

# Comunicación COM desde Java y por consola



[https://commons.wikimedia.org/wiki/File:Arduino\\_Uno\\_-\\_R3.jpg](https://commons.wikimedia.org/wiki/File:Arduino_Uno_-_R3.jpg)

Práctica 5 – Teoría (v1.4 octubre 2022)

## Software para robots

Cristian González García

[gonzalezcristian@uniovi.es](mailto:gonzalezcristian@uniovi.es)

# Índice

1.	Introducción .....	2
2.	Librerías .....	2
	RXTX .....	3
	Arduino .....	5
	Java.....	5
	PanamaHitek.....	10
	Fazecast/jSerialComm.....	10
	Arduino (Envía datos).....	11
	Java (Recibe datos).....	12
	Escuchar el puerto COM continuamente con un Listener .....	13
	jSerialComm (jSSC) de Scream3r.....	15
	Python .....	17
3.	Robot cartesiano .....	18
	Recomendaciones para montarlo .....	18
	Pinza .....	19
4.	Posibles problemas .....	20
	Shutdown or disconnected .....	20
	EXCEPTION_ACCESS_VIOLATION .....	20
	...has been compiled by a more recent version of the Java Runtime (class file version...	21

## 1. Introducción

En muchos casos, puede que nos interese enviar datos a Arduino desde un programa. Esta tarea, en la mayoría de los lenguajes de programación resulta relativamente sencillo, pues consiste en transferir datos a través del puerto COM/Serie (USB).

Este tutorial está basado en el de la página original de Arduino: <https://playground.arduino.cc/Interfacing/Java>. No obstante, este PDF va explicándolo más paso a paso. Además, incluye varias librerías más modernas.

## 2. Librerías

Para comunicarnos desde Java con el Arduino hay varias formas y muchas librerías. Está **Panamahitek** que tiene muchos ejemplos e interfaces en Java, y **jSerialComm de Fazecast** que es más simple. Ambas basadas sobre **jSerialComm de Screarm3r**, cuya última actualización fue en **2014**.

Antiguamente, se utilizaba la librería **RXTX**, pero desde Arduino no recomiendan su uso debido a su **inestabilidad y a su último año de actualización (2006-2009)**. No obstante, sigue funcionando perfectamente y hay ejemplos de código en la página de Arduino, aunque puede contener algún bug y problema debido a que está obsoleta.

Se mantienen todos los apartados por si fueran de interés o se quieran utilizar debido a que hay muchos ejemplos con ellos en Internet. Hay que bajar la versión de bits correcta (32 o 64 bits), así como la versión adecuada para tu sistema operativo (Windows o Linux).

También se puede realizar la conexión desde Python (<https://playground.arduino.cc/Interfacing/Python/>), C, u otros lenguajes de programación. **Python es muy recomendable**, pues hay una gran variedad de ejemplos y es mucho menos verboso.

**Nota:** actualmente RxTx está dando **muchos fallos las librerías de Java, desde Java8\_261** incluida, y con Windows 10, y también a gente por tener 64 bits. Exactamente, errores de violación de acceso (Ilustración 1). La solución para estos problemas es utilizar una versión anterior de Java (como puede ser Java JDK\_7u80, Java8\_241, etc.), a pesar de que desde Java dicen que no es un problema de Java debido a que ocurre en el *Java Native Interface* (JNI) (<https://bugs.openjdk.java.net/browse/JDK-8154044>, <https://bugs.openjdk.java.net/browse/JDK-8225635>, ...). Tened en cuenta que igual en los laboratorios de la EII no podréis hacer estos cambios de versión, así que tendréis que llevar una versión portátil de Java o bien descargarla del Campus Virtual, y después configurar el Eclipse para que use esta versión.

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
# EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x0000000180005b00, pid=10756, tid=14996
#
# JRE version: Java(TM) SE Runtime Environment (13.0.2+8) (build 13.0.2+8)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (13.0.2+8, mixed mode, sharing, tiered, compressed oops, gl gc, windows-amd64)
# Problematic frame:
# C [rxtxSerial.dll+0x5b00]
#
# No core dump will be written. Minidumps are not enabled by default on client versions of Windows
#
# An error report file with more information is saved as:
# D:\Eclipse workspace\RXTXExample\hs_err_pid10756.log
#
# If you would like to submit a bug report, please visit:
# http://bugreport.java.com/bugreport/crash.jsp
# The crash happened outside the Java Virtual Machine in native code.
# See problematic frame for where to report the bug.
#
```

*Ilustración 1 Error RXTX con Java*

En el caso de optar por utilizar un sistema **Linux desde el WSL de Windows 10**, también hay problemas de conexión con el puerto USB y requiere una configuración previa. Lo mejor es utilizarlo directamente en Windows, o en Linux.

## RXTX

Manual: [http://rxtx.qbang.org/wiki/index.php/Installation\\_on\\_MS-Windows](http://rxtx.qbang.org/wiki/index.php/Installation_on_MS-Windows)

Vídeo de instalación utilizando Eclipse (64 bits):

[https://www.youtube.com/watch?v=43Vdpz1YmdU&ab\\_channel=Sourcloud](https://www.youtube.com/watch?v=43Vdpz1YmdU&ab_channel=Sourcloud)

Web:

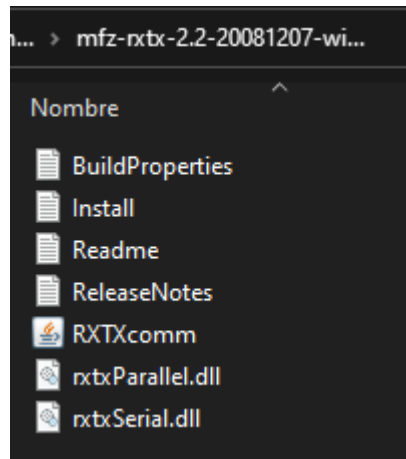
- 64 y 32 bits: <http://fizzed.com/oss/rxtx-for-java>
- 32 bits: [http://rxtx.qbang.org/wiki/index.php/Main\\_Page](http://rxtx.qbang.org/wiki/index.php/Main_Page)

Desarrollo y ejemplos: <http://rxtx.qbang.org/wiki/index.php/Development>

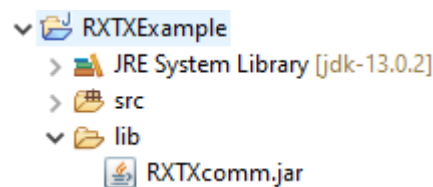
Otra web: <http://fizzed.com/oss/rxtx-for-java>

**Nota:** con RxTx hay que utilizar una versión de Java anterior a Java8\_261.

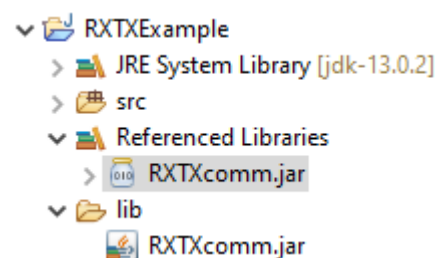
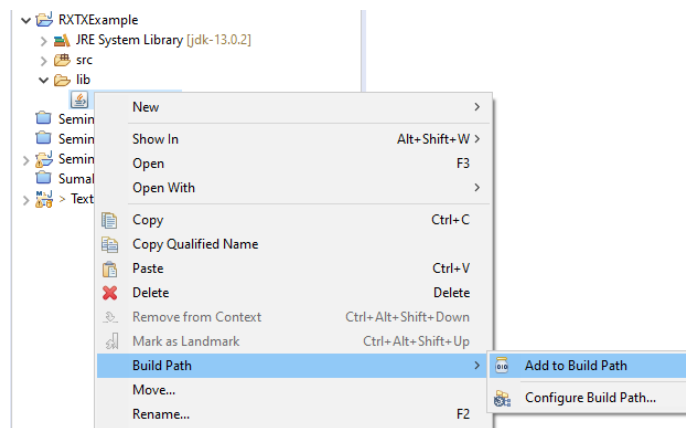
1. Descargar: <http://rxtx.qbang.org/wiki/index.php/Download>
2. Descomprimir archivo descargado



3. Copiamos el fichero **rxtxSerial.dll** y **rxtxParallel.dll** en la ruta **%JAVA\_HOME%\bin**
4. Copiamos el fichero **RXTXcomm.jar** en **%JAVA\_HOME%\bin\ext**
5. Crear un proyecto Java en Eclipse
6. Crear una carpeta llamada «lib»
7. Copiar dentro de la carpeta el archivo «RXTXcomm.jar»



8. Botón derecho sobre el archivo «jssc.jar» -> Build Path -> Add to build Path

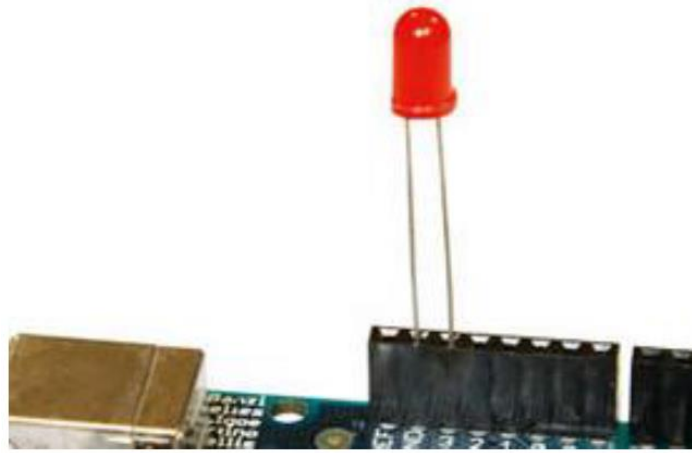


9. Crear una clase Java en el proyecto.
10. Probar a importar «RXTX» que está en «gnu.io» y el siguiente código.

```
import gnu.io.*;
```

### Arduino

Crearemos un programa que se comunique con el PC y que al recibir la cadena «encender» / «apagar» hará lo propio con el LED conectado al Pin 13 y GND (directamente sobre la placa).



```
String command;
int pinLED = 13;

void setup() {
  Serial.begin(9600);
  pinMode(pinLED, OUTPUT);
}

void loop() {
  Serial.println("Gestionamos el LED");
  while (Serial.available() == 0) {} // Esperamos por la
  entrada...

  command = Serial.readString();

  // Si tenemos nueva línea sería "encender\n"
  if(command.equals("encender")) {
    Serial.println("Encendido");
    digitalWrite(pinLED, HIGH);
  } else if (command.equals("apagar")) {
    Serial.println("Apagado");
    digitalWrite(pinLED, LOW);
  }
}
```

### Java

Creamos una nueva clase **Comunicador.Java**

Lo primero que vamos a hacer es crear un método para ver **los puertos abiertos**.

```
import gnu.io.CommPortIdentifier;
import java.util.Enumeration;
```

```

import java.util.HashMap;

public class Comunicador {

    static HashMap<String, CommPortIdentifier> mapaPuertos
=
    new HashMap<String, CommPortIdentifier>();

    public void listarPuertos() {
        CommPortIdentifier idPuerto;

        Enumeration enumPuertos =
CommPortIdentifier.getPortIdentifiers();

        while (enumPuertos.hasMoreElements()) {
            idPuerto = (CommPortIdentifier)
enumPuertos.nextElement();

            if (idPuerto.getPortType() ==
CommPortIdentifier.PORT_SERIAL) {
                System.out.println(idPuerto.getName());
                mapaPuertos.put(idPuerto.getName(),
idPuerto);
            }
        }
    }
}

```

Creamos una nueva clase **Main.java** con un método **Main** y desde ahí llamamos a nuestro comunicador para listar los puertos abiertos.

```

public class Main {

    public static void main(String[] args) throws
InterruptedException {
        System.out.println("Inicio");
        Comunicador comunicador = new Comunicador();

        comunicador.listarPuertos();
        System.out.println("Final");
    }
}

```

Si el Arduino está conectado debería imprimir **COM3**.

Para comenzar la transmisión de datos nos tenemos que **conectar al puerto**. Debemos obtener el **SerialPort**, lo utilizaremos a partir de ahora para el envío y recepción de datos. Añadimos esto a la clase Comunicador.

```

// Puerto de Conexión
private SerialPort puertoSerial;

public void conectarse(String nombrePuerto) {

    CommPortIdentifier selectedPortIdentifier =

```

```

(CommPortIdentifier)mapaPuertos.get(nombrePuerto);
    try {

        // open para abrir el puerto
        puertoSerial =
            (SerialPort)
selectedPortIdentifier.open("ControlPC", 2000); // TimeOut

    } catch (PortInUseException e) {
        System.out.println("Error A Puerto en uso
"+e.getMessage());
    } catch (Exception e) {
        System.out.println("Error B "+e.getMessage());
    }
}

```

Llamamos al método de conexión desde el main.

```

System.out.println("Inicio");
Comunicador comunicador = new Comunicador();

comunicador.listarPuertos();
comunicador.conectarse("COM3");

System.out.println("Final");

```

Debemos tener cuidado cuando realicemos pruebas, **es bastante fácil que el puerto COM3 se quede abierto y no nos deje conectarnos de nuevo**. En este caso hay que cerrar el Eclipse y el IDE de Arduino y volver a abrirlos.

No podemos acceder al mismo puerto simultáneamente desde el Monitor Serie (IDE de Arduino) y el programa Java.

Una vez conectados al puerto serie iniciamos los mecanismos de envío y recepción de datos.

```

private InputStream input = null;
private OutputStream output = null;

public void inicializarES() {
    try {
        //
        input = puertoSerial.getInputStream();
        output = puertoSerial.getOutputStream();

    } catch (IOException e) {
        System.out.println("Error C "+e.getMessage());
    }
}

```



El más sencillo será el método que nos permita enviar datos (escribir).

```
public void escribir(String texto) {
    try {
        output.write(texto.getBytes());
    } catch (IOException e) {
        System.out.println("Error D "+e.getMessage());
    }
}
```

Para poder probarlo sin bloquear el puerto en cada ejecución implementamos también el método de desconexión (y cierre de la E/S).

```
public void cerrarESyDesconectar() {
    try {
        puertoSerial.close();
        input.close();
        output.close();
    }
    catch (Exception e) {
        System.out.println("Error E "+e.getMessage());
    }
}
```

Probamos a ejecutar una secuencia desde el main: conectarse, esperar, enviar el comando encender, esperar, y enviar el comando apagar.

```
comunicador.conectarse("COM3");

comunicador.inicializarES();
Thread.sleep(3000);
comunicador.escribir("encender");
Thread.sleep(3000);
comunicador.escribir("apagar");
Thread.sleep(3000);

comunicador.cerrarESyDesconectar();

System.out.println("Final");
```

Lo único que nos queda es permitir que el programa sea capaz de leer datos. Como Arduino puede enviarnos datos en cualquier momento debemos registrar un **escuchador**.

Hacemos que la clase Comunicador implemente **SerialPortEventListener**.

```
public class Comunicador implements
SerialPortEventListener {
```

Debemos dar implementación al método **serialEvent**.

Los bytes van llegando uno a uno al método. Como son cadenas de texto los vamos transformando y guardando en una cadena, hasta recibir el fin de línea. Será entonces cuando imprimamos todo el mensaje.

```
String lectura = "";

@Override
public void serialEvent(SerialPortEvent arg0) {
    if (arg0.getEventType() ==
        SerialPortEvent.DATA_AVAILABLE) {
        try {
            byte byteLeido = (byte) input.read();
            lectura += new String(new byte[] { byteLeido
        });

            if (new String(new byte[] { byteLeido
        }).equals("\n")) {
                System.out.println("Arduino dice: " +
        lectura);
                lectura = "";
            }

        } catch (Exception e) {
            System.out.println("Error F " +
        e.getMessage());
        }
    }
}
```

Creemos un método para iniciar la escucha, indicando que el escuchador es la propia instancia de la clase (puede porque implementa **SerialPortEventListener**) y que debe notificarnos siempre que haya algún dato disponible para leer.

```
public void inicializarEscucha() {
    try {
        puertoSerial.addEventListener(this);
        puertoSerial.notifyOnDataAvailable(true);
    } catch (TooManyListenersException e) {
        System.out.println("Error G " + e.getMessage());
    }
}
```

Cuando nos desconectamos también deberíamos dejar de escuchar.

```
public void cerrarESyDesconectar() {
    try {
        puertoSerial.removeEventListener();
        puertoSerial.close();
        input.close();
        output.close();
    }
    catch (Exception e){
```

```
        System.out.println("Error H "+e.getMessage());  
    }  
}
```

Probamos a iniciar el escuchador antes de escribir los comandos en el método Main, de esta forma podremos ver las respuestas que el Arduino nos envía al procesar los comandos.

```
System.out.println("Inicio");  
Comunicador comunicador = new Comunicador();  
  
comunicador.listarPuertos();  
comunicador.conectarse("COM3");  
  
comunicador.inicializarEscucha();  
  
comunicador.inicializarES();  
Thread.sleep(3000);  
comunicador.escribir("encender");  
Thread.sleep(3000);  
comunicador.escribir("apagar");  
Thread.sleep(3000);  
  
comunicador.cerrarESyDesconectar();  
  
System.out.println("Final");
```

## PanamaHitek

- Web: [http://panamahitek.com/libreria-panamahitek\\_arduino/](http://panamahitek.com/libreria-panamahitek_arduino/)
- GitHub (versión 3.2.0): [https://github.com/PanamaHitek/PanamaHitek\\_Arduino](https://github.com/PanamaHitek/PanamaHitek_Arduino)
- SourceForge: <https://sourceforge.net/projects/arduinojava/files/>
  - No contiene la última versión

Cuenta con un proyecto Java en el que se muestran muchos y variados ejemplos de uso junto a su interfaz gráfica. Además, funciona con las versiones nuevas de Java (al menos con la 13). **Desde la versión 2.8.0 cambiaron RxTx por JSSC de Scream3r.**

## Fazecast/jSerialComm

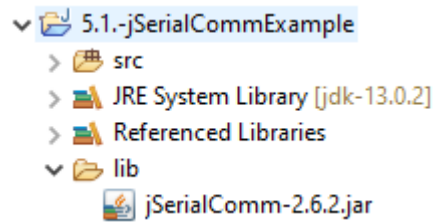
GitHub: <https://github.com/Fazecast/jSerialComm>

Descargar (versión 2.9.2): <https://github.com/Fazecast/jSerialComm/wiki>

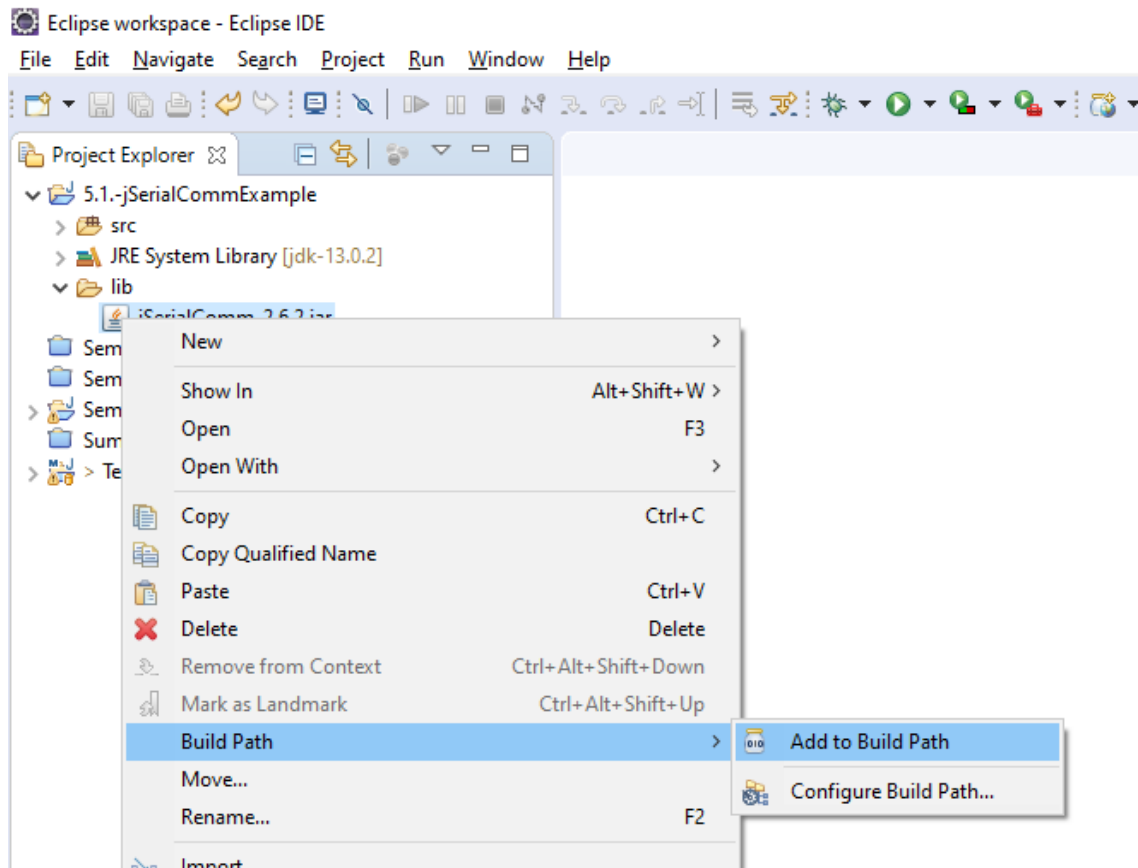
Instalación <https://github.com/Fazecast/jSerialComm/wiki/Building-Tutorial>

Ejemplos: <https://github.com/Fazecast/jSerialComm/wiki/Usage-Examples>

1. Descargar la librería.
2. Crear un proyecto Java en Eclipse
3. Crear una carpeta llamada dentro del proyecto Java en Eclipse «lib».
  - a. De esta manera haremos que la librería siempre esté junto al proyecto.
4. Copiar dentro de la carpeta el archivo «jSerialComm-2.6.2.jar»



5. Botón derecho sobre el archive «jSerialComm-2.6.2.jar» -> Build Path -> Add to build Path



6. Crear una clase Java en el proyecto.
7. Probar a importar «com.fazecast.jSerialComm» y el siguiente código fuente.

#### Arduino (Envía datos)

```
int pinLED = 13;

void setup() {
    Serial.begin(9600);
    pinMode(pinLED, OUTPUT);
}

void loop() {
    Serial.println("Prueba");
}
```

## Java (Recibe datos)

```
import java.nio.charset.StandardCharsets;

import com.fazecast.jSerialComm.SerialPort;
import com.fazecast.jSerialComm.SerialPortEvent;
import com.fazecast.jSerialComm.SerialPortMessageListener;

final class MessageListener implements
SerialPortMessageListener {
    @Override
    public int getListeningEvents() {
        return SerialPort.LISTENING_EVENT_DATA_RECEIVED;
    }

    /**
     * Otros posibles caracteres de final de línea:
     * http://lsi.vc.ehu.es/pablogn/docencia/FdI/FdIc/labs/a1/htm/asciis.html
     */
    @Override
    public byte[] getMessageDelimiter() {
        // 0xA = \n
        return new byte[] { (byte) 0xA };
    }

    @Override
    public boolean delimiterIndicatesEndOfMessage() {
        return true;
    }

    @Override
    public void serialEvent(SerialPortEvent event) {
        // Recibimos una cadena de bytes (byte[]) y la
        almacenamos
        byte[] delimitedMessage = event.getReceivedData();

        // Convertimos el byte[] en String
        String cad = new String(delimitedMessage,
StandardCharsets.UTF_8);

        System.out.println("Mensaje recibido: " + cad);
    }
}

public class Example3 {

    static public void main(String[] args) {
        Example3 example3 = new Example3();
        example3.infinite();
        //main.xSeconds(1000);
    }

    /**
     * El programa queda hasta el infinito ejecutándose
     */
}
```

```

    private void infinite() {
        // Miramos los puertos disponibles y como solo
tenemos 1 cosa enchufada, cogemos el 0 (del Arduino)
        SerialPort comPort = SerialPort.getCommPorts()[0];
        // Abrimos el puerto
        comPort.openPort();

        // Creamos el Listener y lo añadimos al gestor del
puerto
        MessageListener listener = new MessageListener();
        comPort.addDataListener(listener);
    }

    /**
     * Queda el programa durmiendo para seguir escuchando al
Arduino durante X segundos
     * @param milliseconds int
     */
    private void xSeconds(int milliseconds) {
        SerialPort comPort = SerialPort.getCommPorts()[0];
        comPort.openPort();

        MessageListener listener = new MessageListener();
        comPort.addDataListener(listener);

        try {
            Thread.sleep(milliseconds);
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Borramos el Listener
        comPort.removeDataListener();
        // Cerramos el puerto. Importante si queremos hacer
otra tarea en el mismo puerto con el mismo programa
        comPort.closePort();
    }
}

```

### Escuchar el puerto COM continuamente con un Listener

En este ejemplo se muestra cómo se puede estar leyendo indefinidamente o durante X segundos el puerto COM al que está conectado el Arduino por medio de un *Listener*. De esta manera, el programa estará atento para cuando haya datos en ese puerto. Hay más ejemplos con otras posibles lecturas en: <https://github.com/Fazecast/jSerialComm/wiki/Event-Based-Reading-Usage-Example>

### Arduino (Envía datos)

```

void setup() {
    Serial.begin(9600);
}

void loop() {
    Serial.println("Prueba");
}

```

*Java (Recibe datos)*

```
import java.nio.charset.StandardCharsets;

import com.fazecast.jSerialComm.SerialPort;
import com.fazecast.jSerialComm.SerialPortEvent;
import com.fazecast.jSerialComm.SerialPortMessageListener;

final class MessageListener implements
SerialPortMessageListener {
    @Override
    public int getListeningEvents() {
        return SerialPort.LISTENING_EVENT_DATA_RECEIVED;
    }

    /**
     * Otros posibles caracteres de final de línea:
     http://lsi.vc.ehu.es/pablogn/docencia/FdI/FdIc/labs/a1/htm/asciis.html
     */
    @Override
    public byte[] getMessageDelimiter() {
        // 0xA = \n
        return new byte[] { (byte) 0xA };
    }

    @Override
    public boolean delimiterIndicatesEndOfMessage() {
        return true;
    }

    @Override
    public void serialEvent(SerialPortEvent event) {
        // Recibimos una cadena de bytes (byte[]) y la
almacenamos
        byte[] delimitedMessage = event.getReceivedData();

        // Convertimos el byte[] en String
        String cad = new String(delimitedMessage,
StandardCharsets.UTF_8);

        System.out.println("Mensaje recibido: " + cad);
    }
}

public class Example3 {

    static public void main(String[] args) {
        Example3 example3 = new Example3();
        example3.infinite();
        //main.xSeconds(1000);
    }

    /**
     * El programa queda hasta el infinito ejecutándose
     */
}
```

```

    private void infinite() {
        // Miramos los puertos disponibles y como solo
tenemos 1 cosa enchufada, cogemos el 0 (del Arduino)
        SerialPort comPort = SerialPort.getCommPorts()[0];
        // Abrimos el puerto
        comPort.openPort();

        // Creamos el Listener y lo añadimos al gestor del
puerto
        MessageListener listener = new MessageListener();
        comPort.addDataListener(listener);
    }

    /**
     * Queda el programa durmiendo para seguir escuchando al
Arduino durante X segundos
     * @param milliseconds int
     */
    private void xSeconds(int milliseconds) {
        SerialPort comPort = SerialPort.getCommPorts()[0];
        comPort.openPort();

        MessageListener listener = new MessageListener();
        comPort.addDataListener(listener);

        try {
            Thread.sleep(milliseconds);
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Borramos el Listener
        comPort.removeDataListener();
        // Cerramos el puerto. Importante si queremos hacer
otra tarea en el mismo puerto con el mismo programa
        comPort.closePort();
    }
}

```

## jSerialComm (jSSC) de Scream3r

Última actualización en 2014.

Descarga: <https://code.google.com/archive/p/java-simple-serial-connector/downloads>

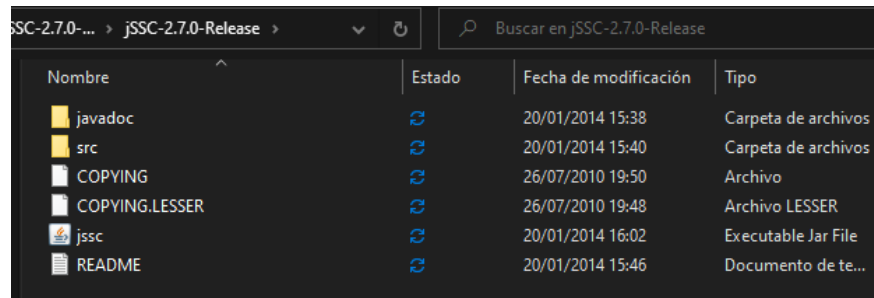
GitHub: <https://github.com/scream3r/java-simple-serial-connector>

Manual: [https://code.google.com/archive/p/java-simple-serial-connector/wikis/jSSC\\_Start\\_Working.wiki](https://code.google.com/archive/p/java-simple-serial-connector/wikis/jSSC_Start_Working.wiki)

Ejemplos: [https://code.google.com/archive/p/java-simple-serial-connector/wikis/jSSC\\_examples.wiki](https://code.google.com/archive/p/java-simple-serial-connector/wikis/jSSC_examples.wiki)

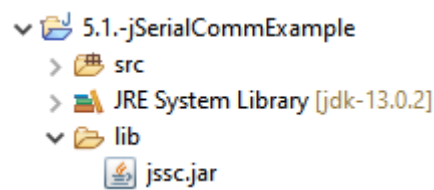


1. Descargar:
  - a. Actual (2.8.0): <https://github.com/scream3r/java-simple-serial-connector/releases>
  - b. <https://code.google.com/archive/p/java-simple-serial-connector/downloads>
2. Descomprimir archivo descargado

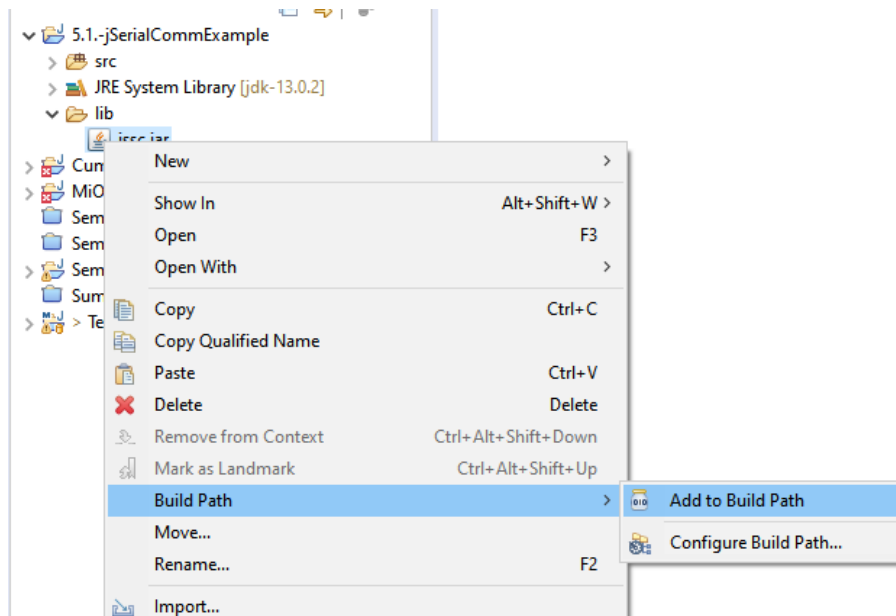


Nombre	Estado	Fecha de modificación	Tipo
javadoc	↻	20/01/2014 15:38	Carpeta de archivos
src	↻	20/01/2014 15:40	Carpeta de archivos
COPYING	↻	26/07/2010 19:50	Archivo
COPYING.LESSER	↻	26/07/2010 19:48	Archivo LESSER
jssc	↻	20/01/2014 16:02	Executable Jar File
README	↻	20/01/2014 15:46	Documento de te...

3. Crear un proyecto Java en Eclipse
4. Crear una carpeta llamada «lib»
5. Copiar dentro de la carpeta el archivo «jssc.jar»



6. Botón derecho sobre el archivo «jssc.jar» -> Build Path -> Add to build Path



7. Crear una clase Java en el proyecto.
8. Probar a importar «jssc».

## Python

Una posible librería es la pyserial:  
<https://pyserial.readthedocs.io/en/latest/shortintro.html>

En este ejemplo, enviamos un número entero desde Python, lo leemos en Arduino, y este devuelve un String confirmando el número que recibió.

Una vez están arrancadas ambas aplicaciones, si se quiere actualizar la del Arduino, hay que apagar la aplicación Python para que libere el puerto Serie. En caso contrario, dará una excepción al subir el código desde el IDE de Arduino.

Python:

```
import serial
import time

arduino = serial.Serial(port='COM5', baudrate=9600, timeout=.1)

def write_read(value):
    """Escribimos como bytes (1 solo dígito) usando UTF-8"""
    arduino.write(bytes(value, 'utf-8'))
    time.sleep(0.05)
    data = arduino.readline()
    return data

def main():
    while True:
        num = input("Envía un número desde a Python a Arduino: ") #
        Taking input from user
        input_value = write_read(num)
        print("Python dice:" + str(input_value)) # printing the value

main()
```

Arduino:

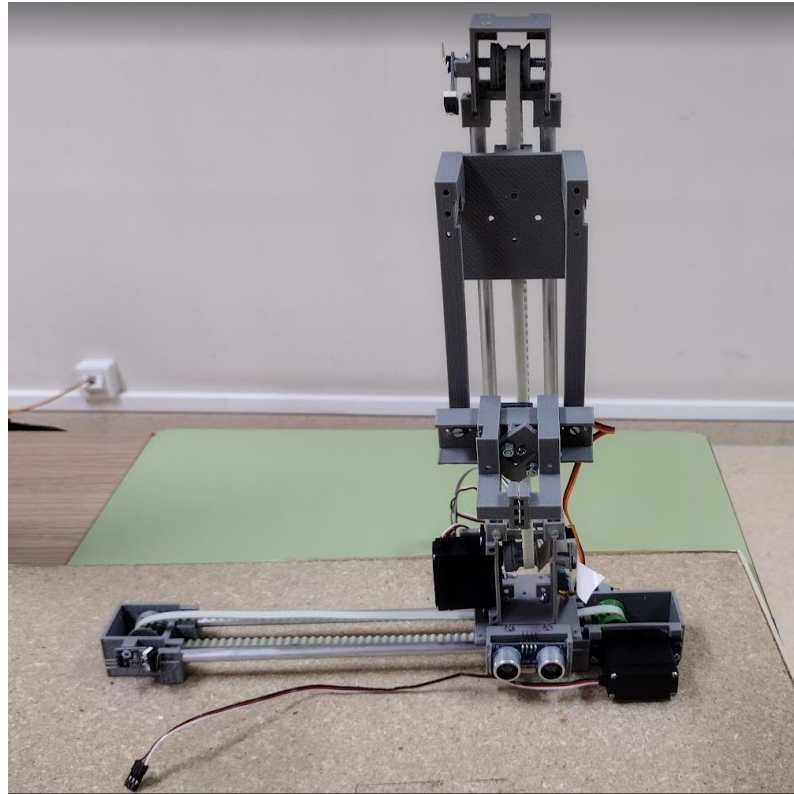
```
int pythonData;

void setup() {
    // Mismos Baudios que desde Python
    Serial.begin(9600);
    // Espera máxima datos del puerto Serie
    Serial.setTimeout(1);
}

void loop() {
    // Esperamos en bucle hasta que tengamos datos en el puerto serie
    while (!Serial.available());
    // Recogemos ese dato que sabemos será un entero
    pythonData = Serial.readString().toInt();
    // Hacemos lo que sea
    Serial.print("Arduino recibio un: ");
    Serial.print(pythonData);
    Serial.print(" y le sumamos 1: ");
```

```
Serial.println(String(int(pythonData)+1));  
}
```

### 3. Robot cartesiano



*Ilustración 2 Robot cartesiano XY*

#### Recomendaciones para montarlo

1. Usar tirafondos en vez de tornillos cuando sea posible.
2. Añadir al actuador de abajo el sensor de ultrasonidos, en caso de que se vaya a usar, aunque eso será en la siguiente práctica.
3. Primero juntar los dos actuadores, pero teniendo la correa dentada del de abajo suelta, pues se atornilla por ese sitio y la correa dentada podría molestar. Aquí viene bien meter un destornillador en lugar del primer tornillo para que sujete ambos actuadores y sea más fácil meter el primer tornillo. Apretar bien las dos partes de unión cuando se meta el tornillo para así conseguir que quede la junta plana y bien nivelada.
4. Ajustar la correa del actuador de abajo
5. Al atornillar el actuador de abajo a la base, no se puede atornillar del todo, pues sino se atravesaría la base con el tirafondo. **¡¡¡Cuidado de no estropear las mesas!!!** Para atornillarlo de una manera más fácil, lo mejor es introducir el tirafondo, darle un par de golpes y después atornillarlo.

Un **joystick 1** debe controlar el movimiento del carro del **actuador del eje X y del actuador eje Y**.

El **joystick 2** debe controlar el movimiento de la pinza:

- Al mover el **joystick 2** hacia la izquierda, la pinza debe comenzar a cerrarse lentamente.
- Al mover el **joystick 2** hacia la derecha, la pinza debe comenzar a abrirse lentamente
- Si en cualquier momento el usuario suelta el **joystick 2**, la pinza debe dejar de moverse y permanecer en el estado actual.
- La pinza solo necesita de un eje para abrirse o cerrarse, no necesita los dos ejes ni el botón del joystick, luego, solo se necesitarán conectar 3 de los 5 pines del joystick.

## Pinza

Para controlar el servo de 180 grados:

1. Añadiremos unos valores máximo y mínimo que serán tanto el tope de la pinza (depende del mecanismo) como de movimiento del servo (0-180).
2. En el setup movemos la pinza a sus grados mínimos (o máximos u otros), pero siempre inicializarla, pues si no, al usarla, podría superar alguno de los dos límites al no conocer su posición actual. Una solución sería a esto sería guardar en tarjeta SD su última posición.
3. Debemos almacenar sus grados actuales. En el caso de que se quiera abrir la pinza vamos sumando grados poco a poco a sus grados actuales. En el caso de que se quiera cerrar la pinza vamos restando grados poco a poco a su estado actual.

### Ejemplo de código de la pinza:

```
#include <Servo.h>

//1.- Grados pinza, MAX y MIN admitido: no sobrepasar apertura pinza
ni servo
int GMAX = 110;
int GMIN = 10;

// JOYSTICK PIN
int joystick_pinza_pin = A0;

// Servo PIN
int servo_pin = 8;

Servo servoPinza;
int gradosPinza = GMIN;

void setup() {
  Serial.begin(9600);
  servoPinza.attach(servo_pin);
  //2.- Iniciamos la pinza en ñps grados mínimos
  servoPinza.write(gradosPinza);
}

void loop() {
  int x = analogRead(joystick_pinza_pin);
  Serial.println("Pinza: "+String(x));

  // Aumentar / reducir grados segun X-Joystick
  // Joystick de 0 a 1024, desechamos las posiciones centrales
  if(x > 700){
    if (gradosPinza < GMAX)
      //3.- Cuanto más sumemos a los grados de la pinza, más
      rápido irá, pero también menos precisa será.
```

```

        gradosPinza = gradosPinza+1;
    } else if (x < 300){
        if (gradosPinza > GMIN)
            //3.- Cuanto más sumemos a los grados de la pinza, más
            rápido irá, pero también menos precisa será.
            gradosPinza = gradosPinza-1;
        }

        // Mover
        servoPinza.write(gradosPinza);
        Serial.println("Grados: "+String(gradosPinza));
    }
}

```

## 4. Posibles problemas

### Shutdown or disconnected

Este error se puede deber a varias cosas:

```

com.fazecast.jSerialComm.SerialPortIOException: This port appears to have been shutdown or disconnected.
    at com.fazecast.jSerialComm.SerialPort$SerialPortOutputStream.write(SerialPort.java:1537)
    at com.fazecast.jSerialComm.SerialPort$SerialPortOutputStream.write(SerialPort.java:1525)
    at Example.main(Example.java:28)

```

**Solo se puede utilizar o la consola del Arduino IDE o enviar comandos por Java.** Si tenemos la primera abierta y tratamos de enviar comandos desde Eclipse recibiremos la excepción de que el puerto debe de estar apagado o desconectado. Si este es el error, se soluciona cerrando la consola del Arduino IDE.

También puede ser debido a que intentamos hacer **conexiones muy seguidas o simplemente a que no conecta**. Se soluciona desconectando y conectando el Arduino de nuevo al puerto USB. Puede ser al mismo puerto USB.

También puede deberse a que haya quedado el programa Java corriendo. En este caso la solución es pararlo.



### EXCEPTION\_ACCESS\_VIOLATION

Actualmente está dando muchos fallos las librerías de Java, desde Java8\_261 incluida y Windows 10. Errores de violación de acceso.

La solución pasa por volver a versiones anteriores de Java (e.j.: Java JDK\_7u80), del sistema operativo o usar otro diferente y/o cambiar a 32 bits.

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
# EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x0000000180005b00, pid=10756, tid=14996
#
# JRE version: Java(TM) SE Runtime Environment (13.0.2+8) (build 13.0.2+8)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (13.0.2+8, mixed mode, sharing, tiered, compressed oops, gl gc, windows-amd64)
# Problematic frame:
# C [rxtxSerial.dll+0x5b00]
#
# No core dump will be written. Minidumps are not enabled by default on client versions of Windows
#
# An error report file with more information is saved as:
# D:\Eclipse workspace\RXTEXample\hs_err_pid10756.log
#
# If you would like to submit a bug report, please visit:
# http://bugreport.java.com/bugreport/crash.jsp
# The crash happened outside the Java Virtual Machine in native code.
# See problematic frame for where to report the bug.
#
```

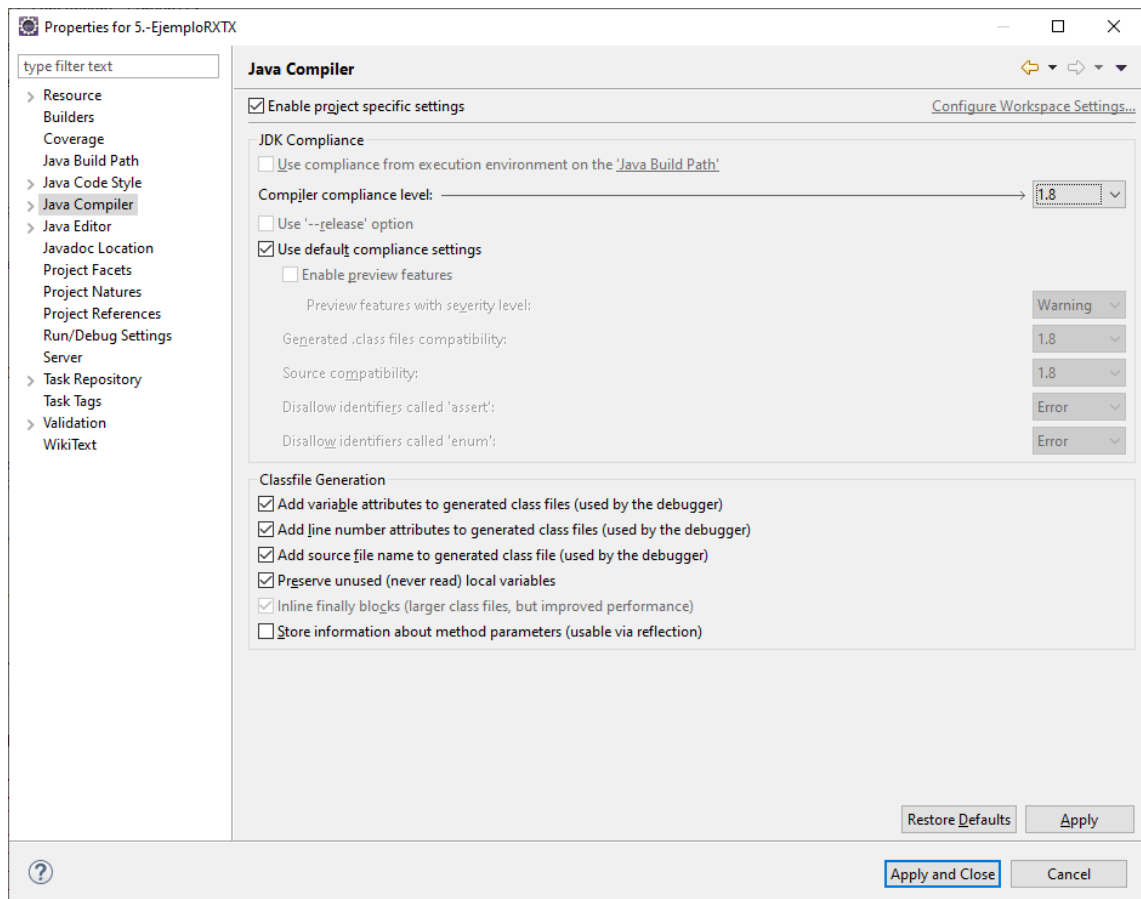
En el caso de optar por utilizar un sistema Linux desde el WSL de Windows 10, también hay problemas de conexión y requiere una configuración previa. En este caso mejor una máquina virtual con Linux que no sea WSL.

...has been compiled by a more recent version of the Java Runtime (class file version...

Este error ocurre cuando hemos compilado con una versión de Java, por ejemplo, la 13, y cambiamos después el proyecto para que utilice una versión más vieja, como la 8.

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: Main has been compiled by a more recent version of the Java Runtime (class file version 56.0), this version of the Java Runtime only recognizes class
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:756)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:468)
at java.net.URLClassLoader.access$100(URLClassLoader.java:74)
at java.net.URLClassLoader$1.run(URLClassLoader.java:369)
at java.net.URLClassLoader$1.run(URLClassLoader.java:365)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:362)
at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:355)
at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:495)
```

Para solucionarlo, hay que ir a propiedades del proyecto -> Java Compiler -> Compiler compliance level -> Seleccionar la versión de compilación que queremos utilizar.



Esto lo podemos saber si miramos el error y nos guiamos por su versión:

49 = Java 5  
 50 = Java 6  
 51 = Java 7  
 52 = Java 8  
 53 = Java 9  
 54 = Java 10  
 55 = Java 11  
 56 = Java 12  
 57 = Java 13  
 58 = Java 14  
 59 = Java 15  
 60 = Java 16