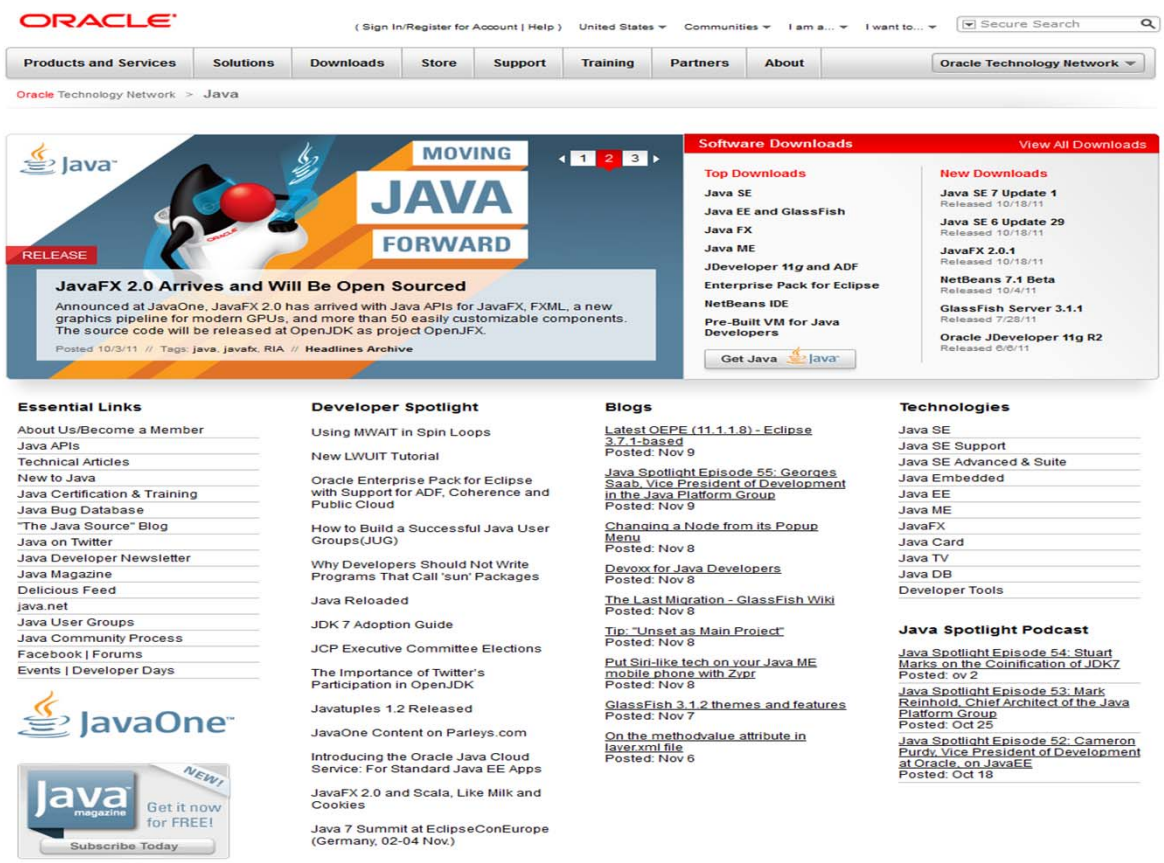
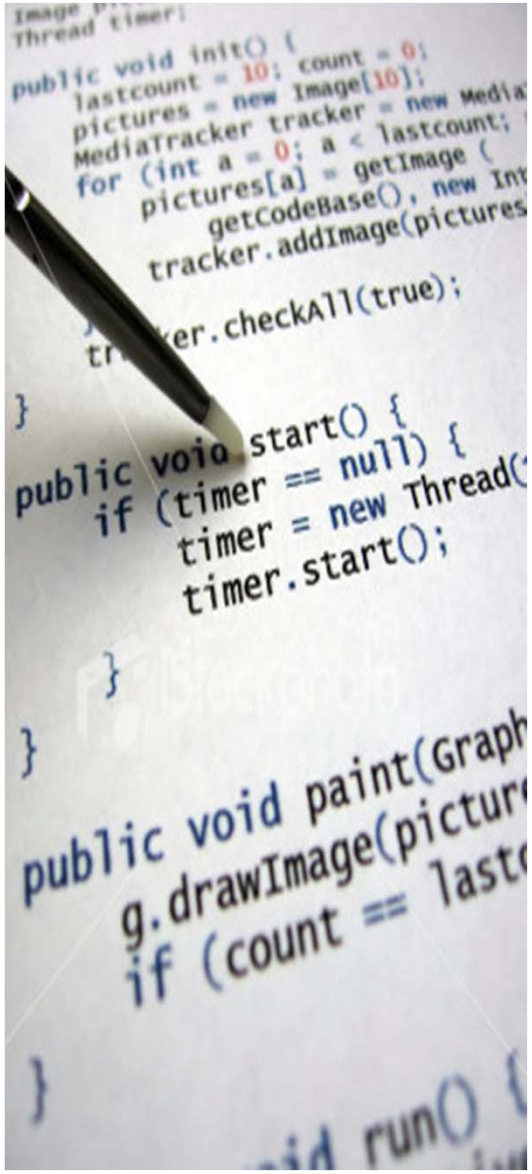
T3: Java Sockets





Bibliography :

- [Kurose10] Apartados 2.1, 2.7, 2.8
- http://www.oracle.com/technetwork/java/index.html
- http://java.com/es/about/
- http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html

Socket programming

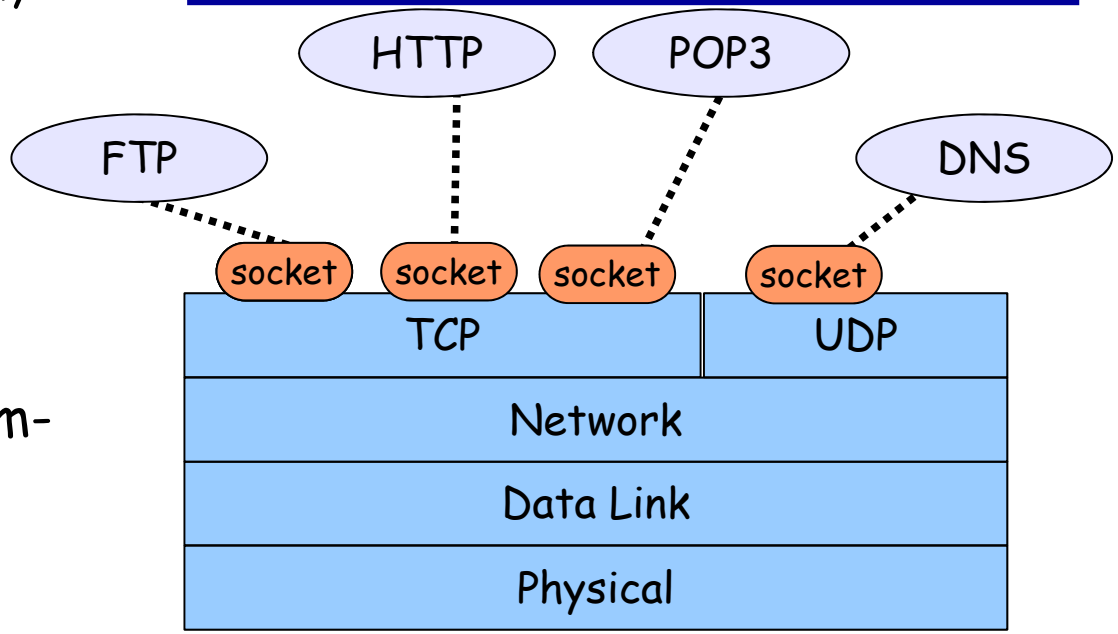
Goal: learn how to build client/server application that communicate using sockets

socket

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented



Java sockets

Socket:

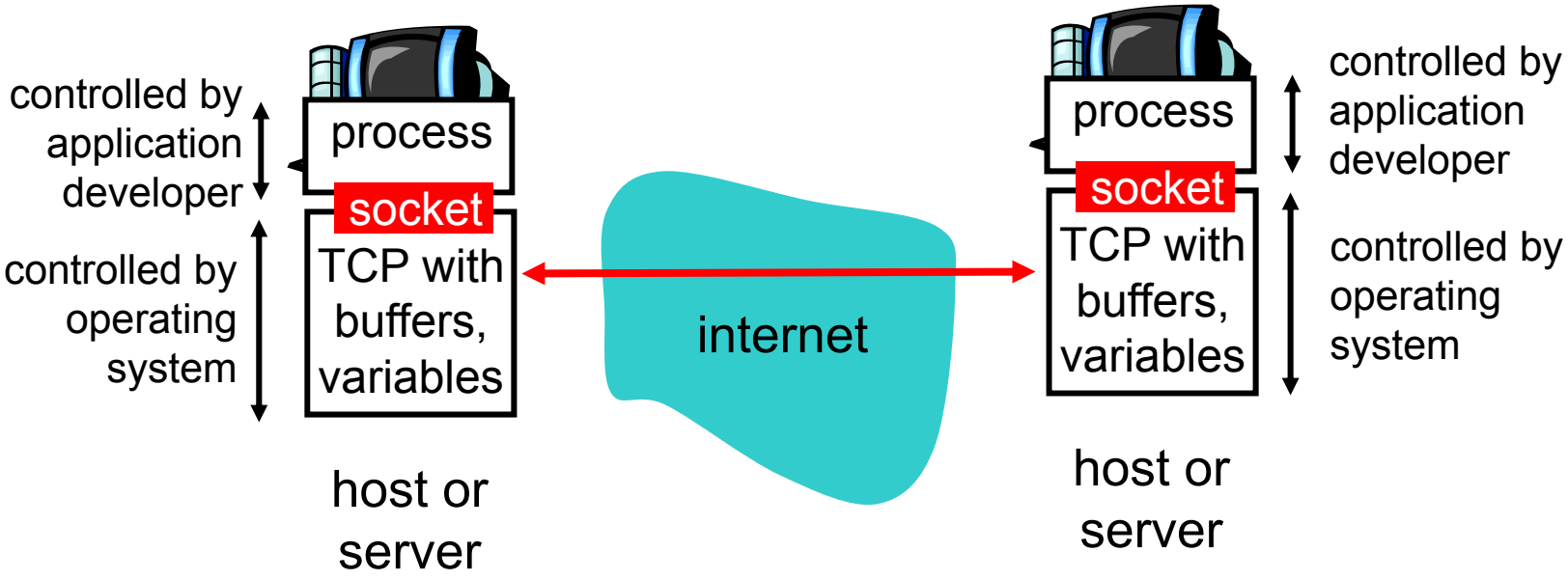
- a socket is an endpoint for communication between two machines.
- a socket is like a door between application process and end-end-transport protocol (UCP or TCP)

- java.net package defines three classes of sockets:
  - **`Socket`** TCP Client
  - **`ServerSocket`** TCP Server
  - **`DatagramSocket`** UDP Client/Server

Socket-programming using TCP

TCP service:

- reliable transfer of *bytes* from one process to another



Socket programming *with TCP*

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

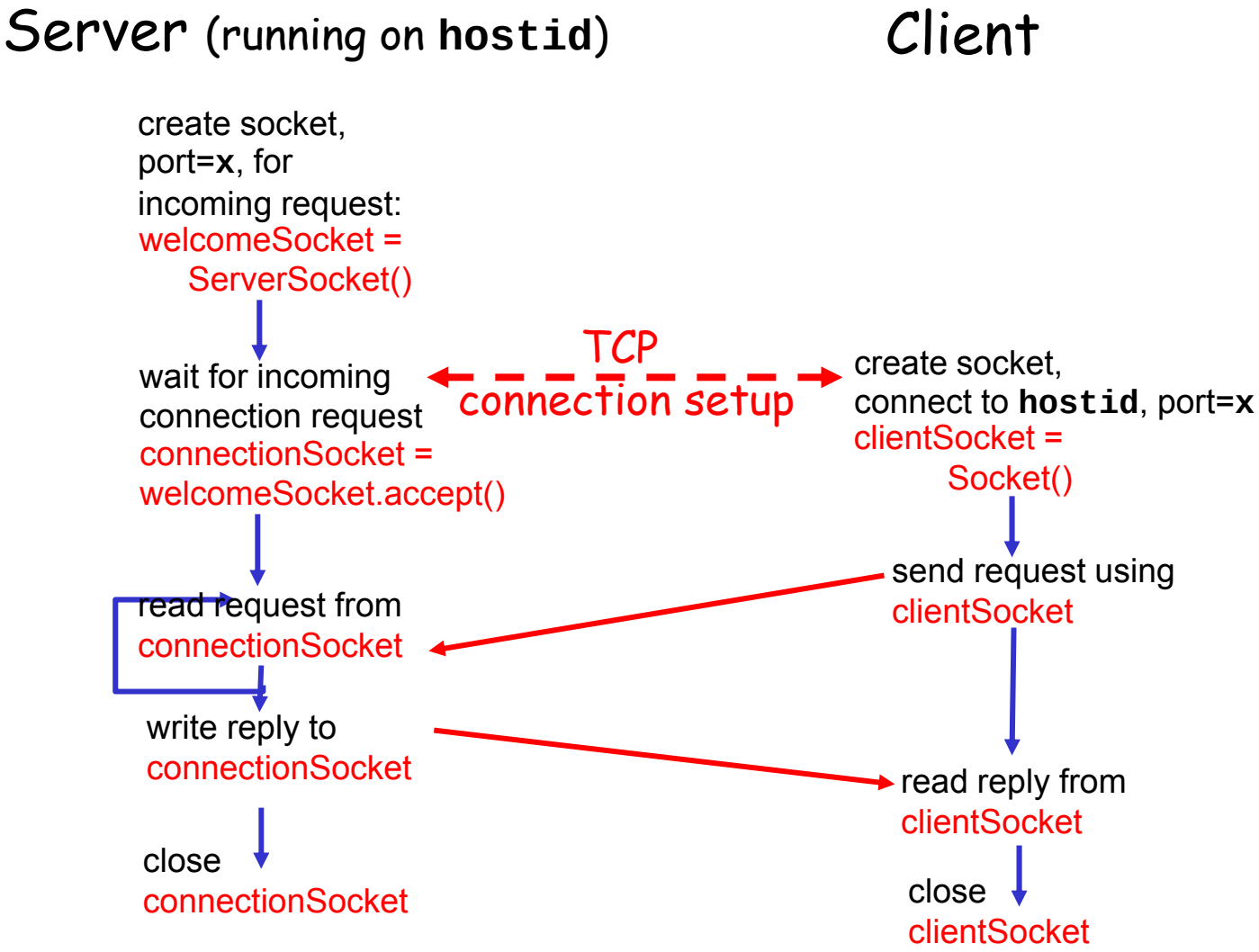
- creating client-local TCP socket
- specifying IP address, port number of server process
- when client creates socket: client TCP establishes connection to server TCP

- when contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint

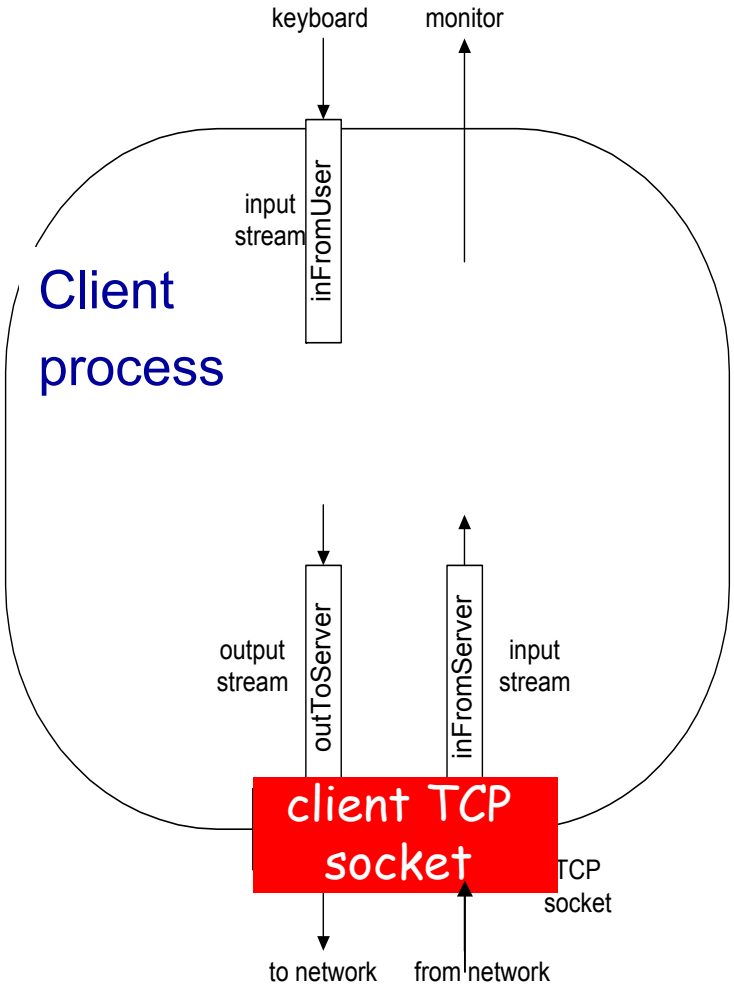
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/server socket interaction: TCP



Stream jargon

- stream is a sequence of characters that flow into or out of a process.
- input stream is attached to some input source for the process, e.g., keyboard or socket.
- output stream is attached to an output source, e.g., monitor or socket.



TCP Clients

- Class **Socket** (**Class java.net.Socket**)
  - This class implements client sockets (also called just "sockets").

Constructors

Modifier	Constructor and Description
	`Socket()` Creates an unconnected socket, with the system-default type of SocketImpl.
	`Socket(InetAddress address, int port)` Creates a stream socket and connects it to the specified port number at the specified IP address.
	`Socket(InetAddress host, int port, boolean stream)` **Deprecated.** *Use DatagramSocket instead for UDP transport.*
	`Socket(InetAddress address, int port, InetAddress localAddr, int localPort)` Creates a socket and connects it to the specified remote address on the specified remote port.
	`Socket(Proxy proxy)` Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.
`protected`	`Socket(SocketImpl impl)` Creates an unconnected Socket with a user-specified SocketImpl.
	`Socket(String host, int port)` Creates a stream socket and connects it to the specified port number on the named host.
	`Socket(String host, int port, boolean stream)` **Deprecated.** *Use DatagramSocket instead for UDP transport.*
	`Socket(String host, int port, InetAddress localAddr, int localPort)` Creates a socket and connects it to the specified remote host on the specified remote port.

Example of a Basic TCP Client

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

public class ClienteTCP0 {
    public static void main(String args[])
                  throws UnknownHostException, IOException {
      Socket s=new Socket("zoltar.redes.upv.es", 7);
      System.out.println("Conectado");

      …..









}
}
```

Methods

Modifier and Type	Method and Description
void	bind(SocketAddress bindpoint) Binds the socket to a local address.
void	close() Closes this socket.
void	connect(SocketAddress endpoint) Connects this socket to the server.
void	connect(SocketAddress endpoint, int timeout) Connects this socket to the server with a specified timeout value.
SocketChannel	getChannel() Returns the unique SocketChannel object associated with this socket, if any.
InetAddress	getInetAddress() Returns the address to which the socket is connected.
InputStream	getInputStream() Returns an input stream for this socket.
boolean	getKeepAlive() Tests if SO_KEEPALIVE is enabled.
InetAddress	getLocalAddress() Gets the local address to which the socket is bound.
int	getLocalPort() Returns the local port number to which this socket is bound.
SocketAddress	getLocalSocketAddress() Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
boolean	getOOBInline() Tests if OOBINLINE is enabled.
OutputStream	getOutputStream() Returns an output stream for this socket.
int	getPort() Returns the remote port number to which this socket is connected.
int	getReceiveBufferSize() Gets the value of the SO_RCVBUF option for this Socket, that is the buffer size used by the platform for input on this Socket.
SocketAddress	getRemoteSocketAddress() Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
boolean	getReuseAddress() Tests if SO_REUSEADDR is enabled.

Read in from Socket

- **Class InputStream**
  - This abstract class is the superclass of all classes representing an input stream of bytes.

Methods

Modifier and Type	Method and Description
int	`available()` Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream.
void	`close()` Closes this input stream and releases any system resources associated with the stream.
void	`mark(int readlimit)` Marks the current position in this input stream.
boolean	`markSupported()` Tests if this input stream supports the `mark` and `reset` methods.
abstract int	`read()` Reads the next byte of data from the input stream.
int	`read(byte[] b)` Reads some number of bytes from the input stream and stores them into the buffer array `b`.
int	`read(byte[] b, int off, int len)` Reads up to `len` bytes of data from the input stream into an array of bytes.
void	`reset()` Repositions this stream to the position at the time the `mark` method was last called on this input stream.
long	`skip(long n)` Skips over and discards `n` bytes of data from this input stream.

Read in from Socket

- **Class Scanner** facilitates the reading of a byte stream
  - to read the next word (String): next ()
  - to read the next integer: nextInt ()
  - to read a floating-point number: nextFloat ()
  - to read the following line: nextLine ()

Read in from Socket – Scanner Class

Methods

Modifier and Type	Method and Description
void	`close()` Closes this scanner.
Pattern	`delimiter()` Returns the `Pattern` this `Scanner` is currently using to match delimiters.
String	`findInLine(Pattern pattern)` Attempts to find the next occurrence of the specified pattern ignoring delimiters.
String	`findInLine(String pattern)` Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
String	`findWithinHorizon(Pattern pattern, int horizon)` Attempts to find the next occurrence of the specified pattern.
String	`findWithinHorizon(String pattern, int horizon)` Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
boolean	`hasNext()` Returns true if this scanner has another token in its input.
boolean	`hasNext(Pattern pattern)` Returns true if the next complete token matches the specified pattern.
boolean	`hasNext(String pattern)` Returns true if the next token matches the pattern constructed from the specified string.
boolean	`hasNextBigDecimal()` Returns true if the next token in this scanner's input can be interpreted as a `BigDecimal` using the `nextBigDecimal()` method.
boolean	`hasNextBigInteger()` Returns true if the next token in this scanner's input can be interpreted as a `BigInteger` in the default radix using the `nextBigInteger()` method.
boolean	`hasNextBigInteger(int radix)` Returns true if the next token in this scanner's input can be interpreted as a `BigInteger` in the specified radix using the `nextBigInteger()` method.
boolean	`hasNextBoolean()` Returns true if the next token in this scanner's input can be interpreted as a boolean value using a case insensitive pattern created from the string "true|false".
boolean	`hasNextByte()` Returns true if the next token in this scanner's input can be interpreted as a byte value in the default radix using the `nextByte()` method.
boolean	`hasNextByte(int radix)` Returns true if the next token in this scanner's input can be interpreted as a byte value in the specified radix using the `nextByte()` method.

String	next() Finds and returns the next complete token from this scanner.
String	next(Pattern pattern) Returns the next token if it matches the specified pattern.
String	next(String pattern) Returns the next token if it matches the pattern constructed from the specified string.
BigDecimal	nextBigDecimal() Scans the next token of the input as a BigDecimal.
BigInteger	nextBigInteger() Scans the next token of the input as a BigInteger.
BigInteger	nextBigInteger(int radix) Scans the next token of the input as a BigInteger.
boolean	nextBoolean() Scans the next token of the input into a boolean value and returns that value.
byte	nextByte() Scans the next token of the input as a byte.
byte	nextByte(int radix) Scans the next token of the input as a byte.
double	nextDouble() Scans the next token of the input as a double.
float	nextFloat() Scans the next token of the input as a float.
int	nextInt() Scans the next token of the input as an int.
int	nextInt(int radix) Scans the next token of the input as an int.
String	nextLine() Advances this scanner past the current line and returns the input that was skipped.
long	nextLong() Scans the next token of the input as a long.
long	nextLong(int radix) Scans the next token of the input as a long.
short	nextShort() Scans the next token of the input as a short.
short	nextShort(int radix) Scans the next token of the input as a short.

Read in from Socket - Scanner Class

- For example, this code allows a user to read a line from a Socket:

```
import java.util.Scanner;
...

Scanner inFromSocket=new Scanner(s.getInputStream());
inFromSocket.nextLine();
```

```
Scanner inFromSocket= new Scanner(s.getInputStream());
...
while (inFromSocket.hasNext()){
  System.out.println(inFromSocket.nextLine());
}
```

Read in from Socket – Scanner Class

- For example, this code allows a user to read a line from the keyboard:

```
import java.util.Scanner;
...

Scanner inFromKeyboard=new Scanner(System.in);
System.out.println("Name of destination
server: ");
String server = inFromKeyboard.nextLine();
System.out.println("Enter the destination
port: ");
int port = inFromKeyboard.nextInt();
```

Write out to Socket

- **Class OutputStream**
  - This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Methods

Modifier and Type	Method and Description
void	`close()` Closes this output stream and releases any system resources associated with this stream.
void	`flush()` Flushes this output stream and forces any buffered output bytes to be written out.
void	`write(byte[] b)` Writes `b.length` bytes from the specified byte array to this output stream.
void	`write(byte[] b, int off, int len)` Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.
abstract void	`write(int b)` Writes the specified byte to this output stream.

Write out to Socket

- **Class PrintWriter**
  - PrintWriter lets you send text (characters)
  - It has methods for writing text:

    `print(String s )`, `println(String s)`, `printf(String s, Object ... args)`

Constructors

Constructor and Description
`PrintWriter(File file)` Creates a new PrintWriter, without automatic line flushing, with the specified file.
`PrintWriter(File file, String csn)` Creates a new PrintWriter, without automatic line flushing, with the specified file and charset.
`PrintWriter(OutputStream out)` Creates a new PrintWriter, without automatic line flushing, from an existing OutputStream.
`PrintWriter(OutputStream out, boolean autoFlush)` Creates a new PrintWriter from an existing OutputStream.
`PrintWriter(String fileName)` Creates a new PrintWriter, without automatic line flushing, with the specified file name.
`PrintWriter(String fileName, String csn)` Creates a new PrintWriter, without automatic line flushing, with the specified file name and charset.
`PrintWriter(Writer out)` Creates a new PrintWriter, without automatic line flushing.
`PrintWriter(Writer out, boolean autoFlush)` Creates a new PrintWriter.

Methods

Modifier and Type	Method and Description
void	flush() Flushes the stream.
PrintWriter	format(Locale l, String format, Object... args) Writes a formatted string to this writer using the specified format string and arguments.
PrintWriter	format(String format, Object... args) Writes a formatted string to this writer using the specified format string and arguments.
void	print(boolean b) Prints a boolean value.
void	print(char c) Prints a character.
void	print(char[] s) Prints an array of characters.
void	print(double d) Prints a double-precision floating-point number.
void	print(float f) Prints a floating-point number.
void	print(int i) Prints an integer.
void	print(long l) Prints a long integer.
void	print(Object obj) Prints an object.
void	print(String s) Prints a string.
PrintWriter	printf(Locale l, String format, Object... args) A convenience method to write a formatted string to this writer using the specified format string and arguments.
PrintWriter	printf(String format, Object... args) A convenience method to write a formatted string to this writer using the specified format string and arguments.
void	println() Terminates the current line by writing the line separator string.
void	println(String x) Prints a String and then terminates the line.
protected void	setError() Indicates that an error has occurred.

Write out to Socket – PrintWriter Class

- For example, this code allows a user to write a line to a Socket:

```
PrintWriter outToSocket= new
PrintWriter(s.getOutputStream());
outToSocket.print("GET / HTTP/1.0" + "\r\n");
outToSocket.flush();
```

```
PrintWriter outToSocket = new
PrintWriter(s.getOutputStream(), true);
outToSocket.print("GET / HTTP/1.0" + "\r\n");
// needless outToSocket.flush();
```

Basic TCP Client

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

public class ClienteTCP0 {
    public static void main(String args[])
                    throws UnknownHostException, IOException {
      Socket s=new Socket("zoltar.redes.upv.es", 7);
      System.out.println("Connected");
      PrintWriter outToSocket= new PrintWriter(s.getOutputStream());
      outToSocket.print("Hello World!\r\n");
      outToSocket.flush();
      Scanner inFromSocket=new Scanner(s.getInputStream());
      System.out.println(inFromSocket.nextLine());
      s.close();
      System.out.println("Disconnected");
    }
}
```

- This client connects to the ECHO server (port 7), sends and receives a line and then closes the connection.

TCP Client modified - Catching Exceptions

```
public class ClienteTCP1 {
    public static void main(String args[])
                throws UnknownHostException, IOException {
    try{
          Socket s=new Socket(“unknownHost.redes.upv.es", 7);
          System.out.println("Connected");
          PrintWriter outToSocket= new PrintWriter(s.getOutputStream());
          outToSocket.print(“Hello World!\r\n");
          outToSocket.flush();
          Scanner inFromSocket=new Scanner(s.getInputStream());
          System.out.println(inFromSocket.nextLine());
          s.close();
          System.out.println("Disconnected");
        } catch (UnknownHostException e) {
            System.out.println("Host desconocido");
            System.out.println(e);
        } catch (IOException e) {
            System.out.println("No se puede conectar");
            System.out.println(e);
        }
    }
}
```

Class InetAddress

- This class represents an Internet Protocol (IP) address.
- An instance of an InetAddress consists of an IP address and possibly its corresponding host name
  - depending on whether it is constructed with a host name or whether it has already done reverse host name resolution.
- Some important methods
  - static InetAddress getByName(String name)
    - Gets the IP address associated with a name
  - String getHostAddress()
    - Returns the IP address in "aa.bb.cc.dd" format
    - Example:

```
InetAddress inet =InetAddress.getByName("www.mit.edu");
System.out.println ("IP  : " + inet.getHostAddress());
```

  - String getHostName()
    - Returns the host name
    - Example:

```
InetAddress address = InetAddress.getLocalHost();
String sHostName = address.getHostName();
System.out.println(sHostName);
```

How to get TCP connection information

```
public class EjemploInetAddress {
    public static void main(String args[])
                throws UnknownHostException, IOException {
    try{
        InetAddress zoltar = InetAddress.getByName("zoltar.redes.upv.es");
        Socket s=new Socket(zoltar, 7);
        System.out.println("Connected");
        System.out.print("local Host:"); System.out.println(s.getLocalAddress().getHostName());
        System.out.print("local IP:"); System.out.println(s.getLocalAddress().getHostAddress());
        System.out.print("local Port:"); System.out.println (s.getLocalPort());
        System.out.print("remote Host:"); System.out.println(s.getInetAddress().getHostName());
        System.out.print("remote IP:"); System.out.println(s.getInetAddress().getHostAddress());
        System.out.print("remote Port:"); System.out.println(s.getPort());
        s.close();
        System.out.println("Disconnected");
        } catch (UnknownHostException e) {
            System.out.println("Disconnected Host");
            System.out.println(e);
        } catch (IOException e) {
            System.out.println("Socket can not connect");
            System.out.println(e);
        }
    }
}
```