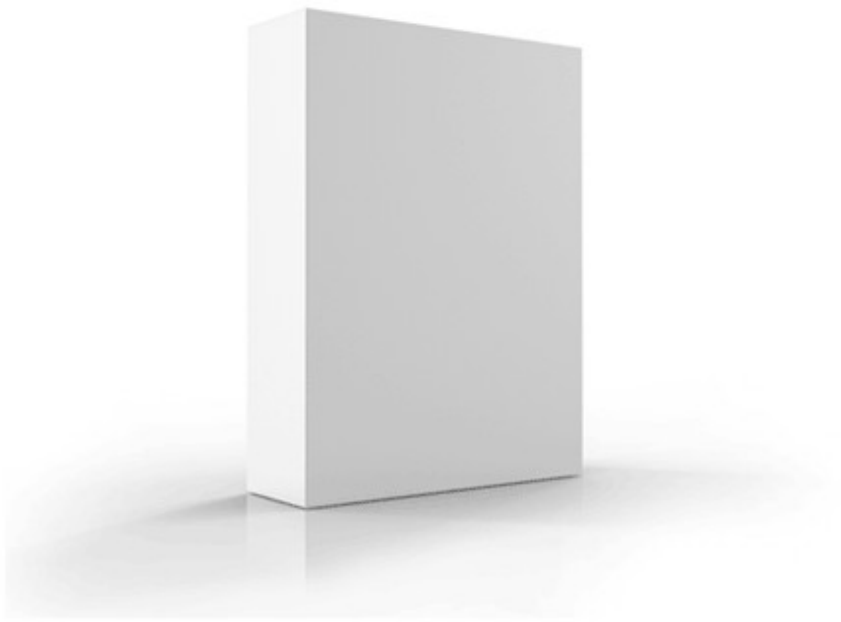
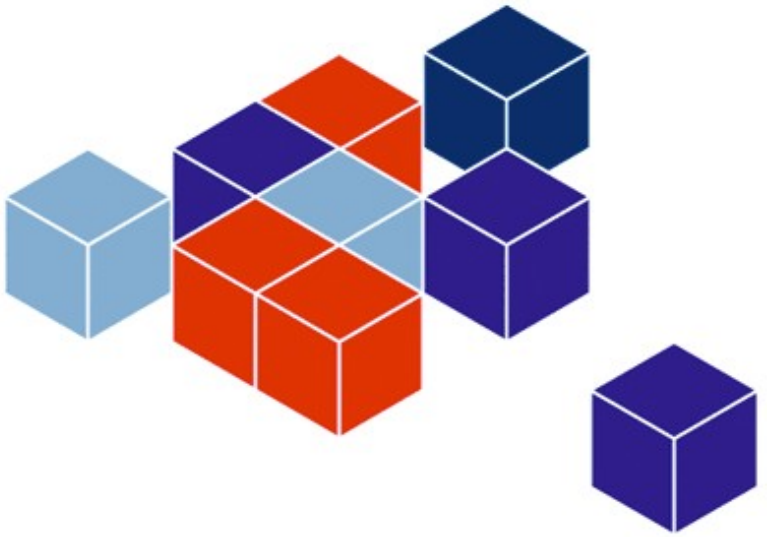
Comunicación entre procesos y objetos distribuidos

Germán E. Lescano
gelescano@unse.edu.ar | german.lescano@gmail.com

Aplicación Monolítica vs Aplicación Distribuida





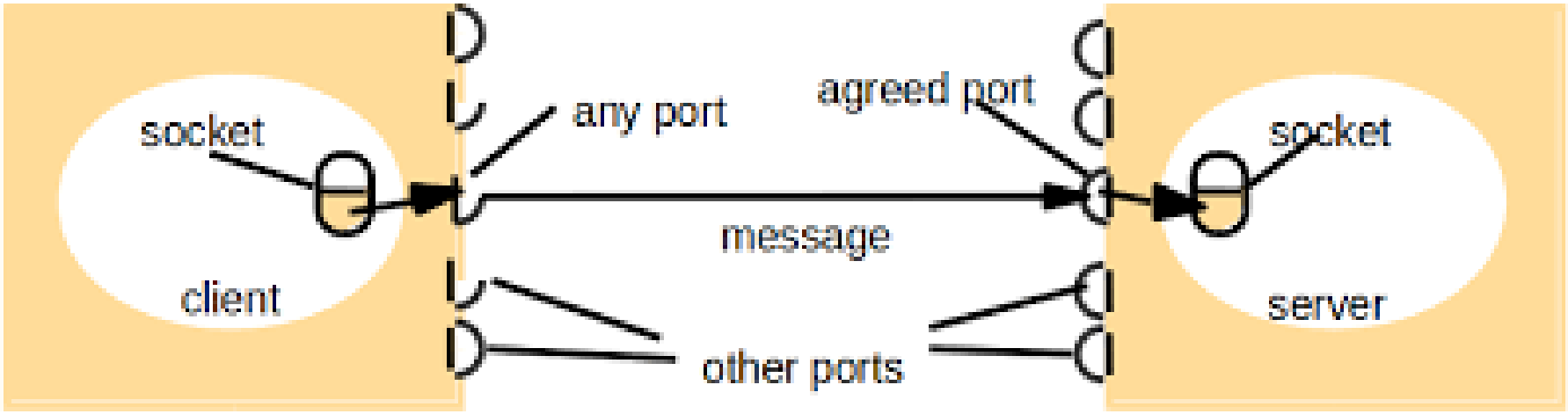
Comunicación entre procesos

Tipo de comunicación

- Comunicación cliente-servidor
- Comunicación en grupo

Protocolo empleado para la comunicación:

- UDP
- TCP



Comunicación Sincrónica y Asincrónica

Cantidad de puertos 2^{16}

Características de la comunicación empleando UDP

- **Tamaño de mensaje de longitud fija (2^{16}, aunque mayormente se impone un tamaño de 8kb).**
- **El receptor es el único que se bloquea.**
- **Existe tiempo límite de espera.**
- **El proceso receptor puede recibir de cualquier emisor.**
- **Tipos de fallos:**
  - Fallos de omisión
  - Fallos de ordenación

Características de la comunicación empleando TCP

- **Las aplicaciones deciden la cantidad de datos que desean leer o escribir.**
- **Control de mensajes perdidos.**
- **Control de flujo.**
- **Control de duplicación.**
- **Ordenación de mensajes.**
- **Cuando hay congestión severa no proporciona comunicación fiable.**

Clases Java involucradas con la comunicación UDP

- **DatagramPacket. Permite crear una instancia compuesta por una cadena de bytes que almacena el mensaje, la longitud del mensaje, la dirección de internet y el número de puerto local del conector destino.**
- **DatagramSocket. Maneja conectores para enviar y recibir datagramas UDP.**

Principales sentencias involucradas en un Servidor (comunicación UDP)

```
// Creación de un socket para recibir y enviar mensajes
DatagramSocket socketUDP = new DatagramSocket(3584);

// Creación de un buffer
byte[] bufer = new byte[1000];

// Creación un objeto datagrama para recibir mensaje enviado por un cliente
DatagramPacket datagrama = new DatagramPacket(bufer, bufer.length);

// recepción de datagrama del cliente
socketUDP.receive(datagrama);
```

Principales sentencias involucradas en un Cliente (comunicación UDP)

```
// Creación de un socket para enviar y recibir mensajes
DatagramSocket socketUDP = new DatagramSocket();

// Creación de un objeto que representa la dirección del servidor
InetAddress servidor = InetAddress.getByName("192.168.122.3");

// Creación del datagrama a enviar
DatagramPacket datagrama = new DatagramPacket(bytesEntrada,
bytesEntrada.length, servidor, 3584);

// Envio del datagrama
socketUDP.send(datagrama);
```

Clases Java involucradas con la comunicación TCP

- **ServerSocket. A través del método accept() recibe conexiones de clientes. En caso de no haber solicitudes se bloquea. Al aceptar una conexión se crea una instancia de Socket, un conector que da acceso a streams para comunicarse con el cliente.**
- **Socket. Es la clase utilizada por el par de procesos de una conexión. Tiene dos métodos importantes:**
  - **getInputStream()**
  - **getOutputStream()**

Estructura de un mensaje petición-respuesta

tipoMensaje	int (0 = petición, 1 = respuesta)
idPetición	int
referenciaObjeto	RemoteObjectRef
idMétodo	int o método
Argumento	cadena de bytes

Implementación de hilos en Java

Heredando de la clase java.lang.Thread

- **Java.lang.Thread implementa la clase pública Runnable**
- **La clase que hereda de Thread debe implementar el método run()**
- **Invocación:**

```
ClockThread reloj = new ClockThread();
reloj.start();
```

Implementación de hilos en Java

Implementando la interfaz Runnable

- **Para implementar la interfaz Runnable es necesario definir el método run()**
- **Invocación:**

```
ClockRunnable reloj = new ClockRunnable();
Thread tr = new Thread(reloj);
tr.start();
```

Sincronización entre hilos

- **Se emplean los métodos wait(), notify() y notifyAll().**
- **Estos métodos son definidos por la clase java.lang.Object por lo cual son heredados por todas las clases.**
- **El método wait() bloquea la ejecución de un hilo hasta que se cumpla una condición. El cumplimiento de la condición es señalado por otro hilo mediante los métodos notify() o notifyAll().**
- **Cuando el método wait() se invoca desde un método synchronized, al mismo tiempo que la ejecución del hilo es pausada, se libera el bloqueo que el hilo tiene sobre el método. Tan pronto como la condición se cumpla, el hilo es reanudado y debe esperar para tomar nuevamente la ejecución exclusiva del método.**

Sincronización entre hilos

```
public class AlumnoComAsincronica extends Thread {

    int repeticiones;
    String nombre, palabraAAprender;
    BufferPizarronComAsincronica pizarron;

    public AlumnoComAsincronica(String nomAlumno, int cantRepeticiones,
            BufferPizarronComAsincronica p) {
        nombre = nomAlumno;
        repeticiones = cantRepeticiones;
        pizarron = p;
    }

    public void repetir() {
        for (int i = 0; i < repeticiones; i++) {
            System.out.println(nombre + " ... " + palabraAAprender);
            try {
                Thread.sleep(4000);
            } catch (InterruptedException ex) {
                Logger.getLogger(AlumnoComAsincronica.class.getName()).
                        log(Level.SEVERE, null, ex);
            }
        }
    }

    public void run() {
        while (true) {
            palabraAAprender = pizarron.AprenderPalabra();
            repetir();
        }
    }
}
```

Sincronización entre hilos

```
public class ProfesorComAsincronica extends Thread {
    String palabraEnseniar;
    BufferPizarronComAsincronica pizarron;

    public ProfesorComAsincronica(BufferPizarronComAsincronica p){
        pizarron = p;
    }

    public void run(){
        pizarron.EnseniarPalabre("pizza");
        pizarron.EnseniarPalabre("perro");
        pizarron.EnseniarPalabre("jabon");
        pizarron.EnseniarPalabre("casa");
        pizarron.EnseniarPalabre("licuado");
        pizarron.EnseniarPalabre("pera");
    }
}
```

Sincronización entre hilos

```
public class BufferPizarronComAsincronica {

    Queue<String> colaPalabras;
    int limiteCola;
    int cantCola;

    public BufferPizarronComAsincronica() {
        colaPalabras = new LinkedList<>();
        limiteCola = 4;
        cantCola = 0;
    }

    public synchronized void EnseniarPalabre(String nPalabra) {
        if (cantCola >= limiteCola) {
            try {
                System.out.println("Esperando por disponibilidad de cola");
                wait();
            } catch (InterruptedException ex) {
                Logger.getLogger(BufferPizarronComAsincronica.class.
                        getName()).log(Level.SEVERE, null, ex);
            }
        }

        cantCola += 1;
        colaPalabras.add(nPalabra);
        notifyAll();

    }
```

Sincronización entre hilos

Continuación de la clase BufferPizarronComAsincrónica

```
    public synchronized String AprenderPalabra() {
        if (cantCola == 0) {
            try {
                wait();
            } catch (InterruptedException ex) {
                Logger.getLogger(BufferPizarronComAsincronica.class.
                        getName()).log(Level.SEVERE, null, ex);
            }
        }
        cantCola -= 1;
        notifyAll();
        return colaPalabras.remove();
    }
}
```