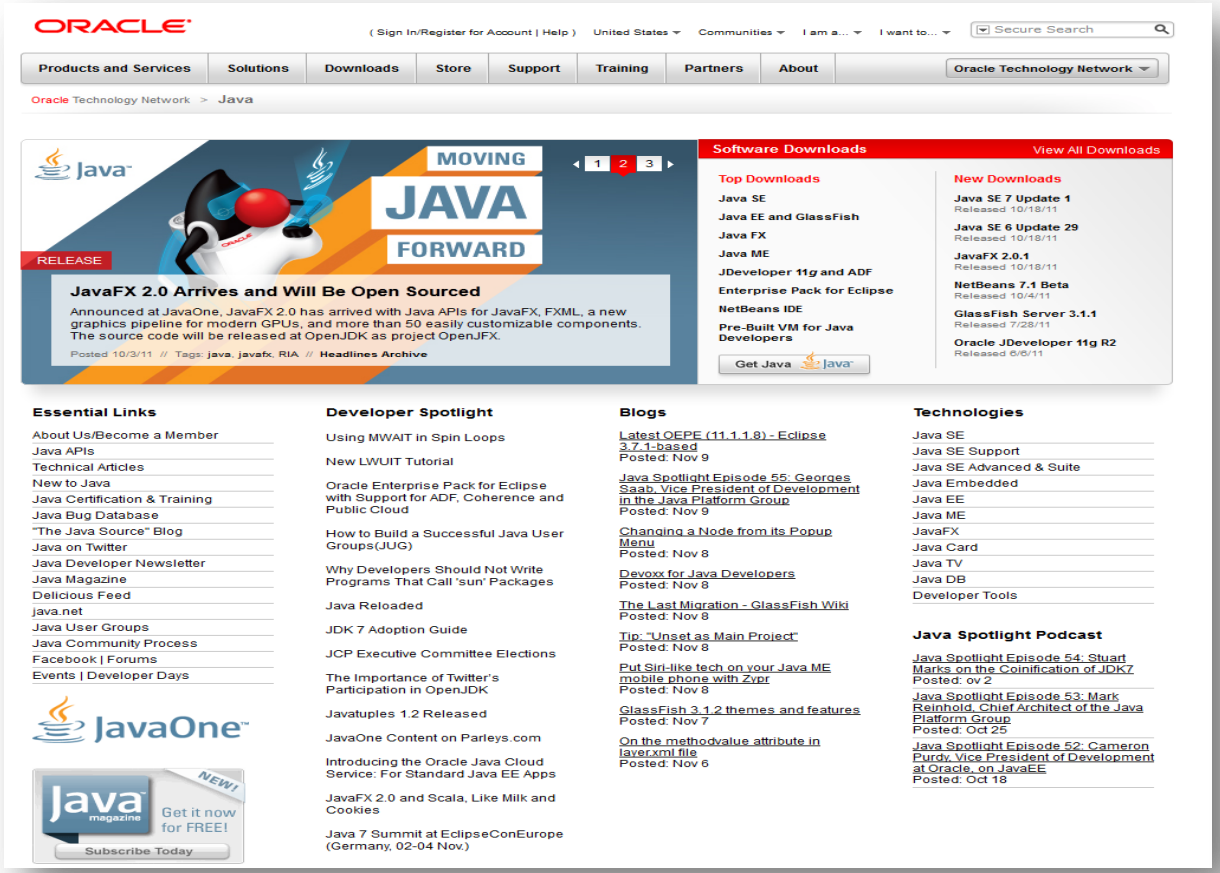
Tema 3: Java y los Sockets

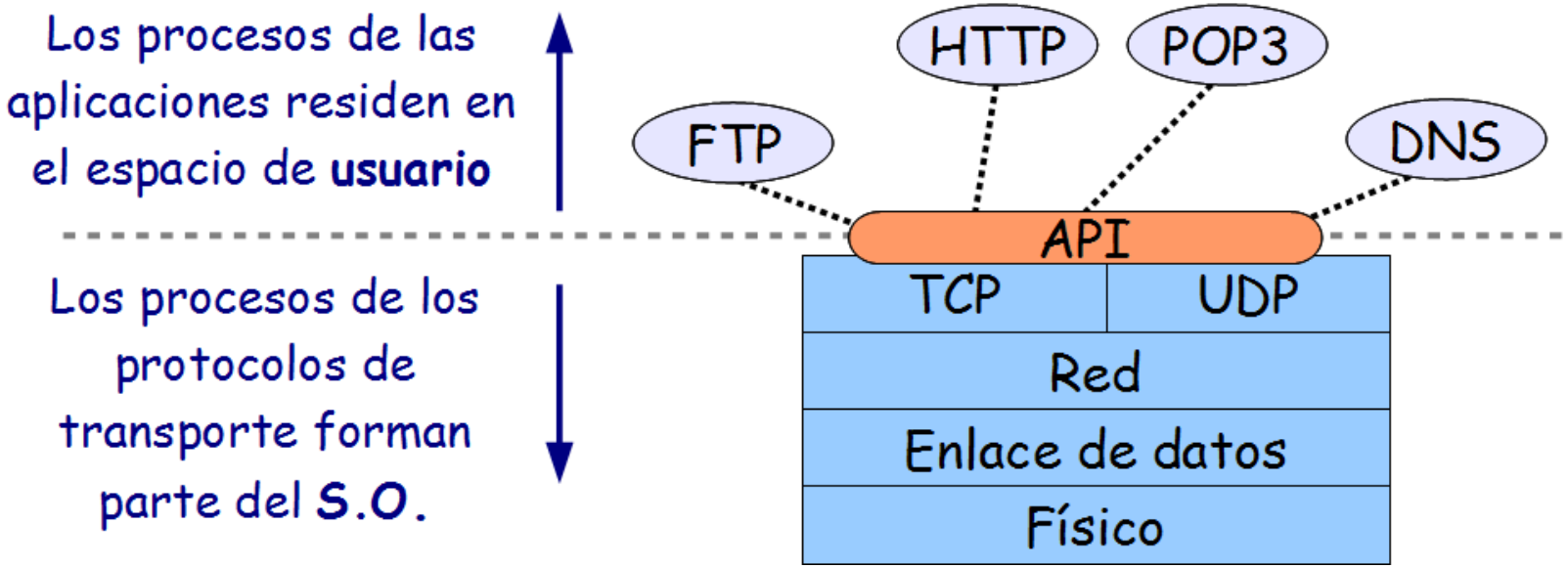


Bibliografía:

- [Kurose10] Apartados 2.1, 2.7, 2.8
- http://www.oracle.com/technetwork/java/index.html
- http://java.com/es/about/
- http://download.oracle.com/javase/6/docs/api/index.html

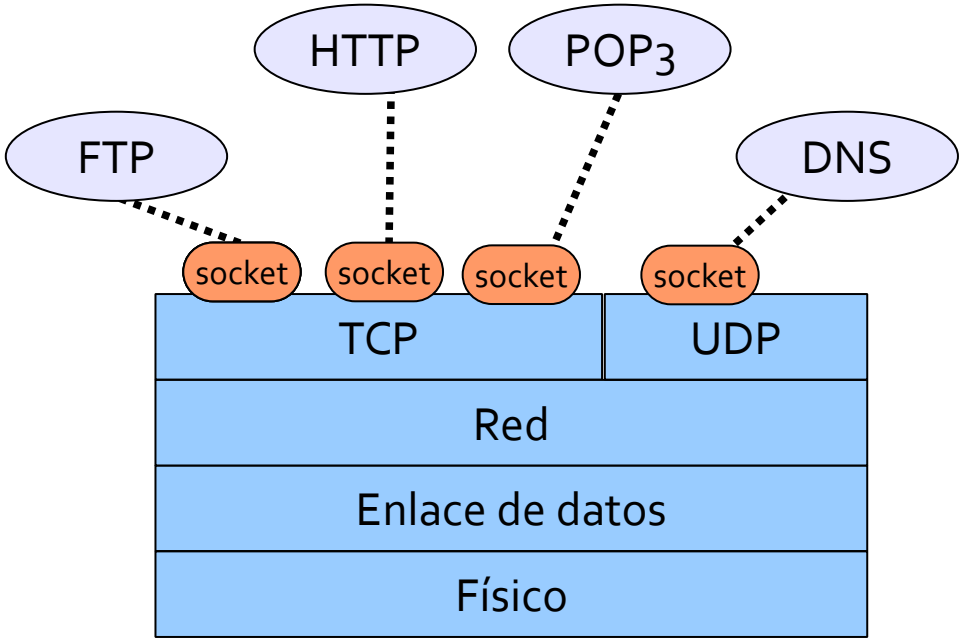
La abstracción de los sockets

- Clientes y servidores utilizan protocolos de transporte
- Se necesita un mecanismo para ponerlos en contacto
  - API (*Application Programming Interface*)



API socket

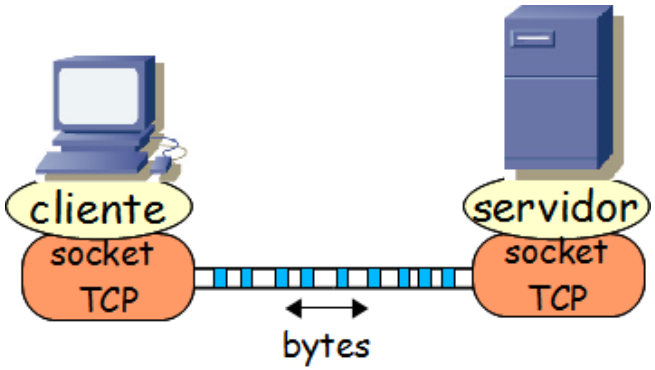
- Originalmente diseñado en BSD Unix
  - Permite a las aplicaciones utilizar los protocolos de la pila TCP/IP
- Define las operaciones permitidas y sus argumentos
  - Parecido a la forma de acceder a los ficheros en Unix
  - Operaciones: open, read, write, close
  - Cuando se abre un fichero obtenemos un descriptor
- Es un estándar de facto



Tipos de sockets

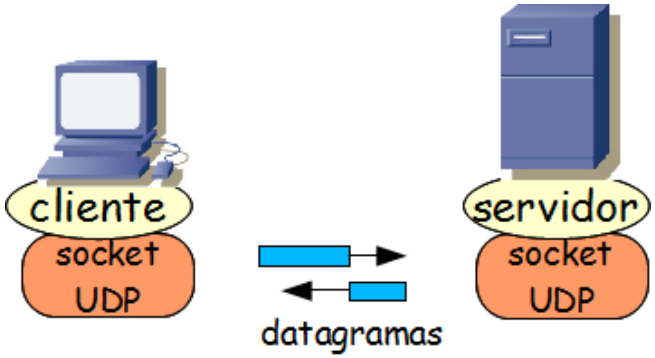
Sockets TCP

- Las aplicaciones piden al S.O. una comunicación controlada por TCP:
  - Orientada a la conexión
  - Comunicación fiable y ordenada
- También se denominan sockets de tipo Stream



Sockets UDP

- Las aplicaciones piden al S.O. una comunicación controlada por UDP:
  - Transferencia de bloques de datos
  - Sin conexión ni fiabilidad ni entrega ordenada
  - Permite difusiones
- También se denominan sockets de tipo Datagram



- Dentro del paquete java.net existen tres clases de sockets:
  - `Socket` Cliente TCP
  - `ServerSocket` Servidor TCP
  - `DatagramSocket` Cliente/Servidor UDP

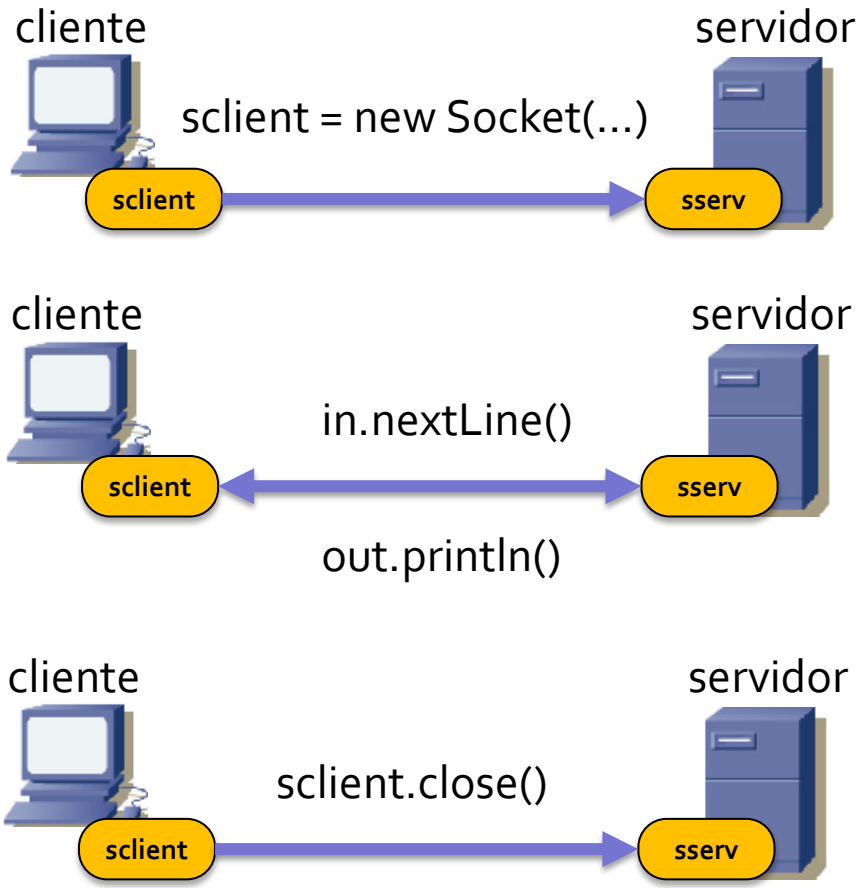
- También hay otras clases auxiliares que facilitan la programación de aplicaciones en red
  - Ver referencias

Servidor:

- Ha de estar en ejecución
- Debe haber creado un socket (`sserv`) donde recibir a los clientes que conectan con él

Cliente:

- Crea un socket (`sclient`) y lo conecta con el del servidor
- Transfiere información
- Cierra el socket y la conexión (a veces lo hace el servidor)



- Clase `Socket`
- Constructores más frecuentes
  - Crean un socket y lo conectan con el servidor y el puerto indicados
  - `Socket(String nombre, int puerto)`
  - `Socket(InetAddress dirIP, int puerto)`

- Clase `Socket`
- Algunos métodos importantes
  - `close()`: Cierra el socket
  - `InputStream getInputStream()`
    - Proporciona un descriptor para leer del socket
    - `InputStream` proporciona un flujo de bytes
      - Se puede leer un byte: `read()`
      - O un grupo de bytes: `read(byte[] b)`
  - `OutputStream getOutputStream()`
    - Proporciona un descriptor para escribir en el socket
    - `OutputStream` admite un flujo de bytes
      - Se puede escribir un byte: `write(int b)`
      - O un grupo de bytes: `write(byte[] b)`

- **`Scanner`** facilita la lectura de un flujo de bytes
  - Se puede leer la palabra siguiente (String): **`next()`**
  - el siguiente entero: **`nextInt()`**
  - el siguiente número en coma flotante: **`nextFloat()`**
  - o la siguiente línea: **`nextLine()`**
- Ejemplo:

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

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

Gestión de los flujos de entrada

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

- A veces también es interesante leer de la entrada estándar (teclado)
- Ejemplo:

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

Scanner leeTeclado=new Scanner(System.in);
System.out.println("Nombre del servidor destino: ");
String servidor = leeTeclado.nextLine();
System.out.println("Introduce el puerto destino: ");
int pto = leeTeclado.nextInt();
```

- PrintWriter permite enviar texto (caracteres)
  - Tiene métodos que permiten escribir texto:
    `print(String s )`, `println(String s)`, `printf(String s, Object ... args)`
- Ejemplo:

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

- Ejemplo:

```
PrintWriter salida = new PrintWriter(s.getOutputStream(), true);
salida.printf("GET / HTTP/1.0" + "\r\n");
// no hace falta salida.flush();
```

Cliente TCP básico

```
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");
      PrintWriter salida = new PrintWriter(s.getOutputStream());
      salida.printf("Hola Mundo!\r\n");
      salida.flush();
      Scanner entrada=new Scanner(s.getInputStream());
      System.out.println(entrada.nextLine());
      s.close();
      System.out.println("Desconectado");
    }
}
```

- Este cliente se conecta al servidor ECHO (puerto 7), envía y recibe una línea y después cierra la conexión

Excepciones de la clase Socket

- Al intentar conectar un socket con un servidor hay dos problemas típicos, que generan excepciones en Java:
  - No se puede resolver el nombre del servidor
    - Se genera una excepción UnknownHostException
  - Se ha resuelto el nombre pero no se puede establecer la conexión
    - Se genera una excepción IOException
- Estas excepciones se pueden manejar mediante

  try{...}
  catch(Exception e) {instrucciones al producirse la excepción}
- Hay que capturar primero la excepción más específica
  - (UnknownHostException)

IOException y sus subclases

➔ http://download.oracle.com/javase/1.4.2/docs/api/java/io/IOException.html

java.io

Class IOException

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.io.IOException
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

ChangedCharSetException, CharacterCodingException, CharConversionException, ClosedChannelException, EOFException, FileLockInterruptionException, FileNotFoundException, IIOException, InterruptedIOException, MalformedURLException, ObjectStreamException, ProtocolException, RemoteException, SocketException, SSLException, SyncFailedException, UnknownHostException, UnknownServiceException, UnsupportedEncodingException, UTFDataFormatException, ZipException

Cliente TCP modificado

```
public class ClienteTCP1 {
    public static void main(String args[])
                throws UnknownHostException, IOException {
    try{
          Socket s=new Socket("noexiste.redes.upv.es", 7);
          System.out.println("Conectado");
          PrintWriter salida = new PrintWriter(s.getOutputStream());
          salida.print("Hola Mundo!\r\n");
          salida.flush();
          Scanner entrada=new Scanner(s.getInputStream());
          System.out.println(entrada.nextLine());
          s.close();
          System.out.println("Desconectado");
        } 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);
        }
    }
}
```

Clase InetAddress

- InetAddress es la clase que se utiliza para almacenar direcciones IP en Java
- Algunos métodos importantes
  - static InetAddress getByName(String nombre)
    - Obtiene la dirección IP asociada a un nombre
  - String getHostAddress()
    - Devuelve la dirección IP en formato "aa.bb.cc.dd"
    - Ejemplo:

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

  - String getHostName()
    - Devuelve el nombre del host
    - Ejemplo:

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

Obteniendo información de la conexión

```
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("Conectado");
        System.out.print("Host local:"); System.out.println(s.getLocalAddress().getHostName());
        System.out.print("IP local:"); System.out.println(s.getLocalAddress().getHostAddress());
        System.out.print("Puerto local:"); System.out.println (s.getLocalPort());
        System.out.print("Host remoto:"); System.out.println(s.getInetAddress().getHostName());
        System.out.print("IP remota:"); System.out.println(s.getInetAddress().getHostAddress());
        System.out.print("Puerto remoto:"); System.out.println(s.getPort());
        s.close();
        System.out.println("Desconectado");
        } 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);
        }
    }
}
```

- Argumentos de un programa desde la línea de órdenes:
  - `public static void main(String args[])`
  - args[0], args[1], …
  - En ese caso, args.length debe ser mayor o igual a uno

Strings: obtener información

- Para obtener la longitud, número de caracteres que guarda un string se llama a la función miembro *length*.
  - `String str="El primer programa";`
  - `int longitud=str.length();`
- Podemos conocer si un string comienza con un determinado prefijo, llamando al método *startsWith*, que devuelve **true** o **false**, según que el string comience o no por dicho prefijo
  - `String str="El primer programa";`
  - `boolean resultado=str.startsWith("El");`
  - En este ejemplo la variable resultado tomará el valor **true**.
- De modo similar, podemos saber si un string finaliza con un conjunto dado de caracteres, mediante la función miembro *endsWith*.
  - `String str="El primer programa";`
  - `boolean resultado=str.endsWith("programa");`
- Si se quiere obtener la posición de la primera ocurrencia de la letra p, se usa la función *indexOf*.
  - `String str="El primer programa";`
  - `int pos=str.indexOf('p');`
- Para obtener las sucesivas posiciones de la letra p, se llama a otra versión de la misma función
  - `pos=str.indexOf('p', pos+1);`
  - El segundo argumento le dice a la función indexOf que empiece a buscar la primera ocurrencia de la letra p a partir de la posición pos+1.
- Otra versión de *indexOf* busca la primera ocurrencia de un substring dentro del string.
  - `String str="El primer programa";`
  - `int pos=str.indexOf("pro");`

Strings: Comparación

- Metodo equals():

```
String str1="El lenguaje Java";
String str2=new String("El lenguaje Java");
 if(str1==str2){
            System.out.println("Los mismos objetos");
 }else{
            System.out.println("Distintos objetos");
 }
 if(str1.equals(str2)){
            System.out.println("El mismo contenido");
 }else{
            System.out.println("Distinto contenido");
 }
```

Compara objetos

Compara strings

- Metodo compareTo()
  - devuelve un entero menor que cero si el objeto string es menor (en orden alfabético) que el string dado, cero si son iguales, y mayor que cero si el objeto string es mayor que el string dado.

```
String str="Tomás";
int resultado=str.compareTo("Alberto");
```

  - La variable entera resultado tomará un valor mayor que cero, ya que Tomás está después de Alberto en orden alfabético.