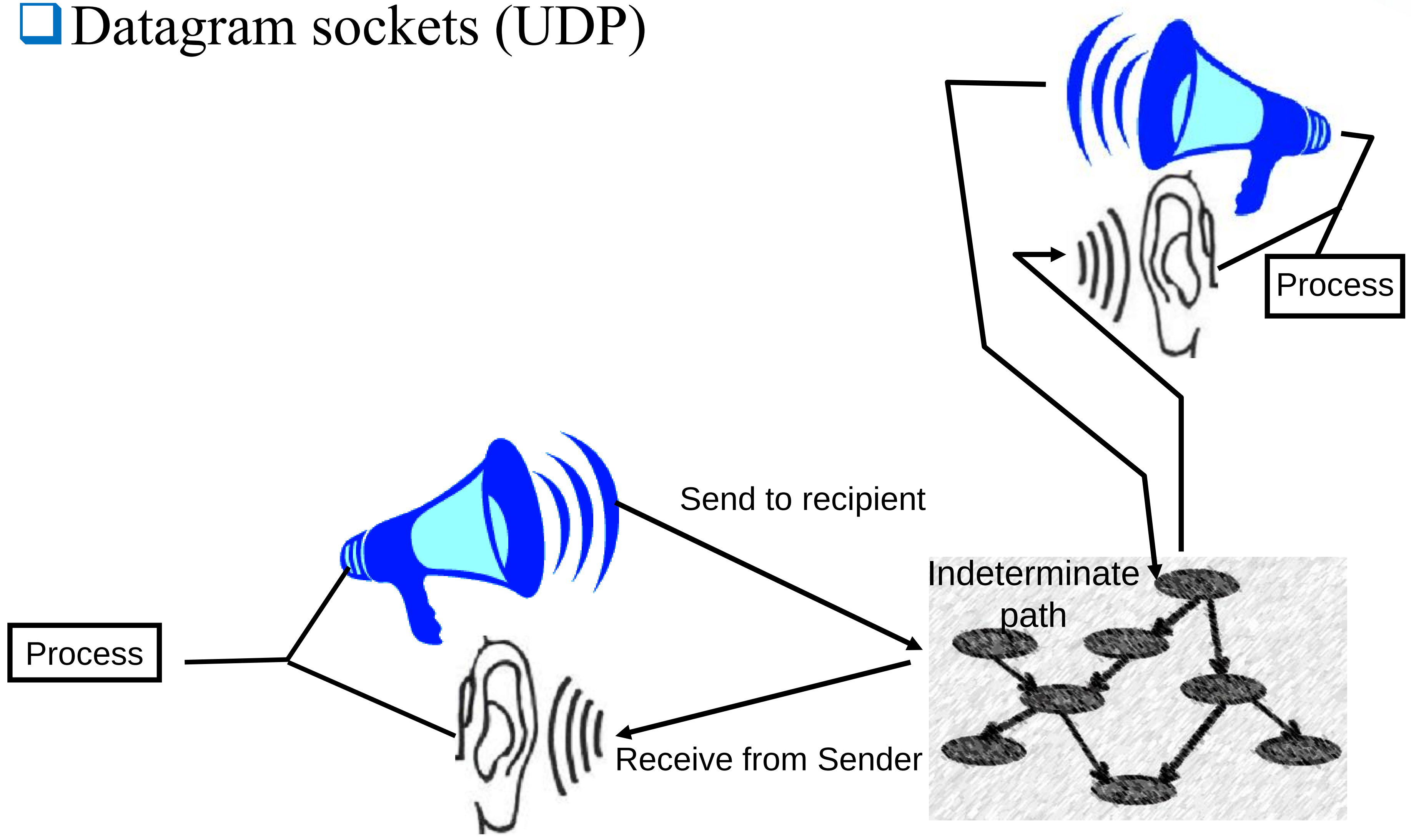
- Streaming sockets (TCP)



Limit on Number of Processes that can successfully request service at a time

TYPE OF SOCKETS

- Datagram sockets (UDP)



PORTS

- A socket involves an interface to send data to/from the network through a port
  - Ports need to be specified at socket creation/configuration time
  - Each host has 65.536 (2^{16}) ports
  - Some ports are reserved for specific applications
    - E.g. 20 and 21 for FTP
    - E.g. 22 for SSH
    - E.g. 23 for Telnet
    - E.g. 80 for HTTP

JAVA.NET

❑ Classes

- The core package *java.net* provides classes that allow to carry out network programming

ContentHandler	ServerSocket
DatagramPacket	Socket
DatagramSocket	SocketImpl
DatagramSocketImplHttpURL Connection	URL
InetAddress	URLConnection
MulticastSocket	URLEncoder

JAVA.NET

❑ Exceptions

BindException	ConnectException
MalformedURLException	NoRouteToHostException
ProtocolException	SocketException
UnknownHostException	UnknownServiceException

JAVA.NET

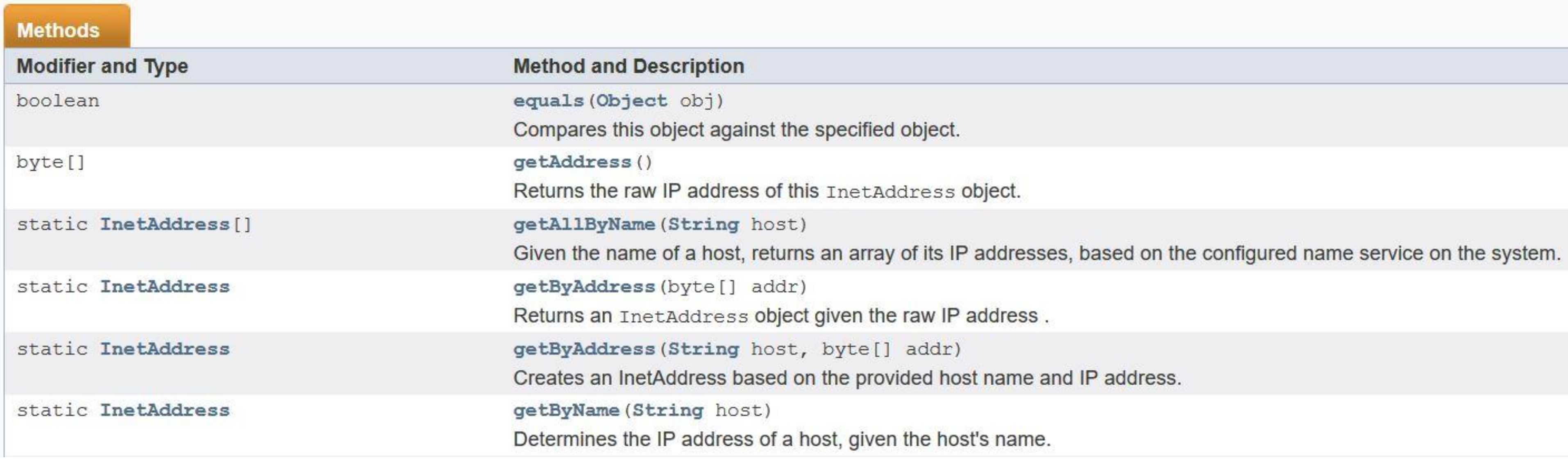
❑ TCP classes

ContentHandler	**ServerSocket**
DatagramPacket	**Socket**
DatagramSocket	SocketImpl
DatagramSocketImplHttpURL Connection	URL
InetAddress	URLConnection
MulticastSocket	URLEncoder

THE INETADDRESS CLASS

- Handles Internet addresses both as host names and as IP addresses
  - Static method ***getByName()*** returns the IP address of a specified host name as an InetAddress object

❑ Example of methods

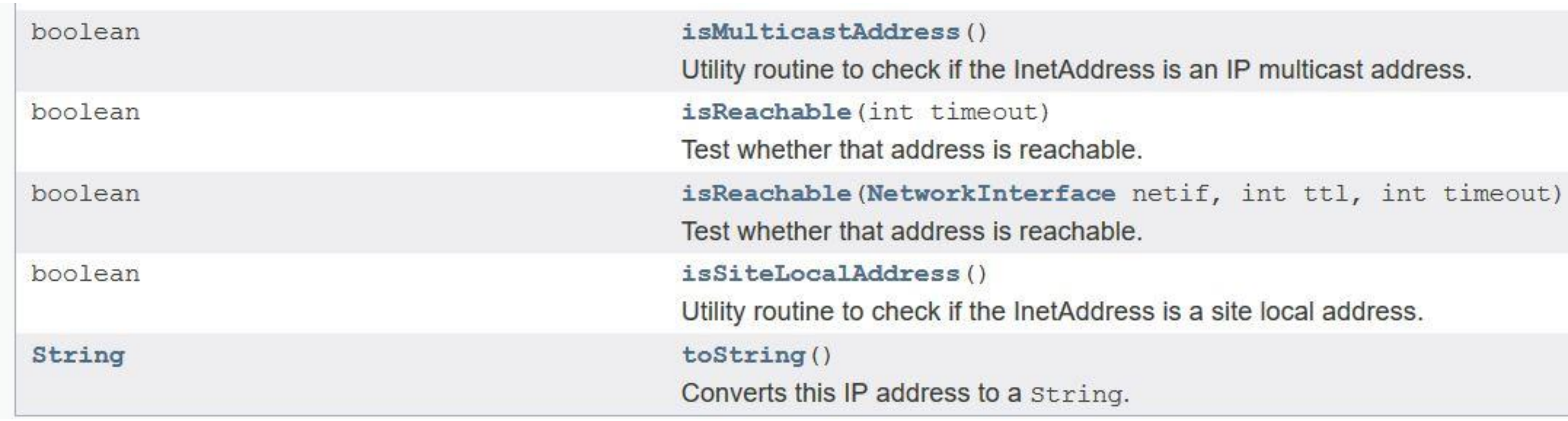


Methods

Modifier and Type	Method and Description
`boolean`	`equals(Object obj)` Compares this object against the specified object.
`byte[]`	`getAddress()` Returns the raw IP address of this `InetAddress` object.
`static InetAddress[]`	`getAllByName(String host)` Given the name of a host, returns an array of its IP addresses, based on the configured name service on the system.
`static InetAddress`	`getByAddress(byte[] addr)` Returns an `InetAddress` object given the raw IP address .
`static InetAddress`	`getByAddress(String host, byte[] addr)` Creates an InetAddress based on the provided host name and IP address.
`static InetAddress`	`getByName(String host)` Determines the IP address of a host, given the host's name.

THE INETADDRESS CLASS

- Example of methods (cont.)



boolean	`isMulticastAddress()` Utility routine to check if the InetAddress is an IP multicast address.
boolean	`isReachable(int timeout)` Test whether that address is reachable.
boolean	`isReachable(NetworkInterface netif, int ttl, int timeout)` Test whether that address is reachable.
boolean	`isSiteLocalAddress()` Utility routine to check if the InetAddress is a site local address.
String	`toString()` Converts this IP address to a `String`.

Comprehensive list is available at: https://docs.oracle.com/javase/8/docs/api/java/net/InetAddress.html

THE INETADDRESS CLASS

Example 1: Find an IP address

```
// File: IPFinder.java
// Get the IP address of a host

import java.net.*;
import java.io.*;
import javax.swing.*;


public class IPFinder
{
		public static void main(String[] args) throws IOException
		{		String host;
				host = JOptionPane.showInputDialog("Please input the server's name");

				try
				{InetAddress address = InetAddress.getByName(host);
				  JOptionPane.showMessageDialog(null,"IP address: " + address.toString());

				}
				catch (UnknownHostException e)
				{JOptionPane.showMessageDialog(null,"Could not find " + host);
				}
		}
}
```

THE INETADDRESS CLASS

❑ Example 2: Retrieving the current machine address

```
import java.net.*;

public class LocalIP
{
    public static void main(String[] args)
     {
                try
                {
                        InetAddress address = InetAddress.getLocalHost();
                        System.out.println (address);
                }
                catch (UnknownHostException e)
                {
                 System.out.println("Could not find local address!");
                }
      }
}
```

THE SOCKET CLASS

- Connection is accomplished via construction
  - Each socket object is associated with exactly one remote host
    - To connect to a different host, you must create a new Socket object.

```
public Socket(String host, int port)  throws UnknownHostException, IOException
```

    - To connect to specified host/port

```
public Socket(InetAddress address, int port) throws IOException
```

    - To connect to specified IP address/port

```
public Socket(String host, int port, InetAddress localAddress,  int localPort) throws IOException
```

    - To connect to specified host/port and bind to specified local address/port

```
public Socket(InetAddress address, int port, InetAddress localAddress,  int localPort) throws
    IOException
```

THE SOCKET CLASS

- Data is transmitted via streams
  - Sending and receiving data is accomplished with output and input streams
    - To get an input stream

```
public InputStream  getInputStream() throws IOException
```

    - To get an output stream

```
public OutputStream getOutputStream() throws IOException
```

THE SOCKET CLASS

- Connection needs to be closed at the end
  - Closing the connection would release the streams
    - To close a socket

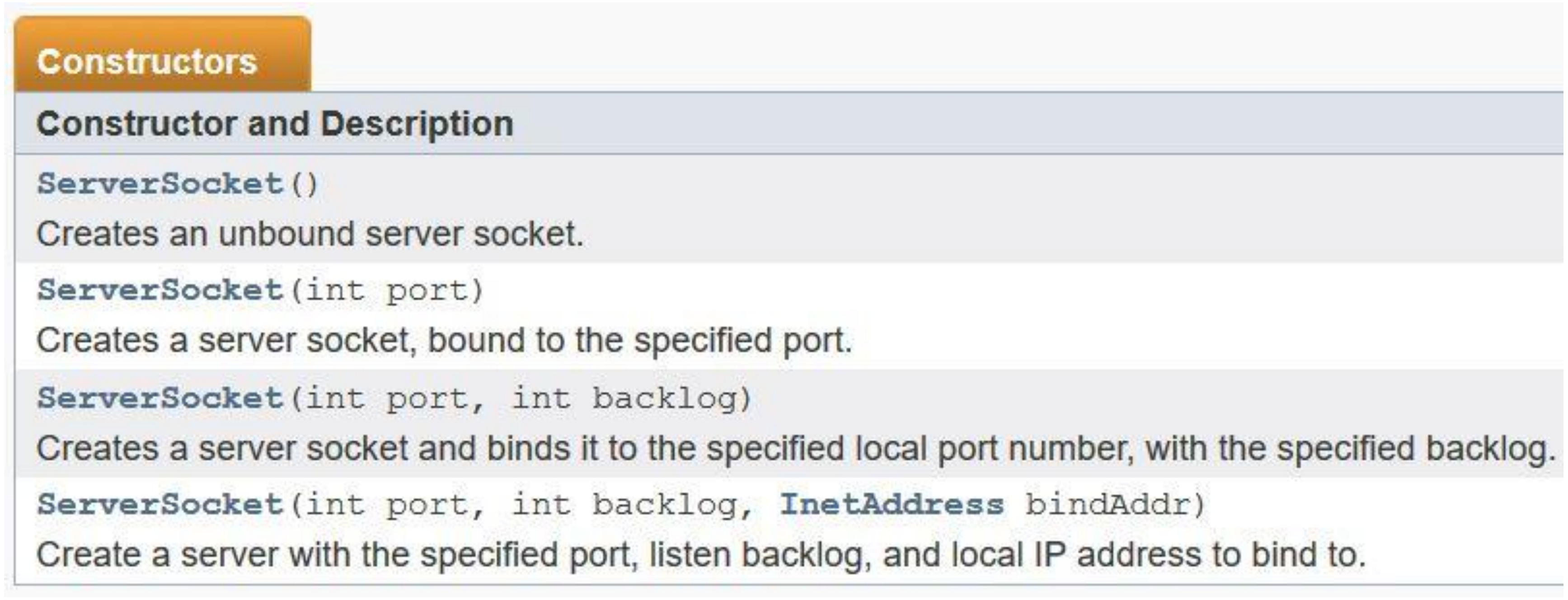
```
public void close() throws IOException
```

THE SERVERSOCKET CLASS

- Implements a specific server socket
- Constructed on a particular port
- Calls ***accept()*** method to listen for incoming connections
  - ***accept()*** blocks until a connection is detected
  - Then, ***accept()*** returns a java.net.Socket object that is used to perform the actual communication with the client (the « plug »)
  - *backlog* is the maximum size of the queue of connection requests

THE SERVERSOCKET CLASS

❑ Constructors

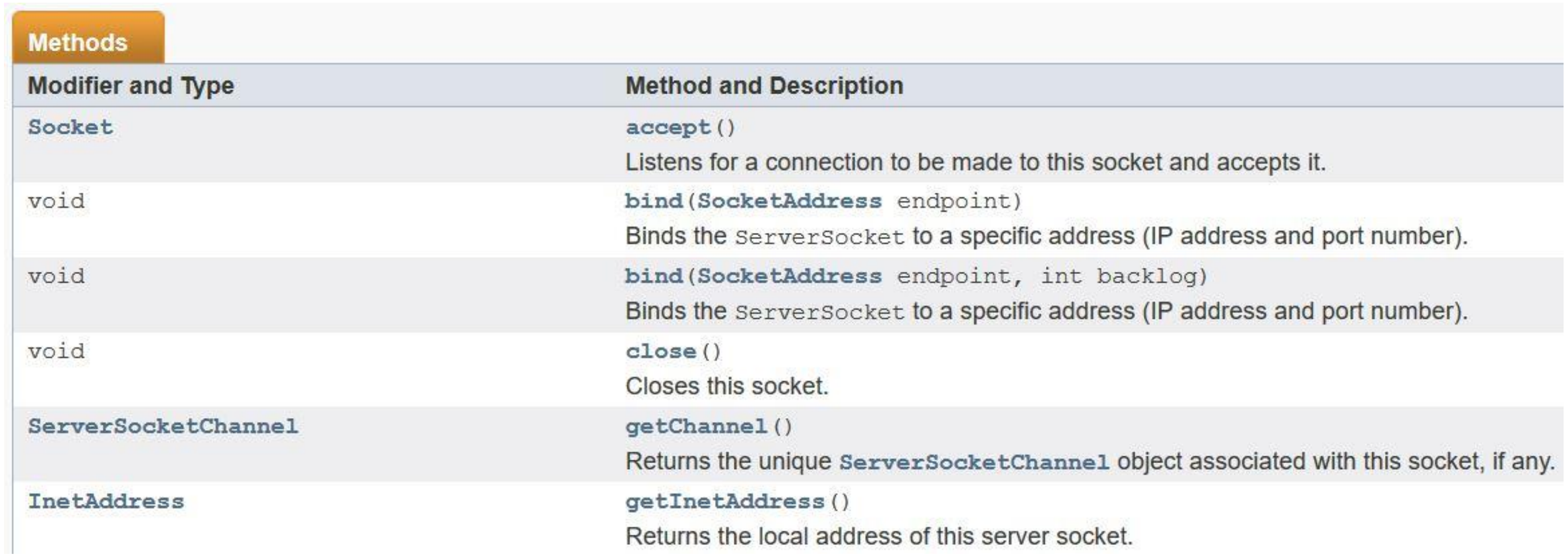


Constructors

Constructor and Description
`ServerSocket()` Creates an unbound server socket.
`ServerSocket(int port)` Creates a server socket, bound to the specified port.
`ServerSocket(int port, int backlog)` Creates a server socket and binds it to the specified local port number, with the specified backlog.
`ServerSocket(int port, int backlog, InetAddress bindAddr)` Create a server with the specified port, listen backlog, and local IP address to bind to.

THE SERVERSOCKET CLASS

- Methods



Methods

Modifier and Type	Method and Description
Socket	accept() Listens for a connection to be made to this socket and accepts it.
void	bind(SocketAddress endpoint) Binds the ServerSocket to a specific address (IP address and port number).
void	bind(SocketAddress endpoint, int backlog) Binds the ServerSocket to a specific address (IP address and port number).
void	close() Closes this socket.
ServerSocketChannel	getChannel() Returns the unique ServerSocketChannel object associated with this socket, if any.
InetAddress	getInetAddress() Returns the local address of this server socket.

Comprehensive list is available at: https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html

THE SERVERSOCKET CLASS

Example – SocketThrdServer.java

SERVER:

1. Create a ServerSocket object

```
ServerSocket servSocket = new ServerSocket(1234);
```

2. Put the server into a waiting state

```
Socket link = servSocket.accept();
```

3. Set up input and output streams

   - use thread to serve this client via *link*

4. Send and receive data

```
out.println(awaiting data…);
String input = in.readLine();
```

5. Close the connection

```
link.close()
```

THE SERVERSOCKET CLASS

❑ Example – SocketThrdServer.java (Cont.)

- Set up input and output streams
  - Methods ***getInputStream()*** and ***getOutputStream()*** of Socket class

```
new BufferedReader(new InputStreamReader(link.getInputStream()));
    PrintWriter out = new PrintWriter(link.getOutputStream(),true);
```

THE SERVERSOCKET CLASS

- Example – SocketThrdServer.java

CLIENT:

1. Establish a connection to the server

   ```
   Socket link = new Socket(<server>,<port>);
   ```

2. Set up input and output streams

3. Send and receive data

4. Close the connection

JAVA.NET

UDP classes

ContentHandler	ServerSocket
DatagramPacket	Socket
DatagramSocket	SocketImpl
DatagramSocketImplHttpURL Connection	URL
InetAddress	URLConnection
MulticastSocket	URLEncoder

THE DATAGRAMSOCKET CLASS

- Implements a connection to port that does the sending and receiving
  - Unlike TCP, there is no distinction between UDP socket and UDP server socket
  - Unlike TCP, DatagramSocket can send to multiple, different addresses (the address to which data goes is stored in the packet, not in the socket)

```
public DatagramSocket() throws SocketException
public DatagramSocket(int port) throws SocketException
public DatagramSocket(int port, InetAddress laddr) throws SocketException
```

THE DATAGRAMSOCKET CLASS

❑ Example – UDPListener.java

SERVER:

1. Create a DatagramSocket object

```
DatagramSocket dgramSocket = new DatagramSocket(1234);
```

2. Create a buffer for incoming datagrams

```
byte[] buffer = new byte[256];
```

3. Create a DatagramPacket object for the incoming datagram

```
DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
```

4. Accept an incoming datagram

```
dgramSocket.receive(inPacket);
```

THE DATAGRAMSOCKET CLASS

- Example – UDPListener.java (Cont.)

SERVER:

5. Accept the sender's address and port from the packet

```
InetAddress clientAddress = inPacket.getAddress();
int clientPort = inPacket.getPort();
```

6. Retrieve the data from the buffer

```
string message = new String(inPacket.getData(), 0, inPacket.getLength());
```

7. Create the response datagram

```
DatagramPacket outPacket = new DatagramPacket(response.getBytes(), response.length(),
                                              clientAddress, clientPort);
```

8. Send the response datagram

```
dgramSocket.send(outPacket)
```

9. Close the DatagramSocket

```
dgram.close();
```

THE DATAGRAMSOCKET CLASS

- Example – UDPTalk.java

CLIENT:

1. Create a DatagramSocket object

```
DatagramSocket dgramSocket = new DatagramSocket;
```

2. Create the outgoing datagram

```
DatagramPacket outPacket = new DatagramPacket(message.getBytes(),
message.length(),host, port);
```

3. Send the datagram message

```
dgramSocket.send(outPacket)
```

4. Create a buffer for incoming datagrams

```
 byte[] buffer = new byte[256];
```

THE DATAGRAMSOCKET CLASS

- Example – UDPTalk.java (Cont.)

CLIENT:

5. Create a DatagramPacket object for the incoming datagram

```
DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
```

6. Accept an incoming datagram

```
dgramSocket.receive(inPacket)
```

7. Retrieve the data from the buffer

```
string response = new String(inPacket.getData(), 0, inPacket.getLength());
```

8. Close the DatagramSocket:

```
dgram.close();
```

THE DATAGRAMPACKET CLASS

- Implements a wrapper for any array of bytes from which data will be sent or into which data will be received
  - Contains the address and port to which the packet will be sent

```
public DatagramPacket(byte[] data, int length)
public DatagramPacket(byte[] data, int length, InetAddress host, int port)
```

THE DATAGRAMPACKET CLASS

❑ Handling data

- Data arrives/is sent as byte array
  - To send int
    - Convert to *String*
    - Use ***getBytes()*** to convert to *byte[]* and send
  - When receiving int
    - Convert *byte[]* to *String*
    - Use ***Integer.ParseInt()*** to convert to *Integer*