

Ejemplos: Comunicación entre Procesos

`java.io.*; java.net.*; tuberías...`

Ejemplos que vamos a ver

Vamos a ver cómo implementar dos forma de comunicación sencilla entre procesos en la misma máquina:

1. Utilizando **sockets**. Un socket crea un canal de comunicación directo entre dos procesos.
2. Gracias al sistema operativo, utilizando tuberías para la **redirección de la salida y entrada estándar** de los procesos.

Además de unas recomendaciones para probar los ejemplos.

Ejemplo 1:

Uso básico de sockets

- Un **socket** permite establecer **un canal de comunicación entre dos procesos a través de la red.**
 - Necesitaremos la **IP** y un **número de puerto** de una de las máquinas que está ejecutando uno de los procesos, para iniciar la creación del canal de comunicación.
- También podemos utilizar sockets para comunicar procesos **dentro de la misma máquina.**
 - localhost o 127.0.0.1; y un número de puerto disponible.

Componentes del ejemplo de sockets:

1. Proceso **Escritor** en Socket:
 - A. Iniciaré la creación del canal de comunicación.
 - B. Esperaré que el lector se conecte al canal.
 - C. Mandaré el mensaje al proceso lector.
2. Proceso **Lector** en Socket:
 - A. Se conectará al canal de comunicación.
 - B. Quedará a la espera de los datos recibidos por el canal.
 - C. Mostrará la información recibida en pantalla.

Ejemplo 1: Uso básico de sockets

Proceso EscritorSocket

Proceso **Escritor** en Socket:

- Import necesarios:
 - Para el uso de sockets → `import java.net.*;`
Utilizaremos las clases `ServerSocket` y `Socket`.
 - Para el uso de streams → `import java.io.*;`
Utilizaremos las clases `PrintWriter` e `IOException`.

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

A. Iniciar la creación del canal de comunicación.

```
ServerSocket conexion = null; //Socket para aceptar conexiones  
try{  
    conexion = new ServerSocket( 12345 );  
    //Solicitamos al sistema operativo que abra un puerto de escucha  
    //de conexiones. El número del puerto es el 12345  
}catch(IOException ex){  
    System.err.println("No se ha podido abrir el puerto de escucha.");  
    System.err.println(ex.toString());  
}
```

Estamos pidiendo al sistema operativo si podemos escuchar conexiones de otros procesos por el puerto número 12345. Si no lo está utilizando otro proceso, nos lo concederá.

Recordemos que es recomendable que utilicemos los puertos entre 6000 y 65535.

B. Esperar la conexión del lector al canal.

C. Mandar el mensaje al proceso lector.

Ejemplo 1: Uso básico de sockets

Proceso EscritorSocket

Proceso **Escritor** en Socket:

B. Esperar la conexión del lector al canal.

```
try{
    System.out.println("Proceso escritor, esperando "+
        "la conexión del proceso lector....");
    canal = conexion.accept();
    //Esperamos hasta que se produzca una conexión al puerto
    //El método ServerSocket.accept(); bloquea (hace dormir)
    //el proceso hasta que se produce una conexión
}catch(Exception ex){
    System.err.println("No se ha podido establecer conexión, " +
        "o no ha ocurrido un fallo al escribir en el canal.");
    System.err.print(ex.toString());
}finally{
    //Nos aseguramos de que se cierren los recursos
    //que estamos utilizando
    if (canal != null) //Socket
        try{
            canal.close();
        }catch (IOException ex) {
            System.err.println("Error al cerrar el socket.");
            System.err.print(ex.toString());
        }
    if (conexion != null) //ServerSocket
        try{
            conexion.close();
        }catch (IOException ex) {
            System.err.println("Error al cerrar ServerSocket.");
            System.err.print(ex.toString());
        }
}
```

Ejemplo 1: Uso básico de sockets

Proceso EscritorSocket

Proceso **Escritor** en Socket:

C. Mandar el mensaje al proceso lector.

```
try{
    streamSalida = new PrintWriter(canal.getOutputStream());
    //Creamos un objeto PrintWriter a partir del Stream de salida
    //del socket o canal de comunicación
    //El objeto PrintWriter, nos permitirá utilizar los métodos
    //print y write para mandar datos al proceso que está
    //escuchando al otro lado del canal.
    System.out.println("Conexión establecida, mandando datos "+
        "al proceso lector...");
    for (int i=0; i<10; i++){
        streamSalida.println(i); //Mandamos del 0 al 9
        streamSalida.flush(); //Forzamos que mande cada número
    }
    System.out.println("Comunicación finalizada.");
}catch(Exception ex){
    System.err.print(ex.toString());
}finally{
    //Nos aseguramos de que se cierren los recursos
    //que estamos utilizando
    if (streamSalida != null)//PrintWriter
        streamSalida.close(); //su cierre no genera excepciones
    if (canal != null) //Socket
        try{
            canal.close();
        }catch (IOException ex) {
            System.err.println("Error al cerrar el socket.");
            System.err.print(ex.toString());
        }
    if (conexion != null) //ServerSocket
        try{
            conexion.close();
        }catch (IOException ex) {
            System.err.println("Error al cerrar ServerSocket.");
            System.err.print(ex.toString());
        }
}
```

Ejemplo 1: Uso básico de sockets

Proceso EscritorSocket

Proceso **Escritor** en Socket (código completo):

```
package socketescritor;

import java.net.*;
import java.io.*;
/**...*/
public class Main {
    /**...*/
    public static void main(String[] args) {
        ServerSocket conexion = null; //Socket para aceptar conexiones
        Socket canal = null; //Socket para establecer canal de comunicación
        PrintWriter streamSalida = null;
        try{
            conexion = new ServerSocket( 12345 );
            //Solicitamos al sistema operativo que abra un puerto de escucha
            //de conexiones. El número del puerto es el 12345
        }catch(IOException ex){
            System.err.println("No se ha podido abrir el puerto de escucha.");
            System.err.println(ex.toString());
        }
        if (conexion != null) //Si hemos podido abrir el puerto
            try{
                System.out.println("Proceso escritor, esperando "+
                    "la conexión del proceso lector....");
                canal = conexion.accept();
                //Esperamos hasta que se produzca una conexión al puerto
                //El método ServerSocket.accept(); bloquea (hace dormir)
                //el proceso hasta que se produce una conexión
                streamSalida = new PrintWriter(canal.getOutputStream());
                //Creamos un objeto PrintWriter a partir del Stream de salida
                //del socket o canal de comunicación
                //El objeto PrintWriter, nos permitirá utilizar los métodos
                //print y write para mandar datos al proceso que está
                //escuchando al otro lado del canal.
            }
```

Ejemplo 1: Uso básico de sockets

Proceso EscritorSocket

Proceso **Escritor** en Socket (código completo):

```
        System.out.println("Conexión establecida, mandando datos "+
            "al proceso lector....");
        for (int i=0; i<10; i++){
            streamSalida.println(i); //Mandamos del 0 al 9
            streamSalida.flush(); //Forzamos que mande cada número
        }
        System.out.println("Comunicación finalizada.");
    }catch(Exception ex){
        System.err.println("No se ha podido establecer conexión, " +
            "o no ha ocurrido un fallo al escribir en el canal.");
        System.err.print(ex.toString());
    }finally{
        //Nos aseguramos de que se cierren los recursos
        //que estamos utilizando
        if (streamSalida != null)//PrintWriter
            streamSalida.close(); //su cierre no genera excepciones
        if (canal != null) //Socket
            try{
                canal.close();
            }catch (IOException ex) {
                System.err.println("Error al cerrar el socket.");
                System.err.print(ex.toString());
            }
        if (conexion != null) //ServerSocket
            try{
                conexion.close();
            }catch (IOException ex) {
                System.err.println("Error al cerrar ServerSocket.");
                System.err.print(ex.toString());
            }
    }
}
```


Ejemplo 1: Uso básico de sockets

Proceso LectorSocket

Proceso **Lector** en Socket:

- Import necesarios:
 - Para el uso de sockets → `import java.net.*;`
Utilizaremos las clases `ServerSocket` y `Socket`.
 - Para el uso de streams → `import java.io.*;`
Utilizaremos las clases `PrintWriter` e `IOException`.

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

A. Conexión al canal de comunicación.

```
Socket canal = null; //Socket para establecer el canal de conexión con el escritor  
try{  
    canal = new Socket("localhost", 12345 );  
    //Pedimos establecer una conexión en el equipo local con el puerto 12345  
    //donde debe estar escuchando el proceso escritor  
}catch (Exception ex){  
    System.err.println("No se ha podido establecer conexión.");  
    System.err.println(ex.toString());  
}
```

Estamos pidiendo al sistema operativo si podemos conectarnos a otro proceso que debe estar escuchando por el puerto número 12345. Si no está escuchando ningún proceso, nos contestará que la conexión ha sido rehusada. Recordemos que el número de puerto tiene que coincidir con el número de puerto utilizado por el proceso Escritor.

B. Esperar recibir datos por el canal.

C. Mostrar información recibida en pantalla (o tratar esa información).

Ejemplo 1: Uso básico de sockets

Proceso LectorSocket

Proceso **Lector** en Socket:

B. Esperar recibir datos por el canal.

```
try{
    entrada = new BufferedReader(new InputStreamReader(canal.getInputStream()));
    //obtenemos el objeto que representa el stream de entrada en el canal
    //Lector con buffer, para no perder ningún dato
    while ((valorEntrada = entrada.readLine()) != null){
        //Mientras que haya datos que leer
        System.out.println(valorEntrada);
        System.out.println("***");
    }
}catch (Exception ex){
    System.err.println("No se ha podido establecer conexión.");
    System.err.println(ex.toString());
}finally{
    //Nos aseguramos de que se cierran los recursos que estamos utilizando
    if (entrada != null)
        try{
            entrada.close();
        }catch (IOException ex) {
            System.err.println("Se ha producido un error al cerrar el InputStreamReader.");
            System.err.println(ex.toString());
        }
    if (canal != null)
        try{
            canal.close();
        }catch (IOException ex) {
            System.err.println("Se ha producido un error al cerrar el Socket.");
            System.err.println(ex.toString());
        }
}
```

Ejemplo 1: Uso básico de sockets

Proceso LectorSocket

Proceso **Lector** en Socket:

C. Mostrar información recibida en pantalla (o tratar esa información).

```
try{
    entrada = new BufferedReader(new InputStreamReader(canal.getInputStream()));
    //Obtemenos el objeto que representa el stream de entrada en el canal
    //Lector con buffer, para no perder ningún dato
    while ((valorEntrada = entrada.readLine()) != null){
        //Mientras que haya datos que leer
        System.out.println(valorEntrada);
        System.out.println("**");
    }
}catch (Exception ex){
    System.err.println("No se ha podido establecer conexión.");
    System.err.println(ex.toString());
}finally{
    //Nos aseguramos de que se cierran los recursos que estamos utilizando
    if (entrada != null)
        try{
            entrada.close();
        }catch (IOException ex) {
            System.err.println("Se ha producido un error al cerrar el InputStreamReader.");
            System.err.println(ex.toString());
        }
    if (canal != null)
        try{
            canal.close();
        }catch (IOException ex) {
            System.err.println("Se ha producido un error al cerrar el Socket.");
            System.err.println(ex.toString());
        }
}
```

Ejemplo 1: Uso básico de sockets

Proceso LectorSocket

Proceso **Lector** en Socket (código completo):

```
package socketlector;

import java.net.*;
import java.io.*;
/**...*/
public class Main {
    /**...*/
    public static void main(String[] args) {
        // TODO code application logic here
        Socket canal = null; //Socket para establecer el canal de conexión con el escritor
        BufferedReader entrada = null; //Para el stream de lectura
        String valorEntrada = null; //Valores que iremos leyendo del canal
        try{
            canal = new Socket("localhost", 12345 );
            //Pedimos establecer una conexión en el equipo local con el puerto 12345
            //donde debe estar escuchando el proceso escritor
        }catch (Exception ex){
            System.err.println("No se ha podido establecer conexión.");
            System.err.println(ex.toString());
        }
        if (canal != null) //Si hemos podido establecer la conexión. Tenemos
            //un canal de comunicación
            try{
                entrada = new BufferedReader(new InputStreamReader(canal.getInputStream()));
                //Obtemos el objeto que representa el stream de entrada en el canal
                //Lector con buffer, para no perder ningún dato
                while ((valorEntrada = entrada.readLine()) != null){
                    //Mientras que haya datos que leer
                    System.out.println(valorEntrada);
                    System.out.println("***");
                }
            }
```

Ejemplo 1: Uso básico de sockets

Proceso LectorSocket

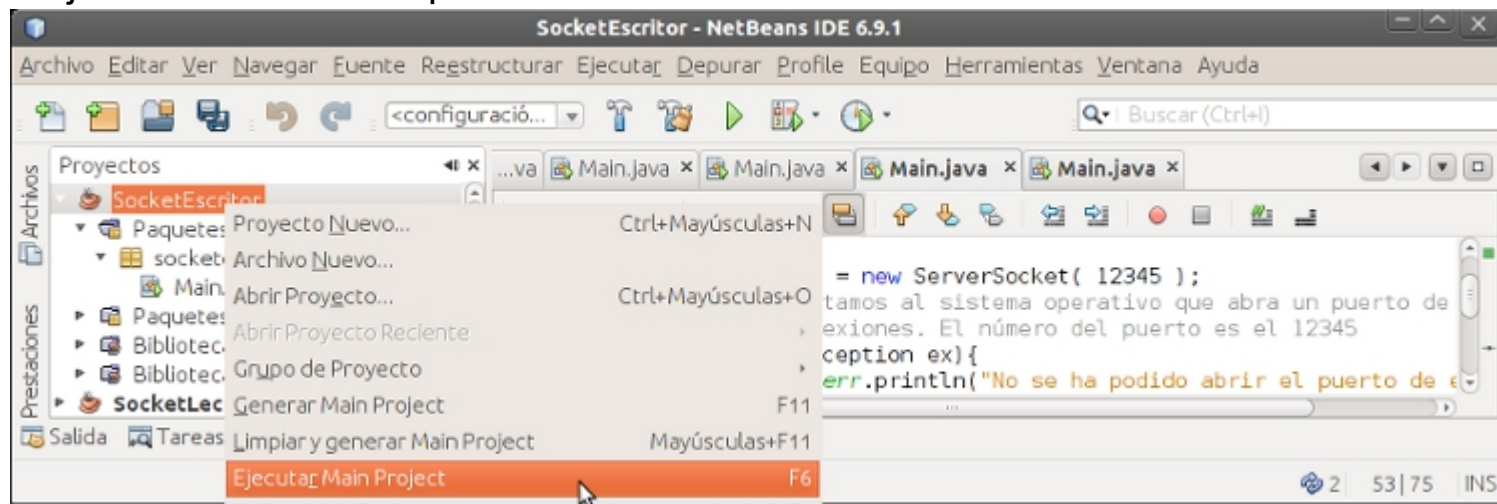
Proceso **Lector** en Socket (código completo):

```
}catch (Exception ex){
    System.err.println("No se ha podido establecer conexión.");
    System.err.println(ex.toString());
}finally{
    //Nos aseguramos de que se cierran los recursos que estamos utilizando
    if (entrada != null)
        try{
            entrada.close();
        }catch (IOException ex) {
            System.err.println("Se ha producido un error al cerrar el InputStreamReader.");
            System.err.println(ex.toString());
        }
    if (canal != null)
        try{
            canal.close();
        }catch (IOException ex) {
            System.err.println("Se ha producido un error al cerrar el Socket.");
            System.err.println(ex.toString());
        }
    }
}
```

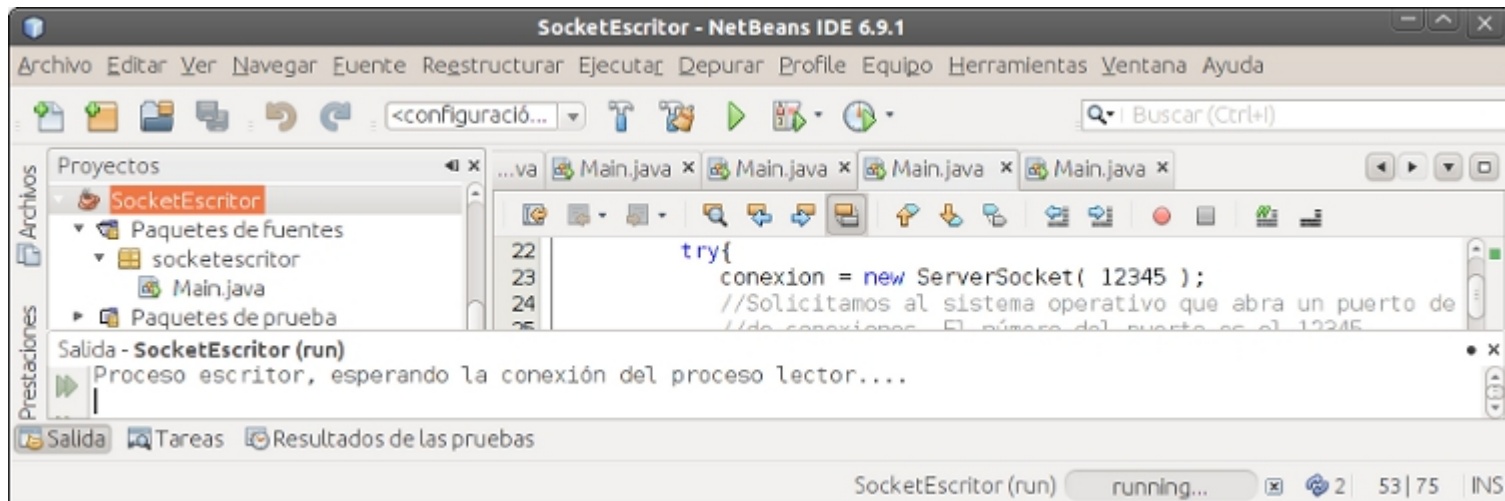
Probar nuestro ejemplo de sockets

Para probar nuestro ejemplo con los procesos que lo forman, haremos lo siguiente:

1. Lanzar la ejecución de nuestro proceso SocketEscritor.



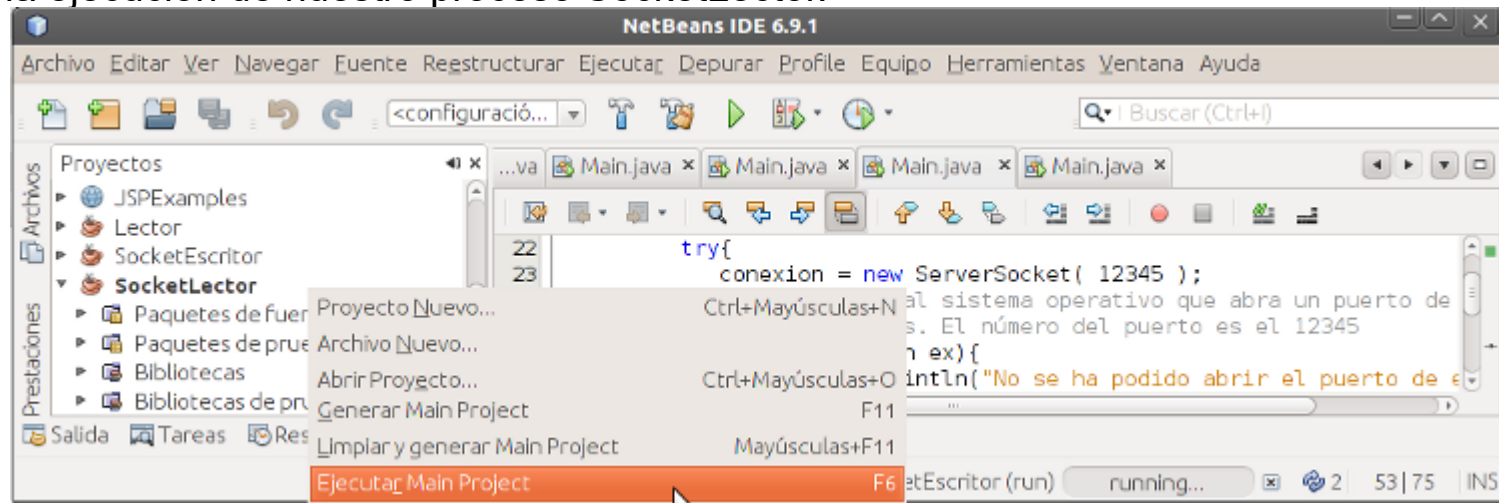
2. El proceso Escritor se queda a la espera de la conexión del proceso Lector.



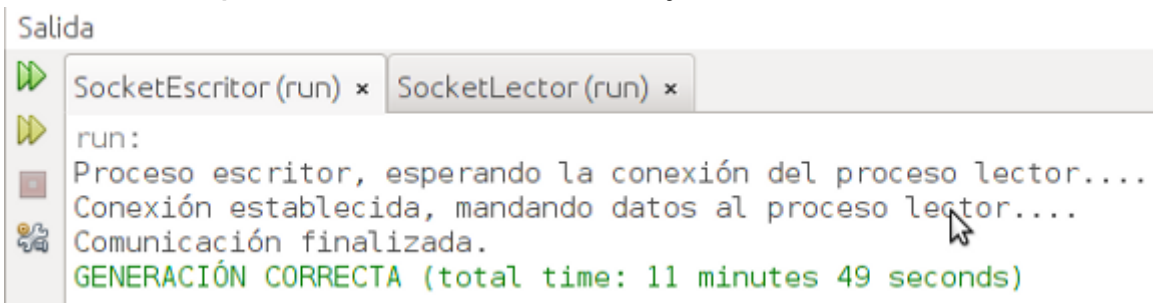
Probar nuestro ejemplo de sockets

Para probar nuestro ejemplo con los procesos que lo forman, haremos lo siguiente:

3. Lanzar la ejecución de nuestro proceso SocketLector.



4. Los procesos SocketEscritor y SocketLector se comunican y finalizan:

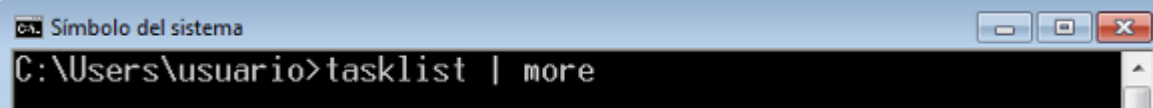


Ejemplo 2:

Comunicación utilizando tuberías

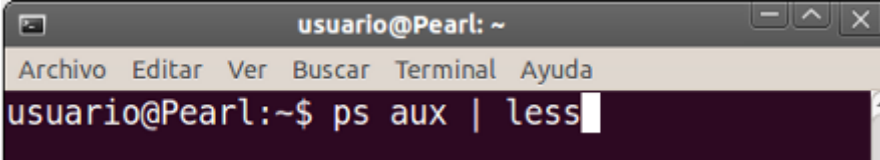
- Una **tubería o pipe (|)** permite establecer **un canal de comunicación entre dos procesos**.
 - La comunicación entre los procesos, se establece en el terminal de comandos.
Comando → proceso1 | proceso2
 - La salida estándar del primer proceso, se convierte en la entrada estándar del segundo proceso.
 - Se suelen utilizar las tuberías, para paginar la salida de los comandos:

En Windows:



```
Símbolo del sistema
C:\Users\usuario>tasklist | more
```

En GNU/Linux:



```
usuario@Pearl: ~
Archivo Editar Ver Buscar Terminal Ayuda
usuario@Pearl:~$ ps aux | less
```

Componentes del ejemplo de tuberías:

1. Proceso **Escritor** en tubería:
 - A. El proceso escribe en su salida estándar: System.out.
2. Proceso **Lector** en tubería:
 - A. El proceso lee de su entrada estándar: System.in.
 - B. Trata la entrada y muestra la información en pantalla.
3. Probar los ejemplos utilizando una tubería para redirigir sus salida y entrada estándar.

Ejemplo 2: Utilizando tuberías

Proceso Escritor

Proceso **Escritor**:

- Import necesarios:
 - Ninguno. Sólo vamos a hacer uso de la salida estándar del proceso: System.out.

A. Escribir en salida estándar: System.out.

```
package escritor;
/**...*/
public class Main {
    /**...*/
    public static void main(String[] args) {
        // Lo único que hacemos es escribir los números de 0 a 9 en
        // la salida estándar del proceso
        for (int i=0; i<10; i++)
            System.out.println(i);
    }
}
```

Estamos escribiendo en la salida estándar del proceso los números del 0 al 9.

Ejemplo 2: Utilizando tuberías

Proceso Escritor

Proceso **Lector** en Socket:

- Import necesarios:

- Para el uso de streams → `import java.io.*;`

Utilizaremos las clases `InputStreamReader`, `BufferedReader` e `IOException`.

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
```

A. Leer de la entrada estándar: `System.in`.

```
public static void main(String[] args) {
    // Vamos a leer de la entrada estándar del proceso y escribir
    // los datos que se reciben en la salida estándar del proceso.

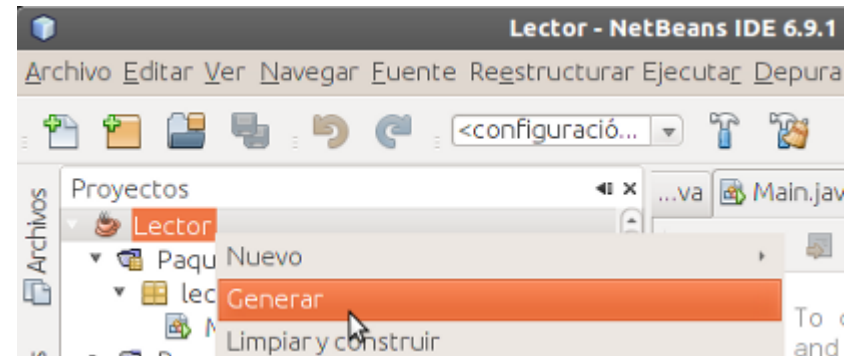
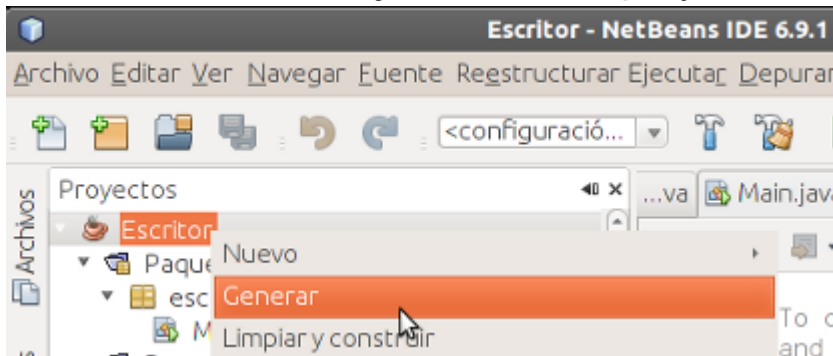
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader bf = new BufferedReader (isr);
    // Obtenemos el stream de lectura de la entrada estándar
    // utilizamos un lector con Buffered, para no perder ningún dato
    String lineaTeclado = null;
    try{
        System.out.println("Proceso lector");
        //Mostramos que el proceso que está escribiendo es el que está
        //leyendo los datos.
        while ((lineaTeclado = bf.readLine()) != null){
            //Vamos leyendo y mostrando los datos
            System.out.println(lineaTeclado);
        }
    }catch(IOException ex){
        System.err.println("Se ha producido un error de E/S.");
        System.err.println(ex.toString());
    }
}
```

B. Tratar la información y generar los resultados.

Probar nuestro ejemplo de tuberías

Para probar nuestro ejemplo con los procesos que lo forman, haremos lo siguiente:

1. Generar los ficheros .jar de ambos proyectos.



2. Lanzar la ejecución desde un terminal de comandos.

Para que nos sea más cómodo, hemos copiado los ficheros .jar generados en la misma carpeta.

En Windows:

```
C:\Users\usuario\Pruebas>
java -jar Escritor.jar | java -jar Lector.jar
Proceso lector
0
1
2
3
4
5
6
7
8
9
C:\Users\usuario\Pruebas>
```

En GNU/Linux:

```
usuario@Pearl:~$
java -jar Escritor.jar | java -jar Lector.jar
Proceso lector
0
1
2
3
4
5
6
7
8
9
usuario@Pearl:~$
```

Posibles mejoras

En los dos ejemplos planteados, buscamos introducirte al uso de mecanismo de comunicación entre procesos.

Se trata de ejemplos muy sencillos y por lo tanto muy mejorables.

En el uso de **sockets**, las **unidades 3 y 2 de este módulo**, nos permitirán completar proyectos de comunicación entre procesos, muy interesantes y potentes.

En cuanto a las **tuberías**, es un ejercicio interesante, que intentes, por ejemplo:

- El proceso escritor genera una secuencia de números aleatorios.
- El lector, devuelve el resultado ordenado.

Recomendaciones para probar el ejemplo

Si no has ido construyendo paso a paso el ejemplo, puedes probar el ejemplo que te suministramos implementado en la plataforma.

Recuerda:

1. Generar cada uno de los proyectos de forma independiente.
2. En el ejemplo de sockets:
 - Ejecutar siempre primero el Proyecto SocketEscritor, y después el Proyecto SocketLector. Si lo haces al contrario, el proceso SocketLector te indicará que no ha podido conectarse al proceso SocketEscritor.
3. En el ejemplo de tuberías:
 - Te resultará más fácil, si después de generar los proyectos, los archivos ejecutables (.jar) los copias a la misma carpeta.
 - Recuerda que el comando de ejecución tiene la siguiente sintaxis:
comando | comando
Donde cada comando, puede ser un comando del intérprete de comandos, o el comando que lanza la ejecución de un proyecto java desde línea de comandos: `java -jar archivo.jar`

Credenciales

| Imagen | Datos de licencia |
|--|-------------------|
| <p>Todas las capturas de pantalla de esta presentación, tienen como datos de licencia:</p> <p>Autoría: Margarita I. Nieto Castillejo Licencia: Uso educativo-no comercial. Procedencia: Capturas de pantalla del IDE NetBeans 6.9.1, terminales de comandos de los sistemas operativos Windows 7 y Ubuntu 10.10.</p> | |