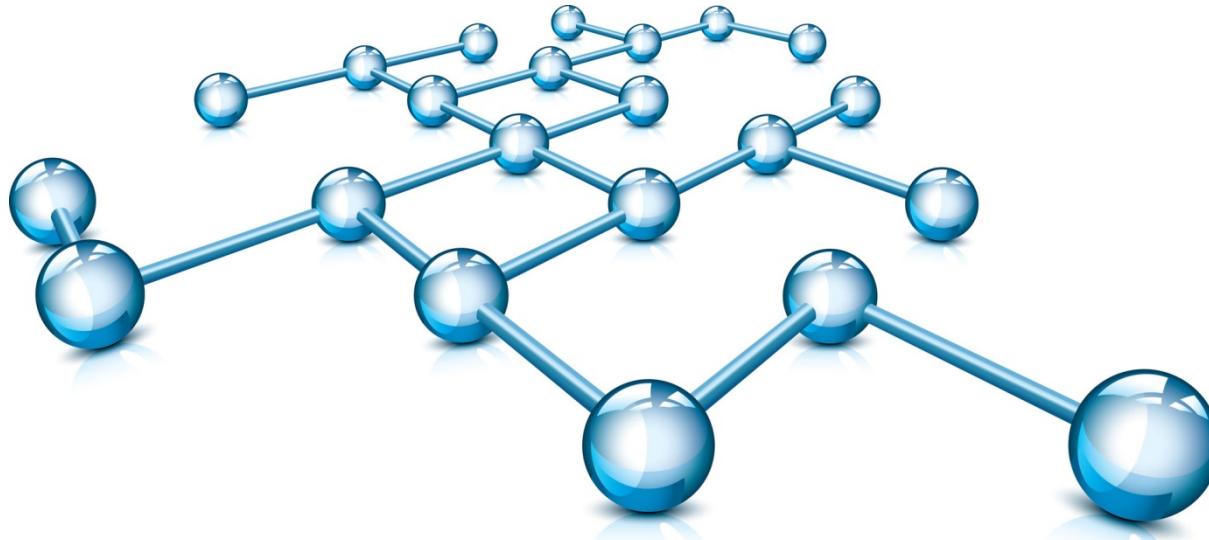


# Comunicación entre Procesos

Profesor:  
Dr. J. Octavio Gutiérrez García

[octavio.gutierrez@itam.mx](mailto:octavio.gutierrez@itam.mx)



# Comunicación entre Procesos

Aplicaciones, Servicios

Invocación remota, comunicación indirecta

Primitivas subyacentes de comunicación entre procesos:  
Sockets, paso de mensajes, empaquetado y  
representación de datos.

UDP y TCP

# Comunicación entre Procesos: Primitivas

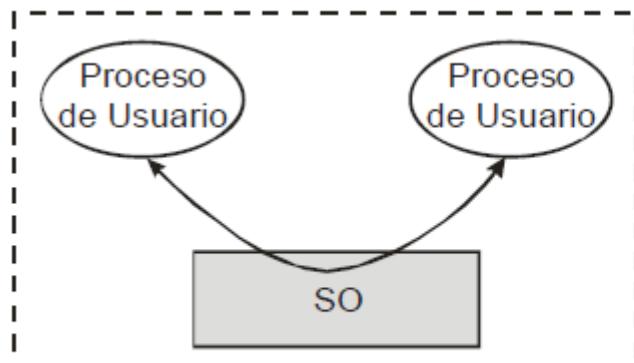


Emisor:

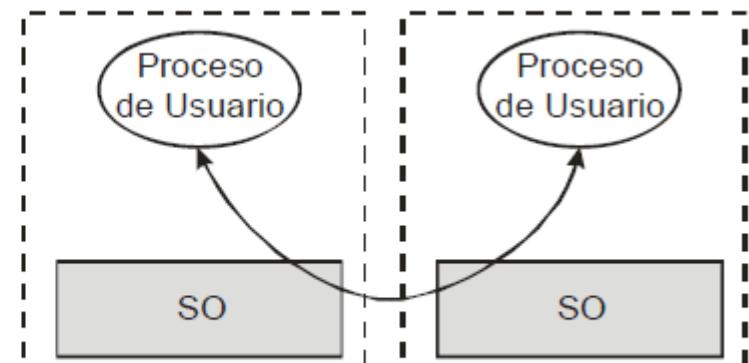
`send(destino, mensaje)`

Receptor:

`receive(origen, mensaje)`



Un computador



Dos computadores

# Clasificación de la comunicación

Síncrona



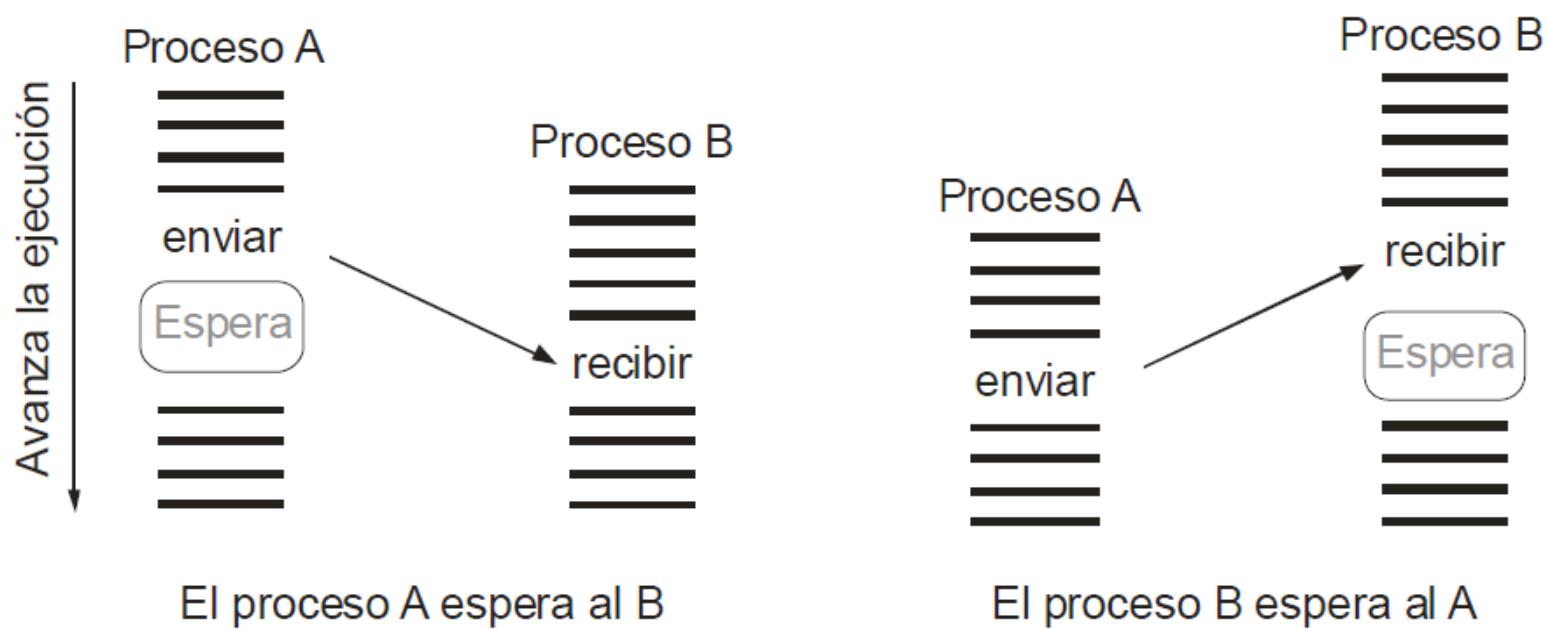
Asíncrona



# Clasificación de la comunicación

- Síncrona (**bloqueante**)

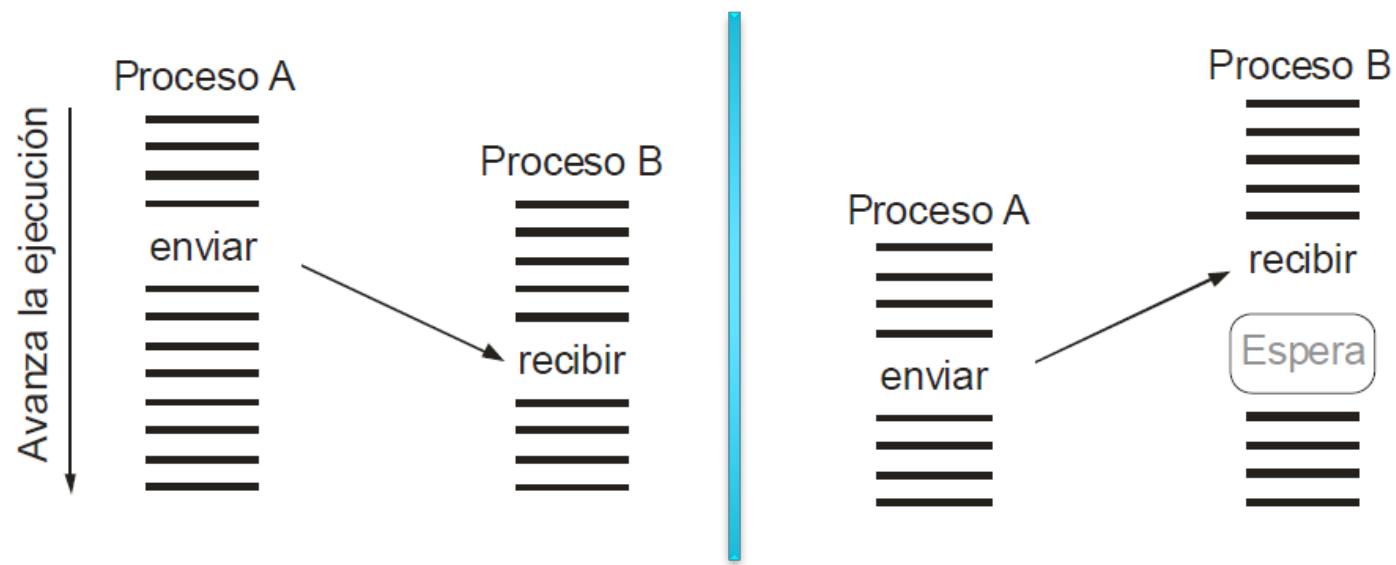
- Enviar() y Recibir() son operaciones de bloqueo



# Clasificación de la comunicación

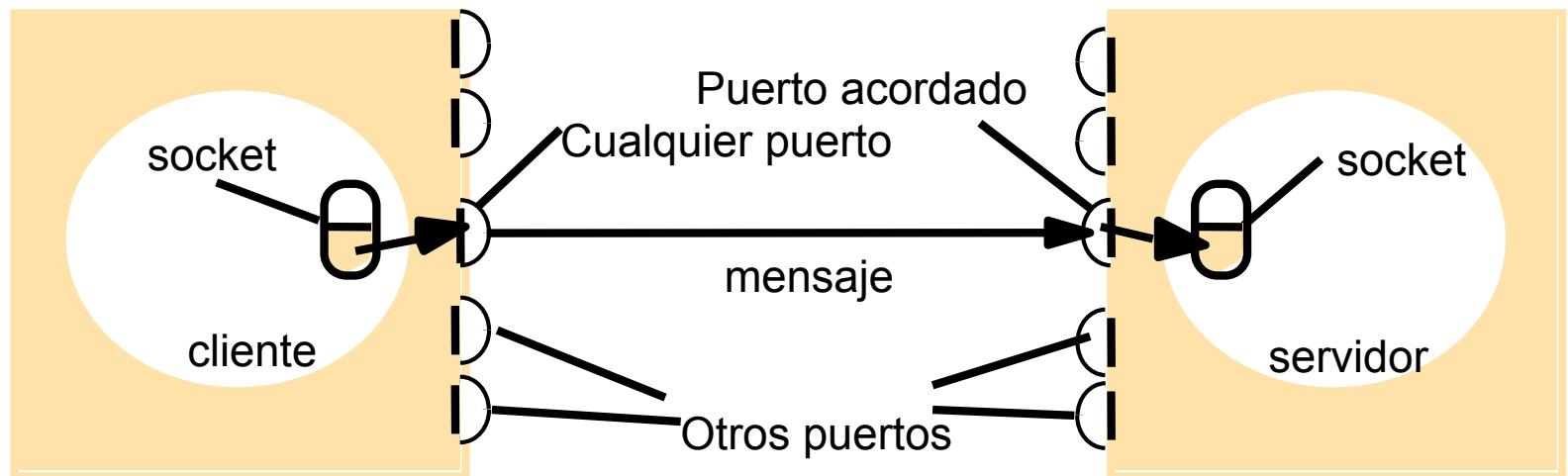
- **Asíncrona (No bloqueante)**

- Enviar() es no bloqueante
- Recibir() puede ser bloqueante o no bloqueante.



# Sockets

- Tanto UDP como TCP utilizan la abstracción de Sockets.

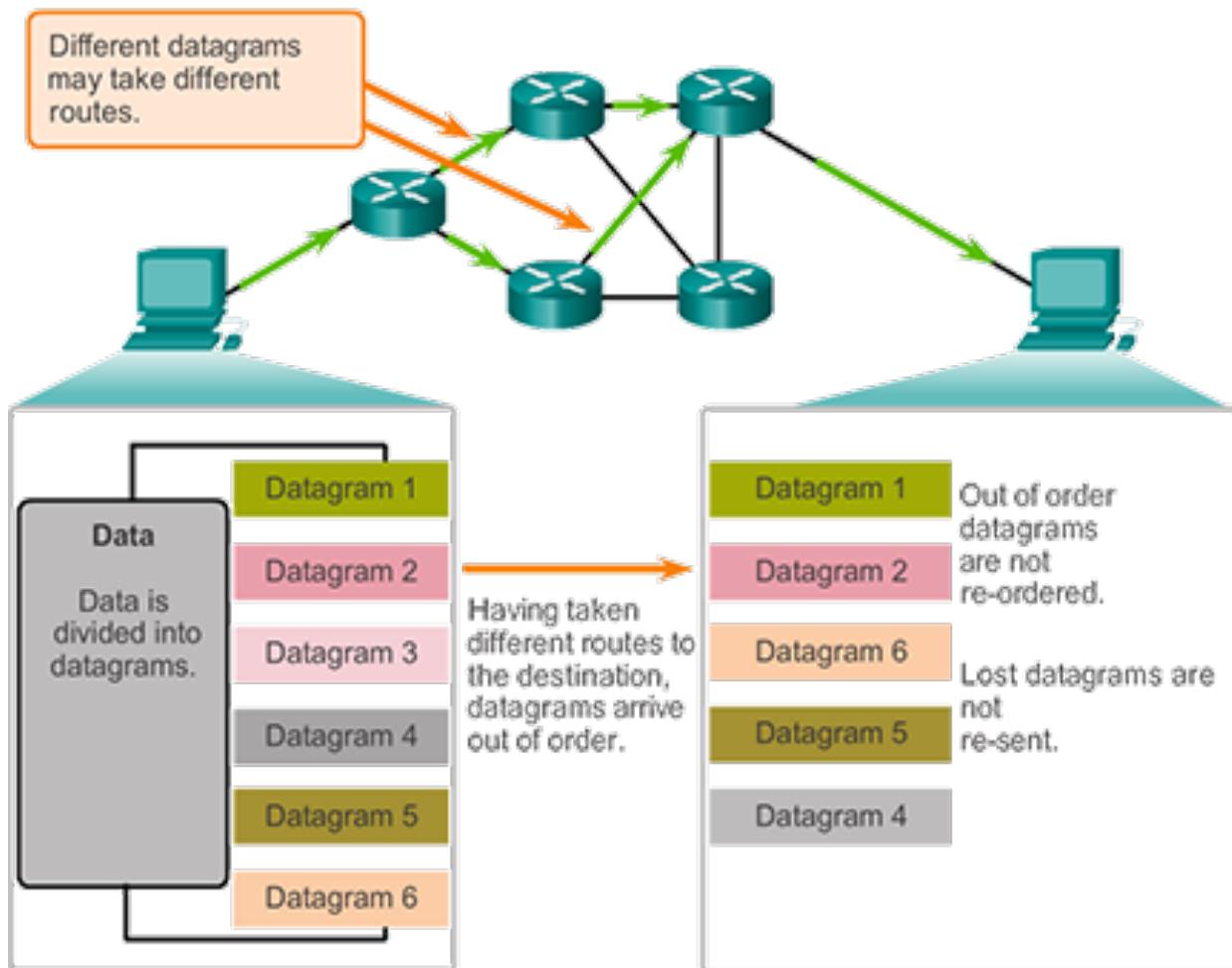


Dirección IP = 138.37.94.248

Dirección IP = 138.37.88.249

# User Datagram Protocol (UDP)

UDP: Connectionless and Unreliable



# Comunicación de datagramas UDP

- Cuestiones importantes para la comunicación de datagramas usando Sockets:
  - El proceso de recepción debe de especificar un arreglo de bytes (de un tamaño en particular) en el cual se recibirá el mensaje (<= 8 Kb)



JavaUDPSockets

# Comunicación UDP en Java

- Librerías

- `java.net.*;`
- `java.io.*;`

- Clases

- `DatagramSocket(serverPort)`
- `DatagramPacket(m, m.length, aHost, serverPort) // envío`
- `DatagramPacket(buffer, buffer.length) // recepción`
- `InetAddress`



JavaUDPSockets

# Comunicación de datagramas UDP

- Cuestiones importantes para la comunicación de datagramas usando Sockets:
  - Bloqueo
    - Enviar() no bloqueante
    - Recibir() bloqueante – opción otro Hilo.
  - Se puede definir un tiempo límite de espera (Timeouts)



# Comunicación de streams (flujos) TCP

- Orientado a conexión.
- La abstracción de streams oculta las siguientes características de la red:
  - La aplicación puede **elegir la cantidad** de datos que quiere **escribir** o **leer** del stream
  - Los mensajes perdidos son detectados vía **acuse de recibo** y son reenviados.

# Comunicación de streams (flujos) TCP

- La abstracción de streams oculta las siguientes características de la red:
  - Control de flujo. TPC **ajusta las velocidades** de los procesos que leen y escriben en un stream.
  - **Duplicación** de mensajes y **ordenamiento** es controlado vía identificadores incrustados en los mensajes.
  - Destino de los mensajes. Los procesos establecen una **conexión** para comunicarse mediante un **stream**.

# Comunicación TCP en Java

- Librerías
  - `java.net.*;`
  - `java.io.*;`
- Clases
  - `Thread`
  - `ServerSocket(serverPort)`
  - `Socket`
  - `Connection(Socket)`
  - `DataInputStream`
  - `DataOutputStream`



# Comunicación de streams (flujos) TCP

- Cuestiones importantes para la comunicación de streams TCP
  - Concordancia de elementos de datos escritos y leídos del stream.

# Comunicación de streams (flujos) TCP

- Cuestiones importantes para la comunicación de streams TCP
  - Bloqueo. **Datos escritos** en el stream se quedan en una **cola** en el Socket destino.
  - Cuando un proceso intenta leer, obtiene datos de la cola, si **no hay datos se bloquea** hasta que haya datos.
  - El proceso que **escribe puede ser bloqueado** (por el mecanismo de control del TCP) si el socket del otro lado tiene **llena la cola de entrada**.



# Comunicación de streams (flujos) TCP

- Cuestiones importantes para la comunicación de streams TCP
  - Los servidores crean **hilos** por cada **cliente**.

# Comunicación de streams (flujos) TCP

- Práctica de Laboratorio
  - Consulta un servidor agenda vía Sockets TCP
    - Recibe número id de persona
      - Si el id es válido
        - Busca en AddressBook
        - Regresa el nombre de la persona
      - Si no
        - Se apaga
  - Medir tiempo de respuesta promedio y desviación estándar para **n** clientes y **m** solicitudes por cliente
  - ¿Cuántos clientes soporta?



```
long startTime = System.currentTimeMillis();  
  
// ... instrucciones ...  
  
long spentTime = System.currentTimeMillis() - startTime;
```



JavaTCPListeners\_lab2

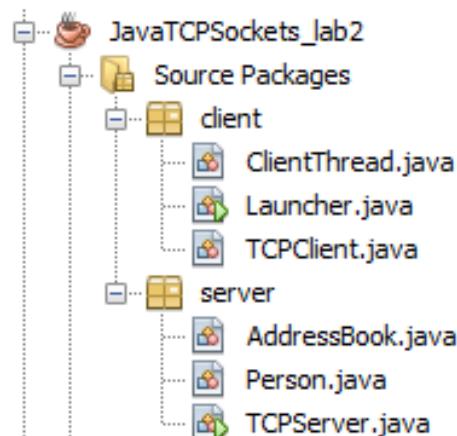
# Comunicación de streams (flujos) TCP

## ● Práctica de Laboratorio

Código en el servidor:

```
addressBook = new AddressBook();
```

```
addressBook.getRecord(key).getName()
```



Clave enviada  
por el cliente  
al servidor

El servidor regresa el  
nombre correspondiente  
a la clave

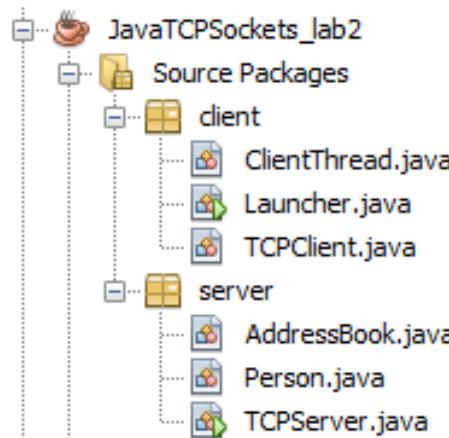


JavaTCPSockets\_lab2

# Comunicación de streams (flujos)

## TCP

- Implementar lanzador de clientes



```
public class Launcher {  
    public static void main (String args[]) {  
        ClientThread clientThread = new ClientThread();  
        clientThread.start();  
    }  
}  
  
class ClientThread extends Thread {  
    public ClientThread () {  
    }  
    public void run(){  
    }  
}
```

Debe  
Instanciar a  
TCPClient



JavaTCP.Sockets\_lab2

# Código adicional

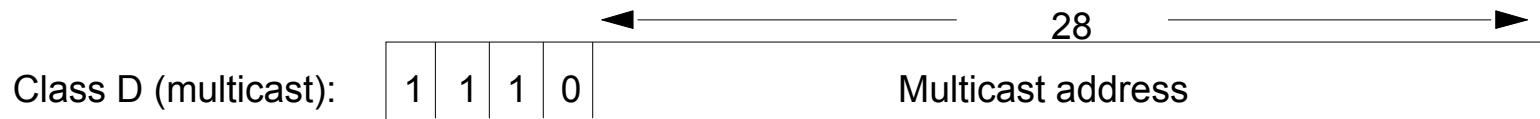
```
private double stdDev(long[] list){  
    double sum = 0.0;  
    double num = 0.0;  
  
    for (int i=0; i < list.length; i++)  
        sum+=list[i];  
  
    double mean = sum/list.length;  
  
    for (int i=0; i <list.length; i++)  
        num+=Math.pow((list[i] - mean),2);  
  
    return Math.sqrt(num/list.length);  
}
```

# Comunicación Multicast

- **Comunicación en grupo**, de un proceso a un grupo de procesos
- Utilidad:
  - **Tolerancia a fallos** en servicios replicados
  - Descubrimiento de servicios
  - **Mejor rendimiento** a través de datos replicados
  - Propagación de **notificaciones**

# Multicast IP

- Grupo multicast -> dirección multicast



De 224.0.0.0 a 239.255.255.255

- Grupos son **dinámicos**

# Multicast IP

- Sólo disponible vía UDP
- Una computadora se une a un grupo multicast cuando uno o más de sus procesos tiene sockets que pertenecen al grupo.
- Time to live (TTL). Número de routers que el mensaje puede pasar (0..255)
- Internet Assigned Numbers Authority (IANA)

# Multicast IP

- **Librerías**
  - `java.io.IOException`
  - `java.net.DatagramPacket`
  - `java.net.InetAddress`
  - `java.net.MulticastSocket`
  - `java.net.SocketException`
- **Clases**
  - `InetAddress`
  - `DatagramPacket(buffer, buffer.length) // recibir`
  - `DatagramPacket(buffer, buffer.length, group, 6789) // enviar`
  - `MulticastSocket`
    - `joinGroup`
    - `receive`
    - `send`
    - `leaveGroup`



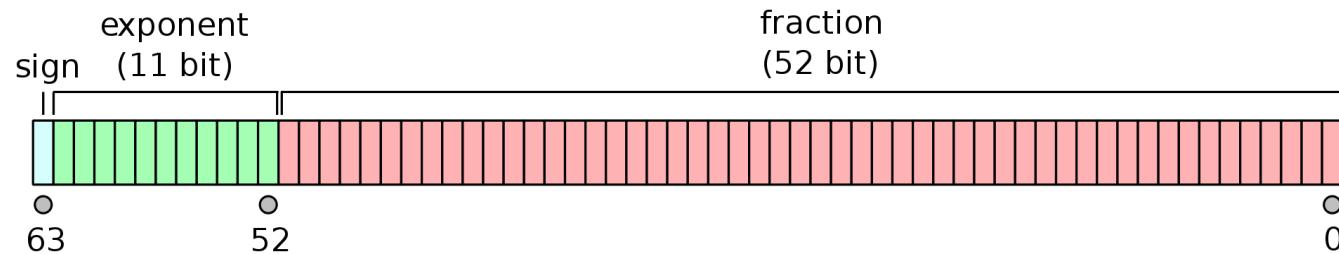
# Multicast IP

- **Práctica de Laboratorio**
  - Servidor “reloj”
    - Envía mensajes multicast a clientes conteniendo la hora actual.
    - Siempre está en ejecución
  - Cliente
    - Se inscribe al grupo multicast del “reloj” para actualizar su hora.
    - Recibe la actualización y termina su ejecución.
  - Tips
    - String currentDate = (new Date()).toString();
    - byte [] m = currentDate.getBytes();
    - Thread.sleep(2000);



# Representación externa de datos

- Datos en programas están en estructuras de datos.
- Transmisión y Recepción: Conversión a secuencia de bytes
- Diferentes dispositivos, diferentes formas de almacenar datos, ejemplo: números de punto flotante.



# Representación externa de datos

- Mecanismos para intercambiar datos entre computadoras de cualquier tipo.
- 1 Utilizar una **representación externa de datos** para su transmisión.
  - 2 Transmitir los datos en el **formato del emisor** e **indicar** que **formato** se utilizó.

# Empaqueamiento

- Proceso que toma una **colección** de elementos de **datos** y los **ensambla** de un modo adecuado para su **transmisión**.



# Representación externa de datos

- Representación común de datos de CORBA  
(Common Object Request Broker Architecture)
- Serialización de objetos en Java
- XML (Extensible Markup Language)
- Json (JavaScript Object Notation)



# Representación común de datos de CORBA

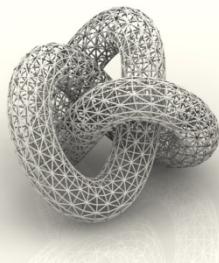
- **Tipos primitivos**

- short, long, unsigned short, unsigned long, float, double, char, boolean, octet, any.



- **Tipos compuestos**

- String, array, struct, enumerated, union, sequence



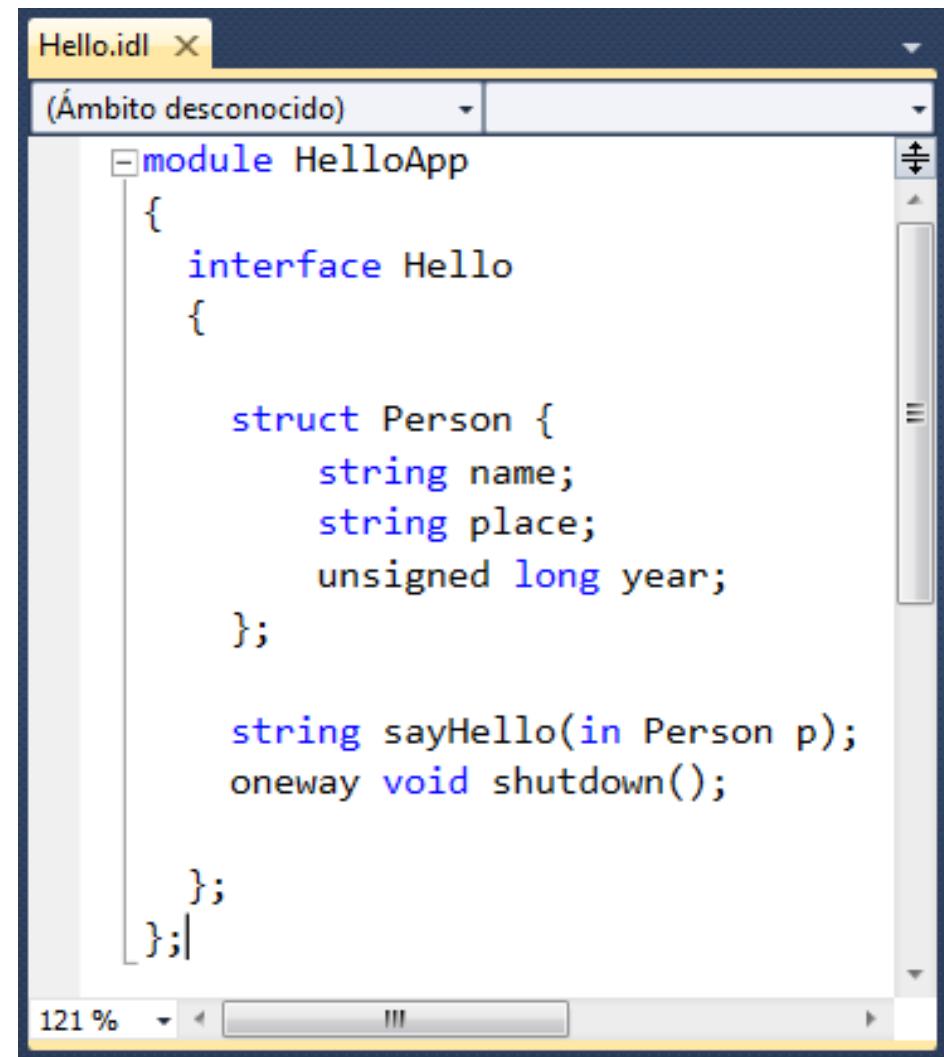
# Representación común de datos de CORBA

struct Person con valor: {'Smith', 'London', 1984}

Indice en Secuencia de bytes	4 bytes
0–3	5
4–7	"Smit"
8–11	"h "
12–15	6
16–19	"Lond"
20–23	"on "
24–27	1984

# Empaquetamiento en CORBA

- CORBA IDL  
Interface definition language



The screenshot shows a code editor window titled "Hello.idl X". The code defines a module named "HelloApp" containing an interface "Hello" with a struct "Person" and two operations: "sayHello" and "shutdown". The code is as follows:

```
module HelloApp
{
    interface Hello
    {

        struct Person {
            string name;
            string place;
            unsigned long year;
        };

        string sayHello(in Person p);
        oneway void shutdown();
    };
}
```

The code editor interface includes a status bar at the bottom showing "121 %".

# Representación externa de datos

- Representación común de datos de CORBA
- **Serialización de objetos en Java**
- XML (Extensible Markup Language)
- Json (JavaScript Object Notation)



# Serialización de objetos en Java

*Valores serializados*

Person	8-byte version number		h0
3	int year	java.lang.String name	java.lang.String place
1984	5 Smith	6 London	h1

```
public class Person {  
    private String name;  
    private String place;  
    private int year;
```

...

# Serialización de objetos en Java

```
import java.io.Serializable;  
  
public class Person implements Serializable {  
    private String name;  
    private String place;  
    private int year;  
    public Person(String aName, String aPlace, int aYear){  
        name = aName;  
        place = aPlace;  
        year = aYear;  
    }  
    // seguido de los métodos para acceder  
    // a los valores de los atributos  
}
```

# Serialización de objetos en Java

- Librería
  - `java.io.Serializable`
- Clases y métodos
  - implements `Serializable`
  - `DataInputStream`   `ObjectInputStream`
  - `DataOutputStream`   `ObjectOutputStream`
  - `writeUTF`   `writeObject`
  - `readUTF`   `readObject`



# Serialización de objetos en Java

## • Práctica de Laboratorio

- Enviar y recibir objetos (Person) vía Sockets TCP (Utilizar proyecto JavaTCPSockets)
- Lectura y escritura de objetos en Streams
  - Usar flujos **ObjectOutputStream**
  - Person me = new Person("Octavio", "Jalisco", 1980);
  - out.writeObject(me);
  - me = (Person) in.readObject();



# Representación externa de datos

- Representación común de datos de CORBA
- Serialización de objetos en Java
- XML (Extensible Markup Language)
- Json (JavaScript Object Notation)



# XML (Extensible Markup Language)

- Especificación/**lenguaje** diseñado especialmente para documentos web.
  - **Datos**
  - **Estructura** de los datos
- Los diseñadores crean sus **propias etiquetas**, permitiendo la definición, transmisión, validación e interpretación de datos entre aplicaciones y entre organizaciones.

# XML: Elementos y atributos

```
<person id="123456789">
    <name>Smith</name>
    <place>London</place>
    <year>1984</year>
    <!-- a comment -->
</person>
```

# XML: Elementos y atributos

```
<person id="123456789">
```

```
    <name>Smith</name>
```

```
    <place>London</place>
```

```
    <year>1984</year>
```

```
    <!-- a comment -->
```

```
</person>
```

# Conflictos en los nombres

- Archivo XML con información de una **tabla HTML**

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

- Archivo XML con información **de mesas**

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

# XML Namespace

Un archivo XML puede contener elementos o atributos de más de un vocabulario XML. El uso de Namespace permite **resolver la ambigüedad**.

```
<pers:person pers:id="123456789" xmlns:pers = "http://any.url.com/person">
    <pers:name> Smith </pers:name>
    <pers:place> London </pers:place >
    <pers:year> 1984 </pers:year>
</pers:person>
```

# XML Namespace

```
<pers:person pers:id="123456789" xmlns:pers = "http://any.url.com/person">
    <pers:name> Smith </pers:name>
    <pers:place> London </pers:place >
    <pers:year> 1984 </pers:year>
</pers:person>
```

# XML Esquemas

Un esquema XML (.xsd) describe la **estructura** de un archivo XML

```
<xs:schema xmlns:xs = "http://any.url.com/MyXMLSchema">

    <xs:element name= "person" type ="personType" />

    <xs:complexType name="personType">
        <xs:sequence>
            <xs:element name = "name" type="xs:string"/>
            <xs:element name = "place" type="xs:string"/>
            <xs:element name = "year" type="xs:positiveInteger"/>
        </xs:sequence>
        <xs:attribute name= "id" type = "xs:positiveInteger"/>
    </xs:complexType>
</xs:schema>
```

# Esquemas: Restricciones en Valores

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer"> ←
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Esquemas: Restricciones en Valores

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string"> ←
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Esquemas: Restricciones en Valores

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]" /> ←
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]" /> ←
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Esquemas: Restricciones en Valores

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[xyz]" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Esquemas: Restricciones en Valores

```
<xs:element name="prodid">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/> ←
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/> ←
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Esquemas: Restricciones en Valores

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}" /> ←
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Esquemas: Restricciones en Valores

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/> ←
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/> ←
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Esquemas: Restricciones en Valores

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

\* = 0 ó +

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])*/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

"sToP"

+ = | ó +

# Hacer referencia a un Esquema XML

- <nt:note xmlns:nt= "http://any.url.com"  
  xmlns:xsi="http://another.arbitrary.url">  
    xsi:schemaLocation="http://any.url.com note.xsd">  
      <nt:to>Juan</nt:to>  
      <nt:from>Pedro</nt:from>  
      <nt:priority>Alta</nt:priority>  
      <nt:date>2015-03-16 <nt:date>  
      <nt:heading>Recordatorio</nt:heading>  
      <nt:body>¡Estudiar para el examen de SD!</nt:body>  
    </nt:note>

## XML Types

<http://www.w3.org/TR/xmlschema-0/#simpleTypesTable>

# Práctica de Laboratorio

- <nt:note xmlns:nt= "http://any.url.com"  
  xmlns:xsi="http://another.arbitrary.url">  
    xsi:schemaLocation="http://any.url.com note.xsd">  
      <nt:to>Juan</nt:to>  
      <nt:from>Pedro</nt:from>  
      <nt:priority>Alta</nt:priority>  
      <nt:date>2015-03-16 <nt:date>  
      <nt:heading>Recordatorio</nt:heading>  
      <nt:body>¡Estudiar para el examen de SD!</nt:body>  
    </nt:note>

XML Types

¿Cuál es el esquema  
XML asociado?

note.xsd

<http://www.w3.org/TR/xmlschema-0/#simpleTypesTable>

# Práctica de Laboratorio

- **To** y **from** sólo deben de aceptar letras minúsculas y mayúsculas. La longitud debe ser 1 ó más
- Valores válidos de **Priority** son “Alta”, “Baja” e “Intermedia”
- **Heading** sólo deben de aceptar letras minúsculas y mayúsculas y números. La longitud puede ser 0 ó más.

# Práctica de Laboratorio

- Al final se deberá tener:
  - Un solo elemento: La nota
  - Un tipo complejo: El tipo complejo de la Nota
  - Tres tipos simples: uno para **To** y **From**; otro para **Priority** y uno más para **Heading**

# Representación externa de datos

- Representación común de datos de CORBA
- Serialización de objetos en Java
- XML (Extensible Markup Language)
- Json (JavaScript Object Notation)



# Json (JavaScript Object Notation)

```
<html>
<body>
<h2>Creación de un Objeto Json en JavaScript</h2>
<p>
    Name: <span id="jname">    </span> <br/>
    Place: <span id="jplace">   </span> <br/>
    Year: <span id="jyear">    </span> <br/>
</p>
<script>
    var JSONObject= {
        name:"Smith",
        place:"London",
        year:1984};
    document.getElementById("jname").innerHTML=JSONObject.name
    document.getElementById("jplace").innerHTML=JSONObject.place
    document.getElementById("jyear").innerHTML=JSONObject.year
</script>
</body>
</html>
```



*...to be continued*