ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

APPLET DIDÁCTICO PARA COMUNICACIONES SERIE ASÍNCRONAS

AUTOR: Jorge Giovanny Araujo Criollo
DIRECTOR: Francesc Burrull i Mestres

Marzo / 2012

Autor	Jorge Giovanny Araujo Criollo
E-mail del Autor	jorge--giovanny@hotmail.com
Director	Francesc Burrull i Mestres
E-mail del Director	francesc.burrull@upct.es
Título del PFC	Applet Didáctico para Comunicaciones Serie Asíncronas
Descriptores	Simulador: Applet Didáctico

Resumen

Se plantea el desarrollo de una herramienta didáctica para la docencia de Fundamentos de Telemática, donde una de las prácticas a desarrollar en el laboratorio consiste en comunicaciones serie asíncronas, en concreto la comprensión de los principios fundamentales del puerto serie RS-232.

Hasta el momento se da al alumno un enunciado de prácticas, y éste las realiza en laboratorio con instrumentos reales. Se pretende dotar de una alternativa con instrumentos virtuales (simulados) y el valor añadido de una ayuda contextual integrada.

Titulación	Ingeniero Técnico de Telecomunicaciones
Intensificación	Telemática
Departamento	Departamento de Tecnologías de Información y Comunicaciones
Fecha de Presentación	

Índice General

1 **Introducción**. 7

2 **Objetivos**. 9

3 **Análisis y Diseño**. 11
3.1 Elementos que intervienen. 11
3.2 El interfaz RS-232. 12
3.3 UART. 14
3.4 Control de flujo. 14
3.5 Null-Módem (supresor del módem). 14
3.6 X-MODEM. 15
3.7 Control de dispositivos usando el puerto serie 15
3.8 Determinación del cable a usar. 15
3.9 Configuración del dispositivo. 15
3.10 Comunicación y control del dispositivo 16
3.11 Estudio del Nivel Físico 16
3.12 Estudio del Nivel de Enlace. 16

4 **ÁMBITO DE APLICACIÓN**. 17
4.1 Elección del lenguaje de desarrollo. 17
4.1.1 C 17
4.1.2 C++ 17
4.1.3 Java 17
4.1.4 Elección asumida 18
4.2 Elección del entorno de desarrollo. 18
4.2.1 JBuilder. 18
4.2.2 Kawa 18
4.2.3 NetBeans. 19
4.2.4 Eclipse 19
4.2.5 Elección asumida 19

5 **Implementación de la aplicación**. 21
5.1 Página de Inicio. 21
5.2 Escritorio Virtual. 23
5.3 HyperTerminal. 23
5.4 Osciloscopio. 31
5.5 POD RS-232. 34
5.6 Diagrama UML de la clase RS232 (RS232.java). 40
5.7 Diagrama UML de la clase HyperTerminal 41
5.8 Diagrama UML de la clase Prefencias de Terminal 45
5.9 Diagrama UML de la clase Comunicaciones 47
5.10 Diagrama UML de la clase Osciloscopio (Osciloscopio.java). 49
5.11 Diagrama UML de la clase PantallaOsciloscopio 51
5.11 Diagrama UML de la clase PODRS232 (PODRS232.java). 52
5.12 Diagrama UML de la clase PodRS232Panel 53
5.13 Diagrama UML de la clase UART (UART.java). 55

6 **Uso académico de la aplicación**. ... 57
6.1 Historia ... 57
6.2 Introducción. ... 57
6.3 Aspectos importantes de la comunicación. ... 58
6.4 Descripción del estándar RS-232. ... 58
6.5 Modos de transmisión. ... 63
6.6 Características Eléctricas ... 65
6.7 Características Mecánicas ... 66
6.8 Características Funcionales. ... 68
6.9 Transmisión Serial. ... 69
6.10 Conexionado DTE-DTE: Null Módem. ... 73
6.11 Contro de Flujo. ... 76

7 **Conclusiones y líneas futuras**. ... 83

8 **Pasos para realizar un archivo ".jar"**. ... 85

9 **Pasos para poder firmarlo con un certificado digital** ... 87

Anexo A. ... 91
Manual de usuario. ... 91
A. Introducción ... 91
B. Objetivos ... 91
C. Anexo ... 91
D. Página Principal ... 92
E. HyperTerminal ... 93
F. Osciloscopio ... 95
G. Pod RS-232 ... 96

Bibliografía ... 97

Anexo B ... 99
B.1 RS-232. ... 101
B.2 UART. ... 105
B.3 HyperTerminal ... 111
B.4 Comunicaciones. ... 131
B.5 Preferencial del Terminal. ... 149
B.6 Pod RS-232 Panel. ... 159
B.7 POD RS-232. ... 167
B.8 Ayuda POD. ... 171
B.9 Pantalla Osciloscopio. ... 175
B.10 Osciloscopio. ... 186

Anexo C. ... 193
H. Cuestionario 1: ... 193
I. Cuestionario 2: ... 194

Capítulo 1.

1 Introducción.

Se plantea el desarrollo de una herramienta didáctica para la docencia de Fundamentos de Telemática, donde una de las prácticas a desarrollar en el laboratorio consiste en Comunicaciones Serie Asíncronas, en concreto la comprensión de los principios fundamentales del puerto serie RS-232.

Hasta el momento se da al alumno un enunciado de prácticas, y éste las realiza en laboratorio con instrumentos reales. Se pretende dotar de una alternativa con instrumentos virtuales (simulados) y el valor añadido de una ayuda contextual integrada.

En este proyecto se ha realizado un simulador de comunicaciones serie asíncronas, herramienta que servirá para la docencia en la Escuela Técnica Superior de Ingeniería de Telecomunicación (ETSIT) de la Universidad Politécnica de Cartagena (UPCT), disponiendo además del código fuente de la herramienta.

¿Qué es un Applet? Un Applet es una aplicación java enfocada a correr principalmente en un navegador Web, un *Java Applet* es un código *JAVA* que carece de un método main, por eso se utiliza principalmente para el trabajo de páginas Web, ya que es un pequeño programa que es utilizado en una página *HTML* y representado por una pequeña pantalla gráfica dentro de ésta.

Capítulo 2.

2 Objetivos.

El objetivo de este Proyecto Fin de Carrera es implementar una aplicación didáctica que consistente en una simulador de Comunicaciones Serie Asíncronas, partiendo como base del aprendizaje realizado en el laboratorio de prácticas de la asignatura de **Fundamentos de Telemática**. Con ello se conseguirá una aplicación de la que se dispondrá el código fuente, y así en un futuro mejorar el autoaprendizaje por parte de los alumnos respecto a la asignatura:

- Facilitar la comprensión del programa mediante una aplicación Web que ayudará al alumno a poder usar dicha herramienta cuando lo necesite, viendo así la utilidad y la función de todos los componentes del programa.

- Modernización: se ha realizado una interfaz gráfica (Applet), haciendo más agradable el entorno visual al alumno.

- Corregir ciertos aspectos teóricos para facilitar la comprensión de las Interfaces **RS-232**, así como los distintos tipos de **Comunicaciones** (tanto, *síncrona* como *asíncrona*).

- En la ventana de **POD RS-232** del Applet, si se ha configurado el *Hyperterminal* la opción de **Comunicaciones**, si esta marcada la opción ***Xon/Xoff*** y en el **POD RS-232** sólo tenemos conectados los pines **2** y **3**, tanto en el *emisor* como en el *receptor* veremos que hay comunicación entre ellos, adaptando así al simulador aún más realidad de lo que es una Comunicación Serie Asíncrona Hardware.

- La portabilidad del programa: este consistirá en un archivo tipo "*.jar*", que se podrá ejecutar en cualquier sistema operativo, como Windows, Linux,……, etc.

- El software está desarrollado en la propia Universidad Politécnica de Cartagena, sin tener que pagar ningún tipo de licencia o tener problemas con ésta, y al ser software de la propia Universidad, estará en español, facilitando así la comprensión al alumno.

- El código se dejará abierto para posibles mejoras futuras del programa.

- Conocer las distintas líneas del interfaz **RS-232**.

- Mostrar la necesidad del control de flujo.

- Analizar y comprender los distintos mecanismos del control de flujo.

- Manejar un programa de comunicaciones del entorno PC. Transmisión de ficheros.

Capítulo 3.

3 Análisis y Diseño.

A continuación se comentarán brevemente las características principales del programa de partida, en concreto la Práctica 1 de Fundamentos de Telemática: Comunicaciones Serie Asíncronas, características que se verán en profundidad más adelante en el simulador implementado en este proyecto:

3.1 Elementos que intervienen.

- 2 PC's que actúan como DTE's con entorno Windows interconectados mediante los respectivos puertos serie.
- Un pod de RS-232, intercalado en paralelo en la línea que une los dos DTE´s.
- Un PC conectado al pod RS-232 que actúa como analizador de puerto serie.
- 1 osciloscopio digital.
- Programa de gestión del puerto serie (**Terminal** de Windows).

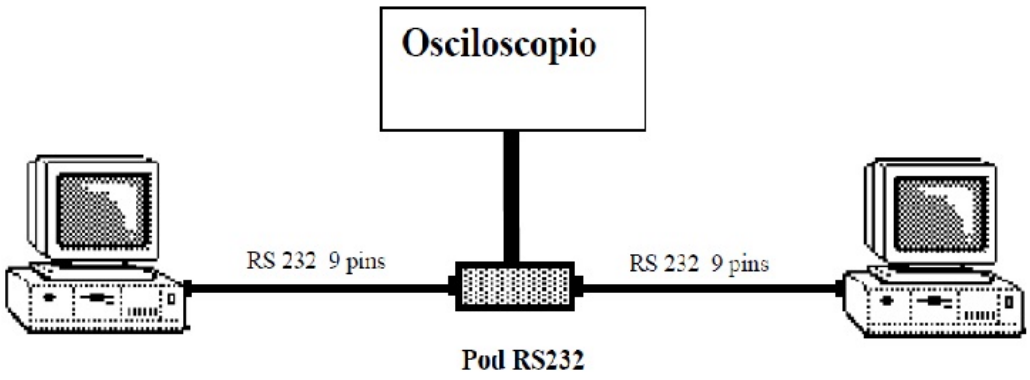


Figura 1. *Configuración del puesto de trabajo.*

3.2 *El interfaz RS-232.*

Uno de los interfaces de nivel físico más utilizados es el RS-232 que fue estandarizado por EIA en 1962. El interfaz consta de 25 contactos formando circuitos que trabajan en modo no balanceado, manejando todos ellos una masa común. La transmisión de información se realiza en serie tanto en modo asíncrono como síncrono.

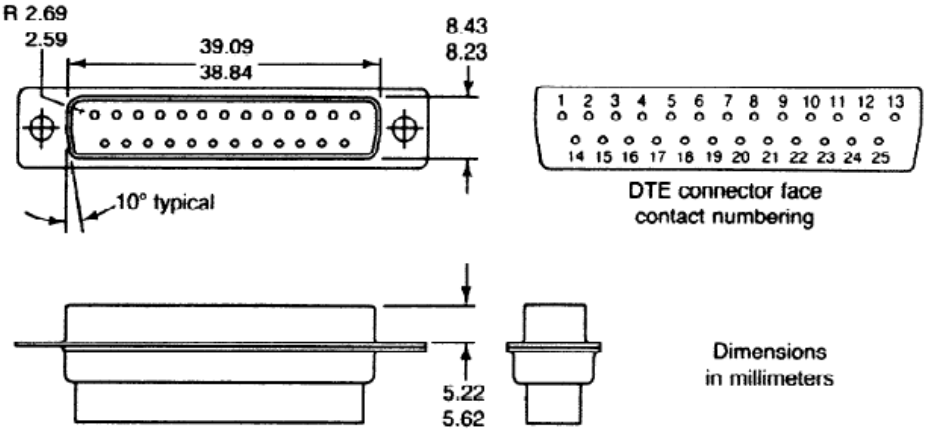


Figura 2. *Especificaciones mecánicas del interfaz **RS-232.***

Número del circuito de enlace	Pin	Nombre de la señal	Del/hacia el ETCD	Tipo señal	DESCRIPCIÓN
102	7	signal ground	-	Tierra	Tierra de señalización o retorno común.
103	2	TXD	hacia	Datos	Transmisión de datos de terminal local hacia el módem remoto
104	3	RXD	del	Datos	Recepción de los datos demodulados desde el terminal remoto
105	4	RTS	hacia	Control	Petición de transmisión
106	5	CTS	del	Control	Preparado para transmitir
107	6	DSR	del	Control	Aparato de datos preparado. Índice que el módem está preparado para recibir o transmitir datos
108.1	20	DTR	hacia	Control	Conecta el módem a la línea
108.2	20	DTR	hacia	Control	Terminal de datos preparado. El terminal está listo para transmitir y recibir datos
109	8	DCD	del	Control	Detector de señales de líneas recibidas por el canal de datos. El módem genera una señal hacia el terminal de datos cuando detecta un nivel de portadora aceptable
110	13	Signal Quality	del	Control	Detector de la calidad de la señal de datos. Cuando la señal es **ON** la calidad de la señal es aceptable. **OFF** baja calidad
111	23	Data Signal Rate	hacia	Control	Selector de velocidad binaria del equipo terminal de datos
112	23		del	Control	Selector de velocidad binaria del módem
113	24	DAA	hacia	Tempori.	Temporización para los elementos de señal en la transmisión. Señal de Temporización generada por un elemento externo al módem
114	15	DB	del	Tempori.	Temporización para los elementos de la señal en la transmisión. Señal de Temporización generada por el módem para sincronizar los datos a transmitir TXD
116	14	Select Standby	hacia	Control	Conmutación de seguridad en modo directo. **OFF** línea punto-punto. **ON** línea RTC
117	16	Standby Indicator	del	Control	**OFF** línea punto-punto. **ON** línea RTC
118			hacia	Datos	Transmisión de datos por el canal de retorno
119			del	Datos	Recepción de datos por el canal de retorno

Figura 3. *Descripción de los circuitos RS-232 DB 25 y sus funcionalidades.*

En la Figura 3. se muestran las especificaciones funcionales del RS-232-D para los conectores DB 25 respectivamente. Los circuitos se pueden agrupar en circuitos de datos que mantienen una comunicación full-dúplex (simultánea en ambos sentidos) y catorce circuitos de control.

Préstese especial atención a los circuitos comprendidos entre el 102 y el 109.

Mediante este interfaz se puede transmitir información tanto en modo asíncrono como síncrono.

La transmisión asíncrona transmite caracteres de forma individual. El carácter de información junto con unos bits de control forman una trama cuya longitud oscila entre 7 y 10 bits. Esta trama permite al receptor sincronizarse, y saber cuando se inicia y finaliza la transmisión de la información. En este tipo de comunicaciones digitales el calificativo de asíncrono es debido a que el tiempo entre dos tramas contiguas es arbitrario.

3.3 UART.

En los equipos de datos las tareas relacionadas con la transmisión asíncrona normalmente son generadas por la UART (Universal Asynchromous Receiver/Transmiter) independizando al procesador central y al programa de la problemática de la comunicación. La UART se puede ver como un bloque funcional que recibe/entrega información del/al canal vía una comunicación asíncrona RS-232.

3.4 Control de flujo.

Durante la transferencia de información entre dispositivos, puede ocurrir que la velocidad de procesado de uno de ellos sea inferior a la del otro lado, con lo que se podría perder parte de dicha información sino se utiliza algún mecanismo de arbitraje. Este mecanismo es precisamente el control de flujo.

3.5 Null-Módem (supresor del módem).

Generalmente el interfaz RS-232 interconecta un equipo terminal de datos ETD (Data Terminal Equipment) con un equipo terminal de circuito de datos ETCD (Data Communications Equipment) o módem. En entornos informáticos es usual conectar punto a punto vía RS-232 dos terminales asíncronos de datos (DTE), siendo necesario modificar algunos circuitos de datos y de control de interfaz. Al cable que realiza tal tipo de conexión se denomina null-módem.

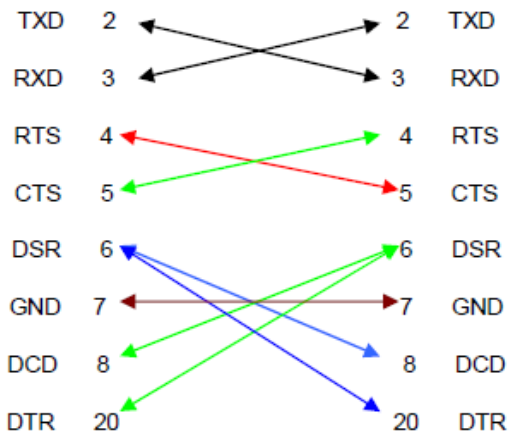


Figura 4. *Null-Módem.*

3.6 X-MODEM

X-Módem es un protocolo ARQ (automatic repeat request, solicitud de Recepción automática) de parada y espera (Stop&Wait) con un campo de datos de longitud fija (128 bytes). La trama del protocolo consta de cinco campos. Empieza con una cabecera de inicio de trama cuyo contenido es SOH (Start of Header) de valor 01H. El segundo campo indica el número de secuencia de la trama y su valor se inicializa a 1. El contenido del tercer campo es el complemento a 1 del número de secuencia de la trama.

Sincronismo SOH	Control secuenciamiento	Complementario Control Secuenciamiento	Datos	Checksum/CRC
1 byte	1 byte	1 byte	128 bytes	1/2 bytes

Figura 5. *Trama X-Módem.*

3.7 Control de dispositivos usando el puerto serie

Aquí usaremos los conocimientos adquiridos para realizar un caso práctico real en el que es necesario conocer el puerto serie. Se trata de controlar un dispositivo desde el puerto serie de un PC. En este caso será el osciloscopio digital que hemos usado para las prácticas. Este osciloscopio dispone un puerto serie para la comunicación con otros dispositivos y mediante el envio de comandos a través de este puerto serie podemos manejar el osciloscopio desde un PC. Para comunicarlos con el osciloscopio usaremos el programa terminal.

3.8 Determinación del cable a usar

Lo primero que debemos hacer para realizar la conexión entre PC y dispositivo, en este caso el osciloscopio, es determinar si el dispositivo funciona como ***ETD*** o como ***ETCD*** (como PC o como módem). Una vez realizado lo anterior se realizará la conexionado entre PC y dispositivo, usando el *POD RS-232* con las conexiones necesarias, en el caso de que fuese necesaria una conexión ***null-módem***.

3.9 Configuración del dispositivo

Lo siguiente que realizaremos es la configuración del dispositivo y el PC con los mismos parámetros de comunicación.

Para ello, consultaremos el manual de comunicaciones del osciloscopio adjunto en el CD de este proyecto, se seleccionará una de las configuraciones posibles y por último pondrá esa misma configuración en el programa terminal del PC.

3.10 Comunicación y control del dispositivo

En el capítulo 4 del manual de comunicaciones del osciloscopio hay un listado y explicación con ejemplos de cada uno de los comandos que podemos usar para controlar el osciloscopio. Usando esta información se deberá enviar los comandos oportunos por controlar como mínimo:

- Activación de canales.
- Control de la base de los tiempos.
- Control de voltios/división de cada canal.
- Posición vertical de cada canal.
- Modo DC, AC y GND de cada canal.

Nota: *El formato de los mensajes enviados se encuentra explicado con detalle en el capitulo 3 del manual de comunicaciones del osciloscopio.*

3.11 Estudio del Nivel Físico

El nivel físico es el nivel más bajo de la arquitectura de protocolos ***ISO/OSI***. Tiene por misión la transmisión de un flujo de bits entre dos entidades sin interpretar su contenido, siendo esta misión responsabilidad de los niveles superiores.

Los protocolos de nivel físico describen cuatro tipos de características:

- Eléctricas.
- Mecánicas.
- De procedimiento.
- Funcionales.

Las características **eléctricas** definen niveles lógicos y de tensión, tiempos, compatibilidad eléctrica entre interfaces y velocidades de transmisión. Las **mecánicas**, el dimensionado de interfaz, el tipo de conector y el número de contactos.

Las especificaciones **funcionales** definen el modo de operación de los circuitos clasificándolos en circuitos de datos, control, temporización y masas. Finalmente, las especificaciones de **procedimiento** indican las secuencias de control y datos para establecer, mantener y liberar la comunicación.

3.12 Estudio del Nivel de Enlace

Hasta ahora, nos hemos limitado a la transmisión de la información sin preocuparnos de lo que ocurre en el caso de que se produzcan errores durante la comunicación. En este apartado nos dedicaremos a la transmisión fiable de información, utilizando los mecanismos de nivel de enlace que proporciona el ***X-Módem***: entramado, control de flujo y de errores.

CAPITULO 4.

4 ÁMBITO DE APLICACIÓN.

4.1 Elección del lenguaje de desarrollo.

Se han barajado los siguientes lenguajes para desarrollar la implementación del programa:

4.1.1 C

El lenguaje por excelencia de programación de sistemas. Es un lenguaje de nivel medio (*no se puede decir que sea un lenguaje de alto nivel, por incorporar muchos elementos propios del* ***ensamblador***), tremendamente ligado a **UNIX** (*aunque es portable y hay compiladores para casi cualquier sistema operativo*). Casi podríamos considerarlo como un ensamblador estructurado y portable. Difícil de aprender (*no es recomendable como primer lenguaje, como sí lo podría ser Pascal*), aunque tremendamente flexible. Bastante dado a errores, sobre todo entre programadores novatos.

4.1.2 C++

Es una evolución sobre el lenguaje de programación de **C**. **C++** casi se puede considerar de alto nivel. Es orientado a objetos, relativamente difícil de aprender (*muchas características, muy complicado*), pero combina la potencia y flexibilidad de **C** con el valor añadido de **C++** es orientado a objetos. Bastante utilizado. Dado a errores, aunque no tantos como en **C**.

4.1.3 Java

JAVA es el lenguaje de "*moda*", de sintaxis parecida al **C**. Orientado a objetos (*JAVA te obliga, C++ sólo te da la posibilidad*), mucho menos flexible que **C++**, pensado para hacer aplicaciones interactivas más que controladores de dispositivos y sistemas operativos. Mucha aceptación popular, muchos recursos, y posibilidad de incluir programas de **JAVA** en páginas **HTML** (*los llamados "****Applet****"*).

JAVA es un lenguaje multiplataforma, los programas JAVA se pueden ejecutar en cualquier plataforma soportada (*Windows, Unix, Mac, etc.*), además es un lenguaje compilado e interpretado, ya que el compilador produce un código intermedio independiente del sistema llamado ***bytecode***. Se necesita instalar en el ordenador la **JVM** (*Java Virtual Machine*), que es el intérprete que convierte el *bytecode* en código máquina. ***Sun Microsystems*** distribuye de forma gratuita el producto base, el llamado **JDK** (*Java Development Kit.*), también llamado **J2SE** (*Java 2 Standard Edition*), y se puede encontrar en la siguiente dirección: ***http://java.sun.com/***.

4.1.4 Elección asumida

Finalmente me he decantado por el lenguaje de programación **JAVA** para la implementación del programa por los siguientes motivos:

i. Se ha elegido en detrimento de **C** al ser un lenguaje orientado a objetos y por su portabilidad.

ii. Frente a **C++** tiene la ventaja de tener más portabilidad, se puede usar las clases compiladas en cualquier plataforma (*Windows, Unix, etc.*).

iii. Una razón muy importante fue el entorno gráfico resultante, Ya que *JAVA* proporciona elementos gráficos muy atractivos, sobre todo utilizando las componentes ***swing***.

4.2 Elección del entorno de desarrollo.

Para el lenguaje de programación de *Java* hay múltiples entornos de desarrollo, entre los que se han barajado los siguientes:

4.2.1 JBuilder

Dirección: (*Borland*) ***http://www.borland.com/jbuilder/***
Versión actual: 6.0
Plataformas: Windows, Linux, Solaris.
Licencia: La versión de evaluación, la personal, es gratis, las avanzadas, profesionales y Enterprise son de pago.

Jbuilder Foundation está diseñado para desarrolladores *Java* que quieren una alta productividad **IDE** (*Entorno de Desarrollo Integrado*) para crear más fácilmente aplicaciones multiplataforma para Linux, Solaris y Windows.

Finalmente, *Jbuilder* permite que los usuarios retoquen a su gusto y extiendan el entorno según sus necesidades de desarrollo usando **Open Tools API**, la cual facilita la integración de otros componentes adicionales.

Tiene la gran utilidad de visualizar los diagramas **UML**, además de poder desarrollar aplicaciones **Web** con **JSP** y **servlets**.

4.2.2 Kawa

Dirección:
Versión actual: 5.0
Plataforma: Windows.
Licencia: Como es habitual, las versiones profesionales y Enterprise son de pago. Pero también disponen de versiones de evaluación.

Kawa es un entorno de múltiples ventanas. Se parte de un proyecto, lleva dos módulos, uno con soporte ***HTML*** y otro para ***JAVA***. Se pueden visualizar las clases, variables…
Sencillo y potente.

4.2.3 NetBeans

Dirección: ***http://www.netbeans.org/***
Versión actual: 7.0.1 (Build 201107282000)
Plataforma: Todas las plataformas con **JVM**.
Licencia: Opensource.

NetBeans es una aplicación *Opensource* (*el código del entorno está abierto a posibles modificaciones*) de desarrollo escritas en *Java*, esto quiere decir que se puede modificar el entorno de acuerdo a ciertos parámetros de licencia. Soporta, aparte de *Java*, otros lenguajes, como *C++*.

NetBeans es la mejor opción a la hora de desarrollar en *Swing* y *J2EE/EJB 3.0*.

Algunas de sus funciones y características son:

- Completado de código.
- Soporte para escritura de *servlets*.
- Ayudas con el código y ayuda on-line.

El *Entorno de Desarrollo Integrado* (***IDE***) *NetBeans* es un entorno de programación de varios lenguajes, incluyendo a *Java* y a *C++*. Este desarrollo es de fuente abierta, es decir, se proporciona el código fuente del entorno para que se pueda modificar de acuerdo a ciertos parámetros de licencia.

Es un entorno muy cómodo en cuanto a interfaz gráfica se refiere, ya que ahorra al usuario escribir código tan sólo con arrastrar los componentes gráficos que soporta *Java* y que están representados mediante iconos. Al añadir estos componentes, en la aplicación se genera el código correspondientes a dicha inclusión del elemento gráfico. Está recomendado por *Sun Microsystems*, aunque una de sus desventajas es que ocupa muchos recursos de memoria.

4.2.4 Eclipse

Dirección: ***http://www.eclipse.org.***
Versión actual: Java 2 Technology Edition 5.0, SR4 (see caveat below).
Plataforma:

Eclipse es en el fondo, únicamente un armazón (*workbench*) sobre el que se pueden montar herramientas de desarrollo para cualquier lenguaje, mediante la implementación de los *plugins* adecuados.

La arquitectura de *plugins* de Eclipse permite, además de integrar ciertos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, etc.

4.2.5 Elección asumida

El entorno principal elegido es el ***NetBeans***. Por su facilidad de uso y gran potencia (aunque consuma demasiada memoria.), es muy ideal para implementar distintos algoritmos, además de ser un entorno de ventanas, mediante el cual la programación se hace mas llevadera.

Para el diseño de la interfaz gráfica, también me he decantado por ***NetBeans***, también por su gran facilidad de uso.

Finalmente, se ha decidido por ***NetBeans*** para realizar los diagramas ***UML*** (*instalando los plugins necesarios.*), de las clases que componen el programa, adjunto dirección de descarga de plugin **UML** para Netbeans ***http://netbeans.org/downloads/6.9.1/zip.html***, (ver Word netbeasn frente a eclipse)

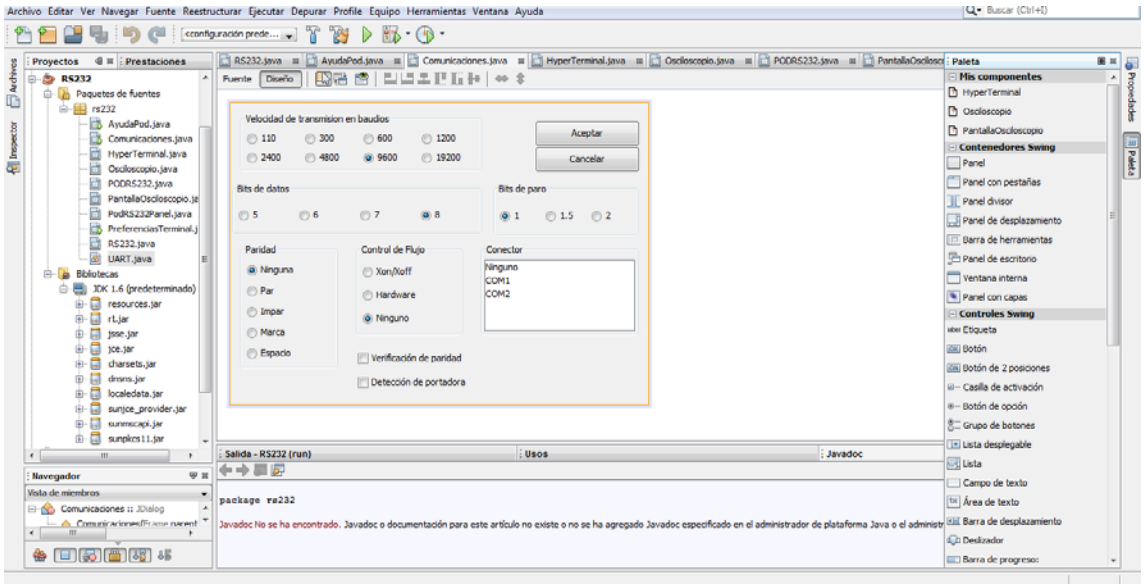
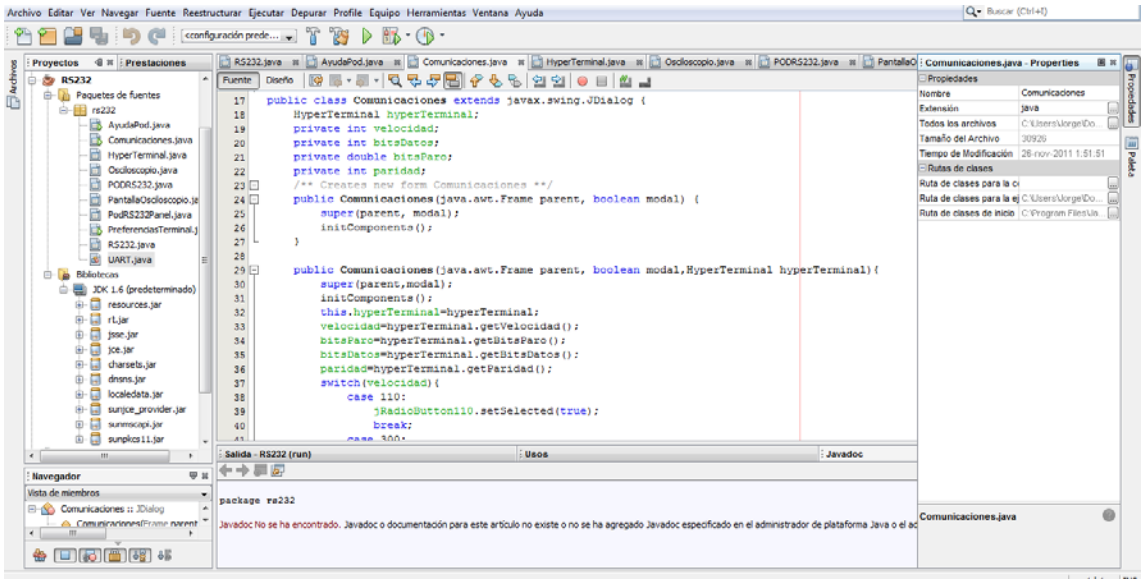


Figura 6. *Entorno de Diseño de NetBeans.*



Capítulo 5.

5 Implementación de la aplicación.

5.1 Página de Inicio.

La aplicación comienza en un diseño Web (disponible en la siguiente dirección http://labit501.upct.es/~jorge/RS232.html) en la cual se muestra un entorno en la cual podremos elegir entre 3 opciones: *EJECUTAR APPLET*, *PDF de la práctica*, y una pestaña con el *MARCO TEÓRICO* de la práctica.

Si se elige la opción de *EJECUTAR APPLET*, nos mostrará el entorno del Applet a ejecutar, y a partir de eso momento empezar a realizar la práctica en cuestión.

Si se elige la opción de *PDF de la práctica*, esta opción nos da la posibilidad de descargarnos el enunciado de la práctica en versión *.pdf* para así poder realizar las prácticas.

Si se elige la opción de *MARCO TEÓRICO*, te redirige a una página Web, en la cual te muestra el enunciado de la práctica en versión on-line, por si no quiere descargarse el enunciado de la práctica en versión *pdf*.

La clase que implementa el escritorio virtual del Applet es la clase principal del programama '*RS232*', que previamente se ha señalado como 'main-class' mediante el archivo *MANIFEST.MF* . Con posterioridad se se explicará con todo detalle cómo se construye el archivo '*.jar*' final. Esta es una clase que se extiende de **JApplet**, en la cual se ha insertado un elemento **JDesktopPane**, y a su vez insertado 3 tipos de elementos que componen el escritorio virtual principal del Applet: dos elementos hyperTerminal (que es un componente **JInternalFrame**), un elemento osciloscopio (que es un componente **JPanel** y se genera dinámicamente con un elemento **JInternalFrame**) y un último elemento PODRS232 (que también tiene un componente **JPanel** y se generan dinámicamente con un componente **JInternalFrame**).

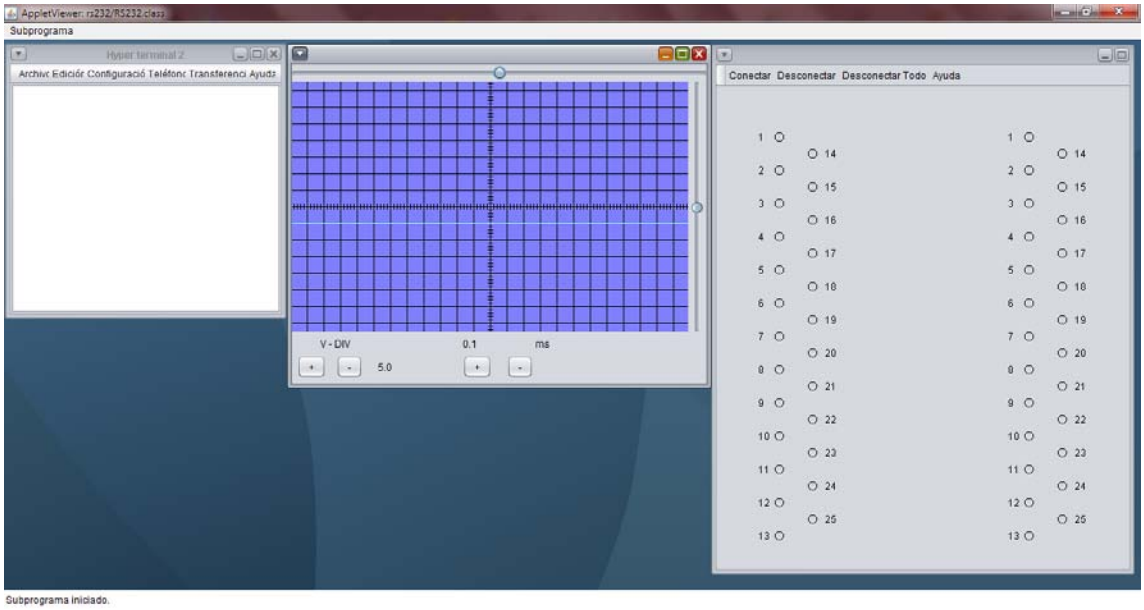


Figura 7. *Entorno Virtual del Applet.*

Por ejemplo, el PODRS232, en este elemento lo primero que se ha hecho es crear un componente **JPanel** que lo generamos dinámicamente mediante un componente **JInternalFrame**, el mótivo de hacer esto es que no podemos dibujar directamente dentro de un **JPanel** sino que tendremos que dibujarlos en un componente **JInternalFrame** y una vez dibujados ahí, ya podremos insertarlos en el **JPanel** y así ya esta dibujada la clase PODRS232, que es la que se muestra en el escritorio virtual del Applet. En este método se hará lo siguiente:

```
private void initComponents() {

        jDesktopPane1 = new javax.swing.JDesktopPane();
        pODRS2321 = new rs232.PODRS232();

        pODRS2321.setVisible(true);
        pODRS2321.setBounds(610, 10, 519, 638);
        jDesktopPane1.add(pODRS2321, javax.swing.JLayeredPane.DEFAULT_LAYER);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jDesktopPane1,javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 939, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jDesktopPane1,    javax.swing.GroupLayout.DEFAULT_SIZE,    559,
Short.MAX_VALUE)
        );
    }
```

En el manejador ocurren tres cosas:

I. Se crea un objeto del tipo `rs232.PODRS232()`, es decir, un **JInternalFrame** con todas las funcionalidades para poder crear la imagen del PODRS232.

II. Dicho **JInternalFrame** creado, se muestra en el **JPanel** por medio del método `setVisible(true)`.

III. Y mediante `jDesktopPane1.add(pODRS2321, javax.swing.JLayeredPane.DEFAULT_LAYER);` insertamos en el **JDesktopPane**, y así mostraremos en el Escritorio virtual el elemento PODRS232.

De esta manera, todos los elementos del escritorio virtual tienen asociados un manejador que creará y mostrará todos los **JInternalFrame** del tipo que corresponden según el componente que creamos, y así finalmente se podrán crear tanto el PODRS232 como el OSCILOSCOPIO.

A continuación, pasaremos a explicar la implementación de todas las opciones del programa.

5.2 Escritorio Virtual.

Este Escritorio Virtual presenta 3 componentes las cuales explicaremos mas adelante con mas detalle, estos componente son: OSCILOSCOPIO, HyperTerminal y PODRS232.

5.3 HyperTerminal.

El HyperTerminal es un programa que puede utilizarse para conectar con otros equipos, sitios *Telnet* de Internet, servicios de boletines electrónicos, servicios en línea y equipos host con módem o un cable módem nulo.

Lo primero que nos encontramos al ejecutar el Applet es el módulo HyperTerminal, mediante la cual se pueden intercambiar información y transferir archivos entre distintos ordenadores o dispositivos que dispongan de conexionado serie RS-232.

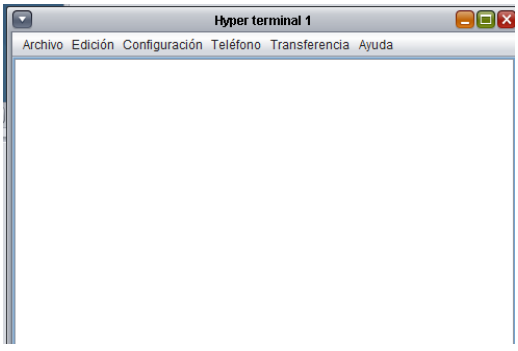


Figura 8. *HyperTerminal*.

Al ejecutarse aparecerá una ventana como la anterior, en la que se observa la línea con las opciones del menú y la zona del terminal en la que se pueden escribir caracteres desde el teclado que serán enviados vía puerto serie al equipo remoto, y también en esta zona aparecerán los caracteres que recibamos del terminal distante.

Uno de los momentos mas importantes en el HyperTerminal es la *configuración* de la misma. Para poder utilizar correctamente el programa terminal, se han de configurar previamente los parámetros de comunicación serie, así como el puerto que se usará.

Deben conocerce a priori los parámetros de transmisión que utilizará es dispositivo que está conectado a vuestro ordenador, en caso contrario no será posible la comunicación: éstos son: velocidad de transmisión, bits de datos, bits de parada, tipo de paridad, y control de flujo. En el caso de que se conecte un **módem**, es muy probable que éste se adapte automáticamente a cualquier velocidad que fijemos.

Con los datos anteriores claramente explicados, procederemos a configurar el programa, accediendo en el menú de **Configuración** a la opción de **Comunicaciones**, con lo que se presenta la siguiente ventana:

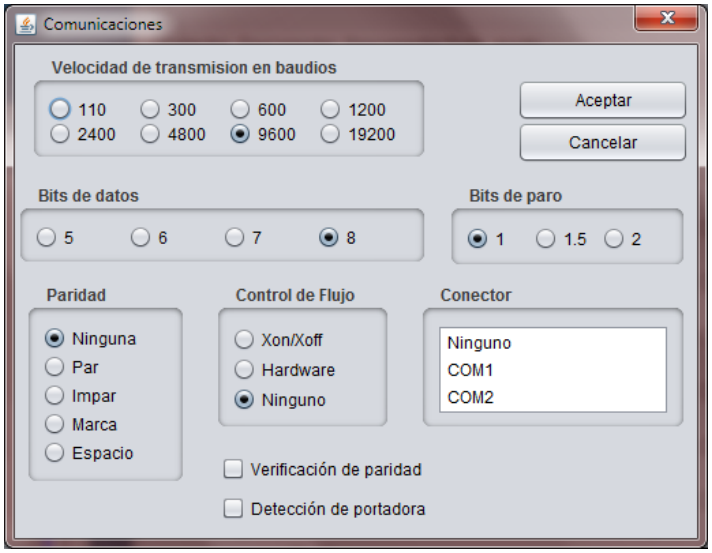
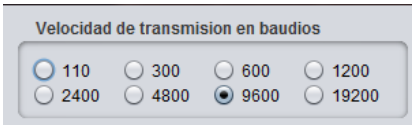


Figura 9. *Comunicaciones.*

La opción de verificación de paridad realiza la comprobación de errores en caso de que se produzcan debido a que la paridad de los bits recibidos no corresponden con la que se indicó. La opción de detección de portadora consiste en escuchar la linea CD (*Carrier Detect*) del interfaz serie; y no permitir enviar nada por el puerto serie hasta que esté activa. Es útil con módems para evitar que se pierdan caracteres cuando se intenta transmitir algo y el módem todavía no se ha conectado con el otro extremo (modem remoto).

- **Velocidad de transmisión en Baudios.**



La velocidad de transmisión es la relación entre la información transmitida a través de una red de comunicaciones y el tiempo empleado para ello. Cuando la información se transmite *digitalizada*, esto implica que está codificada en bits (unidades en base binaria), por lo que la velocidad de transmisión también se denomina a menudo tasa binaria o tasa de bits (bit rate).

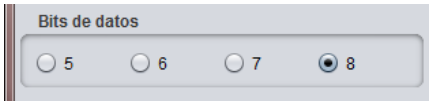
La velocidad de transmisión a través de un canal de comunicaciones hace referencia al número de bits transmitidos por unidad de tiempo, pero esto incluye

también la información contenida en las cabeceras de los protocolos empleados para transmitir la información entre equipos.

El número de bits transmitidos por segundo, utilizados como medidas de velocidad a la que un dispositivo, como un módem, puede transferir datos. Un carácter se compone de 8 bits. En la comunicación asíncrona, cada carácter puede estar precedido por un *bit de inicio* y terminar con un *bit de parada.* Por lo tanto, por cada carácter se transmitirán 10 bits. Por ejemplo, si un módem se comunica a 2400 bits por segundo (bps), se enviarán 240 caracteres por segundo.

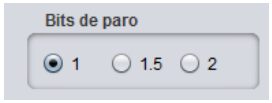
- **Bits de datos.**

Los bits de *datos* son el número de bits de una palabra. La mayor parte de los sistemas usan ocho bits para representar un caráter de datos individual (*ASCII extendido*). En algunas ocasiones, algunos sistemas más antiguos siguen usando siete bits.



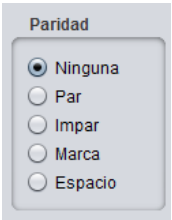
- **Bits de paro.**

Los bits de paro enmarcan los paquetes de datos de las comunicaciones asíncronas. Indican al módem de recepción que se ha enviado un byte. Los protocolos asíncronos actuales no requieren nunca más de un bit de parada.



- **Paridad.**

La paridad controla la forma en la que los módems controlan los errores. El ruido de la línea puede insertar bits adicionales en los datos que se están transmitiendo por la línea telefónica. En los tipos de modulación más antiguos, la paridad comprobaba estos errores. Con la comprobación de la paridad, el módem de transmisión agrega un *bit de paridad* al paquete de datos para que el nùmero de bits 1 en el paquete sea *impar* o *par*. El módem de recepción suma el número de bits 1 que ha recibido y acepta o rechaza el paquete en función de si la suma coincide con el *bit de paridad*.



Paridad	Para hacer
Par	Establece el bit de paridad a 0 o 1 para que el número de bits 1 sea par.
Impar	Establece el bit de paridad a 0 o 1 para que el número de bits 1 sea impar.
Ninguna	No envia bit de paridad .
Marca	Establecer el bit de paridad siempre a 1.
Espacio	Establecer el bit de paridad siempre a 0.

La mayor parte de la conexiones de módem utilizan ahora métodos más confiables y sofisticados para la comprobación de errores (consultar corrección de errores), con lo que, normalmente, suele estar establecido a ***ninguna***.

- **Control de Flujo.**

En una conexión hay implicados seis vínculos distintos: *equipo de transmisión* a *módem de transmisión*, *módem de transmisión* a *módem de recepción*, *módem de recepción* a *equipo* y el inverso de los tres vínculos. Todos ellos pueden tener distintas velocidades de transmisión de datos. Cuando el *módem de recepción* no puede aceptar datos temporalmente, necesita una forma de indicar al *módem de transmisión* que vaya más despacio o que le espere. El *control de flujo* es el método por el que un módem controla la velocidad a la que los restantes módems le envían los datos.

Si el control de flujo no está configurado correctamente, puede que le resulte imposible conectarse a un sistema remoto, la velocidad de transferencia puede disminuir considerablemente o puede interrumpirse la conexión.

Si ve muchos errores y retransmisiones de datos al descargar los archivos, compruebe la configuración del control de flujo del módem y del programa de comunicaciones. La configuración de **control de flujo** del programa de comunicaciones y del módem debe de ser la misma. Muchos programas de comunicaciones establecen esta configuración automáticamente, aunque algunos deben ser configurados por separado.

→ **Control de flujo por Hardware**: El control de flujo por Hardware (**RTS/CTS**) depende del módem para controlar el flujo de datos. Se debe usar con todos los módems de alta velocidad o con los módems que comprimen datos.

→ **Control de flujo por Software**: El control de flujo por Software (llamado también **Xon/Xoff** o **CTRL+S/CTRL+Q**) usa caracteres de datos para indicar que el flujo de datos debe iniciarse o detenerse. Esto permite a un módem enviar un carácter de control para indicar a otro módem que detenga la transmisión mientras se actualiza.

El control de flujo por software es más lento y, normalmente, menos conveniente que el control de flujo por hardware. El control de flujo por software se utiliza sólo para transmitir texto. No se puede utilizar para la transferencia de archivos binarios porque éstos pueden contener caracteres especiales de control de flujo.

La opción de **Preferencias del Terminal**, del menú de **Configuración**, permite fijar los parámetros de visualización:

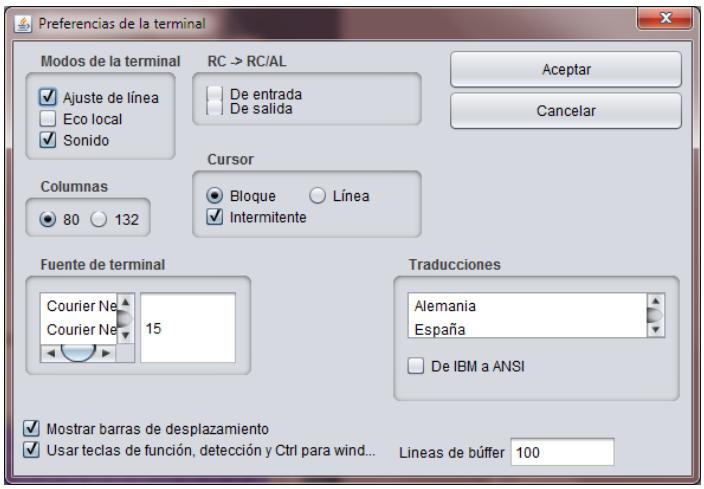
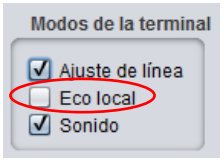


Figura 10. *Preferencias del Terminal*.

Los parámetros más interesantes de esta ventana son:

- **Eco Local.**

  Visualiza en la ventana del terminal todos los caracteres que se pulsan desde el teclado, para que podamos ver lo que escribimos.



  Nota: existen dispositivos, como los módems que también puede realizar ecos, es decir si se les envía un carácter, además de transmitirlo al otro extremo, también lo reenvían de nuevo hacia el terminal, de tal forma que si tenemos la opción anterior activada, veremos cada carácter doble.

- **RC -> RC/AL.**

  Convierte el código ASCII de retorno de carro (13) en los códigos ASCII de retorno de carro (13) + avance de línea (10). Algunos dispositivos, como ordenadores con sistemas operativos *Unix*, interpretan el código de retorno de carro (13) como una función compleja de retorno de carro y avance a la línea siguiente, mientras que en otros, como puede ser *MSDOS* o *Windows*, se requieren dos códigos diferentes, uno para cada función. Si se observa el efecto al pulsar la función *Return*, vuelve el cursor a la izquierda, pero no se baja una línea, sino que se sobreescribe siempre sobre la misma línea, conviene entonces, activar estas opciones adecuadamente.

El resto de parámetros del menú de **Configuración**, permite afinar la configuración del terminal en datalle. De forma resumida, las opciones del menú de configuración son, pero en la implementación del Applet no lo he contemplado:

- **Número de teléfono.**

  Permite especificar un número telefónico, el cual se marcará automáticamente cuando se acceda a la opción de Marcar del menú Telefónico.

- **Emulación del terminal.**

  Permite que el terminal se comporte como un terminal **TTY**, **VT-52** o **VT-100**. Éstos se caracterizan en que interpretan ciertos códigos de control que permiten moverse por la pantalla o cambiar los atributos de los caracteres (colores, subrayado, etc.).

- **Teclas de función.**

  Permite asignar a las teclas de función (**F1** a **F10**) secuencias de caracteres, de tal forma que al pulsar una tecla **F**, se reproducen automáticamente los caracteres asignados. Es útil para definir las secuencias de caracteres más comunes que utilicemos.

- **Transferencias de Texto.**

  Especifica la forma de transmitir archivos de texto, si el método de control de flujo utilizado, el ajuste de las líneas de texto y si el texto se transferirá carácter a carácter cada vez o toda una línea cada vez.

- **Transferencias Binarias.**

  Especifica el protocolo de transmisión de archivos binarios (***Xmodem*** con ***CRC*** o ***Kermit***).

- **Comandos del Módem.**

  Permite configurar con los comandos del módem las funciones básicas de éste como la secuencia de códigos para marcar un número telefónico o colgar la línea. Con estos parámetros se puede automatizar la conexión/desconexión del módem, utilizando el menú **Teléfono**.

El programa terminal puede ser utilizado con diferentes dispositivos, y cada uno de ellos requerirá una configuración específica. Como puede ser muy tedioso tener que configurar cada vez el terminal, el programa permite grabar las configuraciones en ficheros, de tal forma que sólo abriendo uno de estos ficheros, se configuran automáticamente todos los parámetros asociados.

- **Nuevo.**

  Permite crear un nuevo fichero de configuración. Inicialmente se utilizan los parámetros predeterminados.

- **Abrir.**

  Abre un fichero de configuración y ajusta todos los parámetros.

- **Guardar, Guardar como.**

  Permite grabar la configuración actual del terminal en un fichero.

- **Especificar impresora.**

  Permite indicar una impresora de Windows en la que se podrá imprimir todo lo que aparece en la ventana del terminal.

- **Menú Edición.**

  Influye las funciones básicas de copiar, cortar y pegar.

- **Menú Teléfono.**

  Permite automatizar las tareas de conexión y desconexión si se utiliza un módem.

- **Menú Transferencias.**

  Este menú si se ha implementado en el Applet, para realizar todas las opciones de transferencias de ficheros (texto o binarios), tanto *enviar* como *recibir*. La diferencia entre un fichero de texto y uno binario radica en que los ***ficheros de texto*** pueden ser interpretados, incluso modificados mientras se transmiten (por ejemplo, la conversión del código de retorno de carro y avance de línea o viceversa), mientras que lso ***ficheros binarios*** representan una secuencia de 1 y 0 que no deben ser modificados de ninguna manera, ya que representan algún tipo de información estructurada como imágenes, programas ejecutables, etc.

- **Enviar archivo de texto.**

  Permite enviar un archivo de texto. Aparece una ventana en la que pregunta por un nombre de fichero y si se desea realizar algún tipo de mapeo (***CR->CR/LF***, etc).

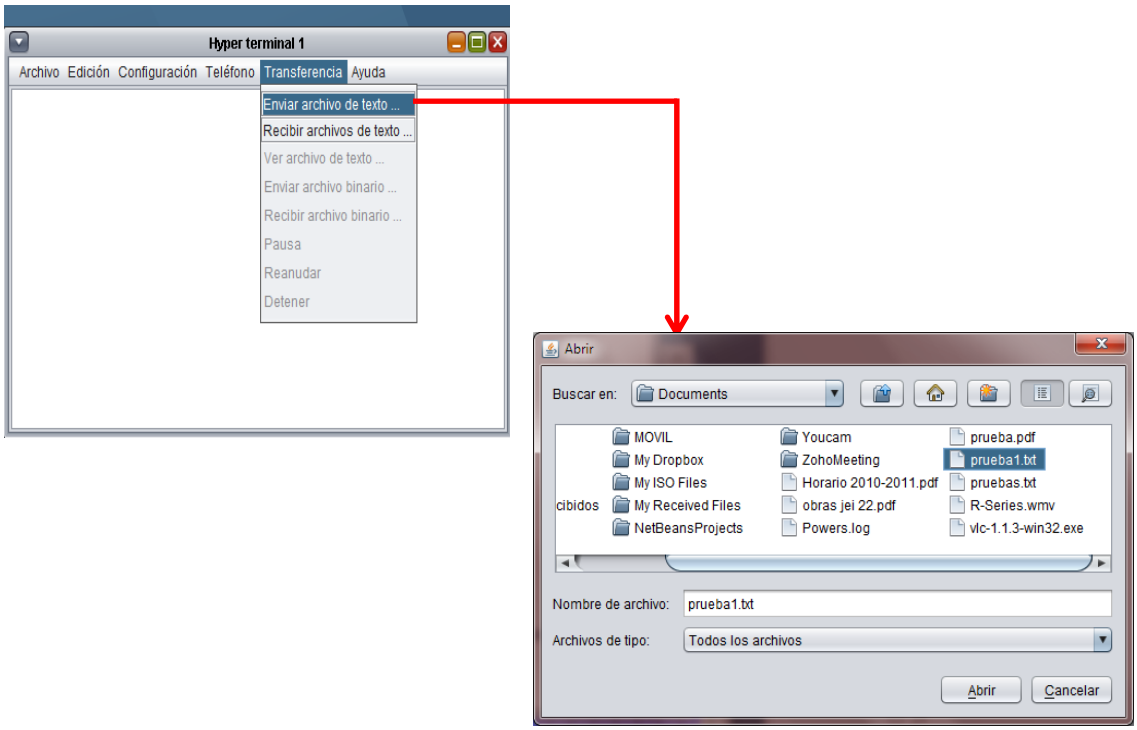


Figura 11. *Enviar archivo de texto.*

- **Recibir archivo de texto.**

Permite especificar el nombre de un fichero donde se recibirá un fichero de texto.

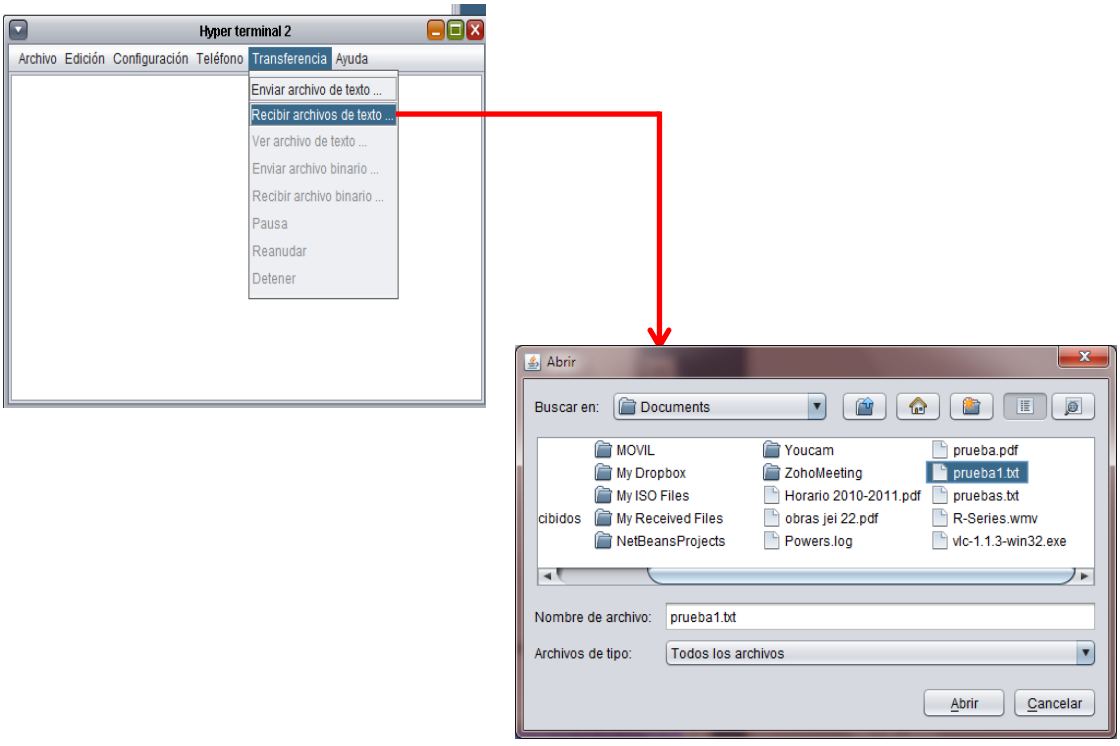


Figura 12. *Recibir archivo de texto.*

- **Ver archivo de texto.** (*no implementada en la aplicación*)

  Muestra por la ventana del terminal un fichero.

- **Enviar archivo binario.** (*no implementada en la aplicación*)

  Permite indicar el fichero que transmitiremos con un protocolo ***Xmodem*** o ***Kermit***, según lo hemos configurado.

- **Recibir archivo binario.** (*no implementada en la aplicación*)

  Permite indicar el fichero donde guardaremos los datos que recibiremos desde el puerto serie. ***Nota***. *El terminal que transmite el fichero debe tener configurado el mismo protocolo (**Xmodem** o **Kermit**), en caso contrario se producirán multiples errores.*

5.4 *Osciloscopio.*

La implementación del Osciloscopio se ha realizado de la siguiente manera: lo primero es que se ha programado la PantallaOsciloscopio.java (***JPanel***), en donde se realiza toda la programación de cómo pintar el osciloscopio (*o sea, la pantalla final donde se pinta la señal final que se esta transmitiendo de un Emisor a un Receptor*), y luego generarla dinámicamente en el escritorio virtual (***JApplet***) lo que lo hacemos es crear un elemento Osciloscopio.java (***JInternalFrame***) y ahí dejo caer un componente PantallaOsciloscopio, y así esta completa la ventana del osciloscopio que vemos en el Applet.

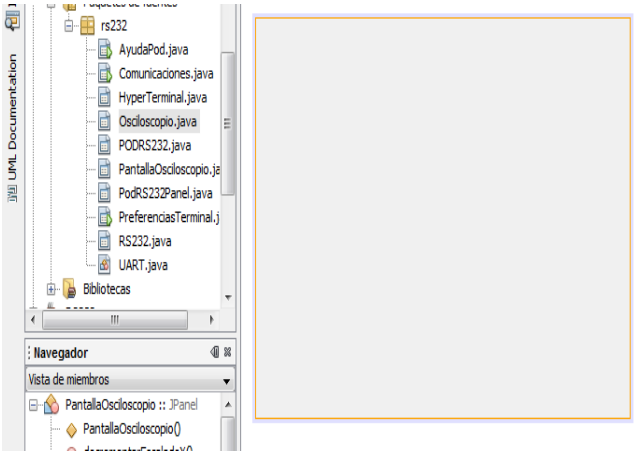


Figura 13. *JPanel, PantallaOsciloscopio en donde se realiza toda la programación del Osciloscopio.*

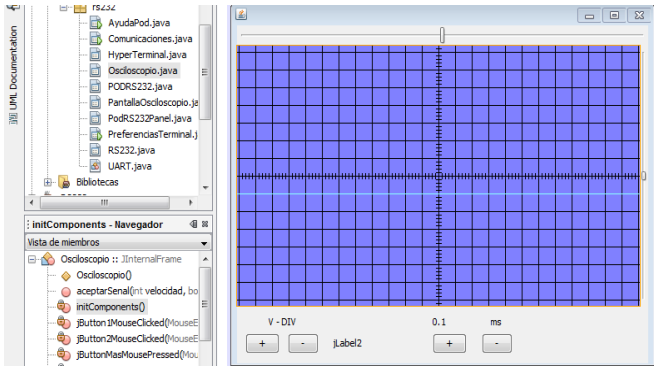


Figura 14. *JInternalFrame, Osciloscopio en donde se deja caer el elemento PantallaOsciloscopio (JPanel) en donde se pintan los ejes y las señales finales (físicamente).*

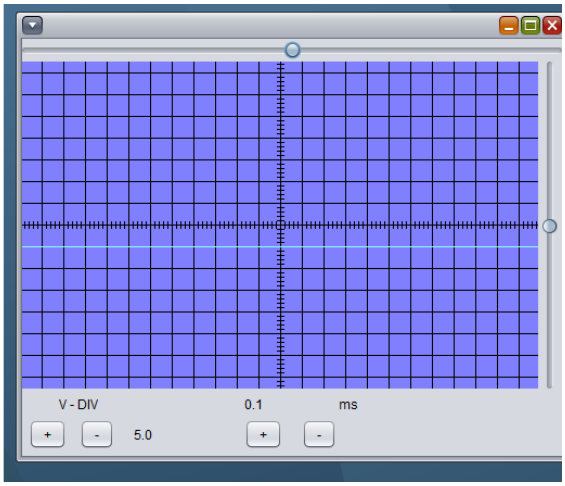


Figura 14. *Osciloscopio (Vista final)*

- **PantallaOsciloscopio.**

En el método paintComponent es donde se programa como se van a pintar las líneas '*rayitas*', mediante las cuales se divide la pantalla del Osciloscopio, para ser más específicos las *rayitas* verticales y horizontales para delimitar las señales que se muestren en la pantalla del Osciloscopio.

Por otra parte tenemos que entender el concepto de los componentes **Swing** en **JAVA**, los componentes **Swing** están '*dibujados*' sobre una ventana que le pedimos al sistema operativo, se puede decir que "*son de la máquina virtual de JAVA*" y no del sistema operativo. Esto no ocurrirá así con las componentes de la librería **AWT**, en la cual las componentes "*eran del S.O*", con lo que una de las ventajas de este hecho es un mayor control sobre la apariencia de los componentes.

Si las programamos con las antiguas librerias **AWT** sólo hay un componente sobre el cual podemos dibujar: ***Canvas***. Es el único objeto que el **S.O**, nos proporciona en el que podemos dibujar. Esto quiere decir, por ejemplo, que si no nos gusta la apariencia de un *scroll* o de un *botón* no podemos hacer nada para modificarla, hemos de aceptarla tal y como se nos da. Por este motivo una misma aplicación se veía diferente en los diferentes **S.O**, un *scroll* en Windows no tiene por que ser igual a un *scroll* en Linux o Mac. En las nuevas librerias **Swing** es posible hacer que una misma aplicación se vea igual en todos los **S.O**, mediante una característica de los componentes que se llama ***Look and Feel***.

Por otra parte en las librerias **Swing** es posible pintar sobre cualquier componente que derivede ***JComponent***, es posible por lo tanto modificar la apariencia de un *botón* o *scroll* a nuestro antojo (al márgen de los distintos *Looks and Feel disponibles*). De todas formas no es una buena programación la de pintar sobre cualquier componente. En la documentación de **JAVA** se aconseja que si queremos mostrar al usuario un dibujo se aconseja que lo hagamos siempre sobre un ***JPanel***.

El mètodo paintComponent sirve para cuando cualquier componente de las librerias *Swing* por cualquier razón necesita '*redibujarse*' en pantalla llama al método paintComponent. En este método se halla toda la información que se necesita para dibujar el componente en pantalla.

Causas externas a un componente que fuerza que este de '*redibuje*' son que la ventana en la que está se minimiza y luego se maximiza o que otra ventana se ponga encima de la ventana que lo contiene. Además el componentese dibujará cuando se crea y cuando se invoque el método ***repaint()***.

Cuando este método es invocado lo único que aparecerá en el componente es lo que se dibuje desde él, todo lo demás es "*borrado*". Este es el motivo por el cual en este proyecto al minimizar y luego maximizar la pantalla dejamos de ver lo que habíamos dibujado. Si queremos que siempre que el componente se redibuje aparecisen nuestros dibujos hemos de sobreescribir el método *paintComponent* y escribir el código necesario para realizar estos dibujos.

A continuación, pasaré a explicar el trozo de código del método paintComponent de la **pantallaOsciloscopio**:

```
public void paintComponent(Graphics g){
        this.ANCHO=this.getWidth();
        this.ALTO=this.getHeight();
        this.DIST_RAYITAS=2;
        this.EJEX=ALTO/2;
        desplazamientoHorizontal=(ANCHO*pHoriz)/100;
        desplazamientoVertical=(ALTO*pVert)/100;
        double factorx=9600.0/((double)velocidad);
        int i;
        int x,y;

        y=0;
        x=0;

        Color color=new java.awt.Color(128,128,255);
        g.setColor(color);
        g.fillRect(0, 0,ANCHO , ALTO);
        g.setColor(Color.BLACK);

        g.drawLine(0, ALTO/2, ANCHO, ALTO/2);
        g.drawLine(ANCHO/2,0,ANCHO/2,ALTO);

        //Rayitas de la graducion
        for(i=0;i<ANCHO/2;i=i+DIST_RAYITAS){
            g.drawLine(ANCHO/2+DIST_RAYITAS*i, ALTO/2-3, ANCHO/2+DIST_RAYITAS*i, ALTO/2+3);
            g.drawLine(ANCHO/2-DIST_RAYITAS*i, ALTO/2-3, ANCHO/2-DIST_RAYITAS*i, ALTO/2+3);
            if(i%5==0){
                g.drawLine(ANCHO/2+DIST_RAYITAS*i,0, ANCHO/2+DIST_RAYITAS*i, ALTO);
                g.drawLine(ANCHO/2-DIST_RAYITAS*i,0, ANCHO/2-DIST_RAYITAS*i, ALTO);
            }
        }

        for(i=0;i<ALTO/2;i=i+DIST_RAYITAS){
            g.drawLine(ANCHO/2-3, ALTO/2+DIST_RAYITAS*i, ANCHO/2+3, ALTO/2+DIST_RAYITAS*i);
            g.drawLine(ANCHO/2-3, ALTO/2-DIST_RAYITAS*i, ANCHO/2+3, ALTO/2-DIST_RAYITAS*i);
            if(i%5==0){
                g.drawLine(0, ALTO/2+DIST_RAYITAS*i, ANCHO, ALTO/2+DIST_RAYITAS*i);
                g.drawLine(0, ALTO/2-DIST_RAYITAS*i, ANCHO, ALTO/2-DIST_RAYITAS*i);
            }
        }

        color = new Color(140,255,255);
        g.setColor(color);

g.drawLine(0,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas
),desplazamientoHorizontal,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensi
onEntreRayitas));

        if(!senal[0]){

g.drawLine(desplazamientoHorizontal,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFE
CTO/tensionEntreRayitas),desplazamientoHorizontal,EJEX+desplazamientoVertical-
2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas));
        }

        for(i=0;i<senal.length;i++){
            x=i*(int)(((double)LONGITUD_BIT_DEFECTO*factorx*escaladoX));
            if(senal[i]){
                y=+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas);
            }
            else{
                y=-2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas);
            }
```

Desplazamiento horizontal en píxeles.

Desplazamiento vertical en píxeles.

Factor de estiramiento horizontal de la señal dependiendo de la velocidad de transmisión.

Dibujamos los ejes y la graduación de los mismos.

Fondo del osciloscopio.

Ejes del osciloscopio.

Rayitas de graduación.

Dibujamos la línea que precede a la señal.

Dibuja la línea vertical que une la señal por defecto con el inicio de la señal.

Dibujamos la señal.

```
            g.drawLine(desplazamientoHorizontal+x,                  desplazamientoVertical+y+EJEX,
desplazamientoHorizontal+x+(int)(((double)LONGITUD_BIT_DEFECTO*factorx*escaladoX)),
desplazamientoVertical+y+EJEX);
            if(i>0 && senal[i-1]!=senal[i]){
                g.drawLine(desplazamientoHorizontal+x,              desplazamientoVertical+y+EJEX,
desplazamientoHorizontal+x, desplazamientoVertical-y+EJEX);
            }
        }
```

Dibujamos la línea vertical que une el final de la señal con la señal con la señal por defecto. por defecto.

```
        if(!senal[i-1]){
            x=i*LONGITUD_BIT_DEFECTO*(int)(((double)LONGITUD_BIT_DEFECTO*factorx*escaladoX));
            g.drawLine(desplazamientoHorizontal+x,                  desplazamientoVertical+y+EJEX,
desplazamientoHorizontal+x, desplazamientoVertical-y+EJEX);
        }

g.drawLine(desplazamientoHorizontal+x,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DE
FECTO/tensionEntreRayitas),ANCHO,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO
/tensionEntreRayitas));
    }
```

Dibujamos la señal por defecto después de la señal.

- **Osciloscopio.**

  Aquí es donde se lleva a cabo la impresión gráfica y la programación de la pantalla del osciloscopio, la cual se genera de manera dinámica en PantallaOsciloscopio y luego la incrustamos dentro del ***JPanel*** Osciloscopio, y así se dará el aspecto final de nuestro ***Osciloscopio*** lo más parecida a un osciloscopio real (‘*que más adelante se explicará con claridad la programación del mismo*’).

5.5 POD RS-232.

La implementación del POD RS-232 se ha realizado de la siguiente manera: lo primero es que se ha programado la PodRS232Panel.java (elemento ***JPanel***), en donde se realiza toda la programación de cómo pintar el los pines de conexionado del ***POD-RS232*** (*o sea, el aspecto final del conexionado del Pod Rs-232*), y luego en el escritorio virtual “RS232.java” (elemento ***JApplet***) lo genero dinámicamente y dejo caer este elemento ‘PodRS232Panel’ en el elemento PODRS232.java (elemento ***JInternalFrame***, y aquí es donde programo todo el conexionado del mismo.

A continuación, pasaré a explicar el como se ha realizado la programación de PodRS232Panel.java:

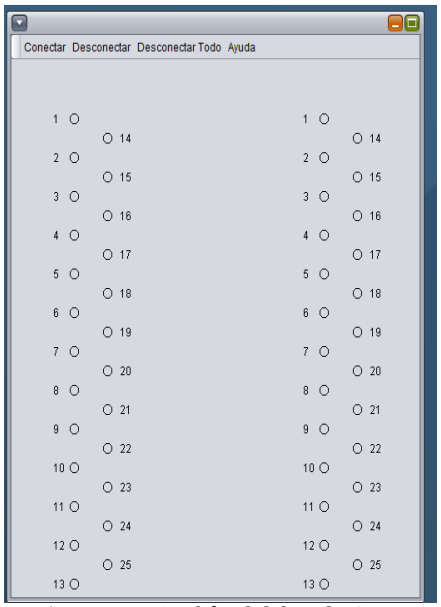


Figura 15. *Vistal final del Pod RS-232.*

Lo primero que hemos realizado para poder dibujar los circulos del Pod RS232 es definirnos ciertas variables para poder trabajar con mas comodidad a la hora de programar dicho **Pod**:

- la primera variable es la 'DISTANCIA_ENTRE_PINES', con esta variable se define la distancia entre pines, o sea, la distancia entre el circulo de conexión de un pin a otro.

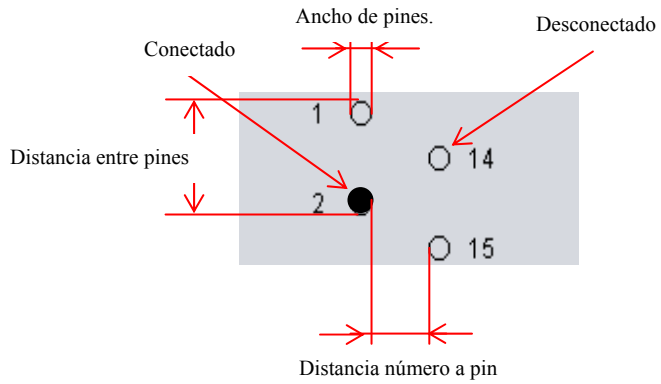
- La segunda variable definada es 'DISTANCIA_NUMERO_A_PIN', que no es mas que la distancia entre el pin de la izquierda y el circulo de la conexión.

- La tercera variable sería en 'ANCHO_PIN', la cual nos define el ancho que va a tener nuestro pin.

- La cuarta y quinta variable son 'CONECTANDO' y 'DESCONECTADO', estas variables nos indican que pines están conectados y desconectados en el Pod RS-232.

- Los arrays 'xConectores[]' e 'yConectores[]', los cuales se encargan de guardar el contenido de las coordenadas de los pines: xConectores, guardará la coordenada 'x' del conector, mientras que yConectores, guardará la coordenada 'y' del conector, entendiendo por conector el pin.

- Y para finalizar 'maxConexion', esta contiene la máxima posición de los array 'conexiones0' y 'conexiones1' que se han escrito, por ejemplo: si hay 3 pines que ya se han conectado, las posiciones 0, 1, 2 de conexiones0 y conexiones1 estarán conectados.



Al igual que en el apartado anterior, daba una breve explicación de alguno de los objetos de java que he utilizado en la implementación del **Applet.** Una de las clases que he utilizado es la clase ***Graphics***, esta clase nos permite dibujar elipses, cuadrados, líneas, mostrar texto y también tiene muchos objetos de dibujo. La clase ***Graphics*** proporciona el entorno de trabajo para cualquier operación gráfica que se realice dentro del **AWT**, para poder pintar un programa necesita un contexto gráfico válido, representada por una instancia de la clase ***Graphics***, pero esta clse no se puede

instanciar directamente, así que debemos crear un componente y pasarlo al programa como un argumento al método ***paint()***.

El único argumento del método ***paint()*** es un objeto de esta clase. Las clases ***Graphics*** disponen de métodos para soportar tres categorías de operaciones gráficas:

i. Dibujo de primitivas gráficas.

ii. Dibujo de texto.

iii. Presentación de imágenes en formato *.gif y *jpeg.

Además, la clase ***Graphics*** mantiene un contexto gráfico: un área de dibujo actual, un color de dibujo del background y otro del foreground, un font con todas sus propiedades, etc. Los eje están situados en la izquina superior izquierda. Las coordenadas se miden siempre en píxeles.

A continuación paso a explicar como se ha implementado el código del PodRS232Panel.JAVA , lo que se explicará será el método más significativo del método.

Función que recibe el objeto ***Graphics*** que le pasemos del método *paintComponent*, esta función tiene por objetivo dibujar un conector en las coordenadas 'x' e 'y', y guardamos esas coordenadas en las variables **xConectores[n]** e **yConectores[n]** y dibujo los conectores en el órden 'n', y en la posición de los array **xConectores[n]** e **yConectores[n]** guardo las coordenadas de los mismos.
NOTA: entenderemos por conector cada uno de los pines del **POD**.

```
private void dibujarConector(Graphics g,int x,int y,int n){

        xConectores[n]=x;
        yConectores[n]=y;

        int[] con;
        int pul;
        if(n==0){
            con=conexiones0;
            pul=pulsadoEn0;
        }
        else{
            con=conexiones1;
            pul=pulsadoEn1;
        }

        int i,j,k;

        for(i=0;i<13;i++){

            k=0;

            while(k<25 && con[k]!=i)k++;

            if(k<25 || i==pul){
                g.fillArc(x+DISTANCIA_NUMERO_A_PIN, y+i*DISTANCIA_ENTRE_PINES, ANCHO_PIN, ANCHO_PIN,0,360);
            }

            else{
                g.drawArc(x+DISTANCIA_NUMERO_A_PIN, y+i*DISTANCIA_ENTRE_PINES, ANCHO_PIN, ANCHO_PIN,0,360);
            }
             //dibuja el número que tiene al aldo del circulo
            g.drawString(String.valueOf(i+1), x, y+i*DISTANCIA_ENTRE_PINES+10);

            }
```

Si n=0 guardaré en la posición 0 de la coordenada 'x' del conector 0, y si n=1 guardaré en la posición 1 de la coordenada 'x del conector 1, y el mismo método se seguirá para la coordenada 'y'.

En **con=conexiones0** tenemos los pines del conector 0 que estan conectados.
En **pul=pulsadoEn0** tenemos el pin que se ha pulsado en el conector 0 pero que todavía no se ha conectado con **conexiones0**

En **con=conexiones1** tenemos los pines del conector 1 que estan conectados.
En **pul=pulsadoEn1** tenemos el pin que se ha pulsado en el conector 1 pero que todavía no se ha conectado con **conexiones1**

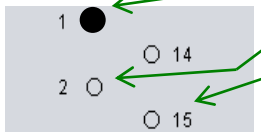
Dibuja los circulos del **Pod**.
Por ejemplo, en el pin 3 del conector de la izquierda está conectado al pin 5 del conector de la derecha, entonces en la posición de **conexiones0** habrá un 3 y en la misma posición de **conexiones1** habrá un 5.

Aquí dibujo los pines del 0 al 13 y compruebo si el **pin** que está dibujado tiene alguna conexión.

Si algún array **con [0 ó 1]** contiene el valor **i** es que ese **pin** está conectado, y este bucle finalizará en el momento en que encuentre el valor **i** en una posición del array '**con[k]**', entonces si el **pin** que estamos dibujando está conectado.

Si el **pin** está conectado y esta pulsado, entonces dibujo el círculo del **pin** con relleno.

Si el **pin** está conectado pero no está pulsado, entonces dibujo el círculo del **pin** sin relleno.



```
        for(j=13;j<25;j++){ //dibuja los pines del 13 al 24
            i=j-14;
            k=0;
            while(k<25 && con[k]!=j)k++;
            if(k<25 || j==pul){
                //si esta conectado y esta pulsado, entonces dibujo el circulo del pin con relleno.
                g.fillArc(x+DISTANCIA_NUMERO_A_PIN+DISTANCIA_ENTRE_PINES, y+i*DISTANCIA_ENTRE_PINES+DISTANCIA_ENTRE_PINES*3/2, ANCHO_PIN, ANCHO_PIN,0,360);
            }
            else{ //si esta conectado pero no esta pulsado, entonces dibujo el circulo del pin sin relleno.
              g.drawArc(x+DISTANCIA_NUMERO_A_PIN+DISTANCIA_ENTRE_PINES, y+i*DISTANCIA_ENTRE_PINES+DISTANCIA_ENTRE_PINES*3/2, ANCHO_PIN, ANCHO_PIN,0,360);
            }
            //dibuja el número que tiene al aldo del circulo
            g.drawString(String.valueOf(j+1), x+2*DISTANCIA_NUMERO_A_PIN+DISTANCIA_ENTRE_PINES, y+i*DISTANCIA_ENTRE_PINES+DISTANCIA_ENTRE_PINES*3/2+10);

        }
}
```

Los siguiente a explicar son los menú que nos encontramos en el *PodRS232.java*, que lo he puesto ahí para el mejor manejo del mismo.

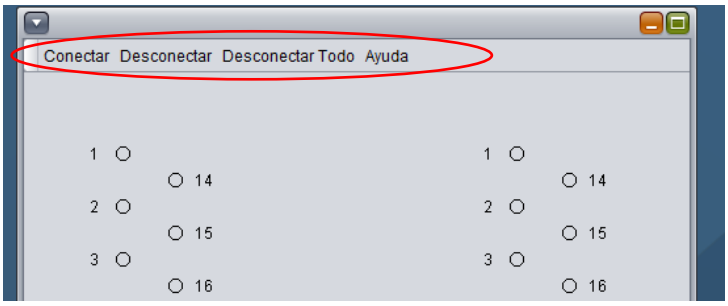


Figura 16. *Menú del Pod RS-232.*

- En **Conectar**, este menú empieza con la conexión de los pines del ***Pod RS232***.
- **Desconectar**, hace la desconexión pin a pin del ***Pod RS232***.
- **Desconectar Todo**, hace la desconexión total del ***Pod RS232*** sin necesidad de tener que desconectar pin a pin.
- **Ayuda**, no abre una ventana con las instrucciones para la conexión y desconexión del ***Pod RS232***.

A continuación se procederá a explicar detalladamente las clases que componen el programa, los diagramas ***UML*** de cada clase van a ser de gran utilidad, con los que se podrán ver las variables utilizadas, los métodos, y las clases que interactúan con dicha clase, ya sea por herencia o por composición. Se empezará explicando la clase *RS232*, que es la clase general del programa.

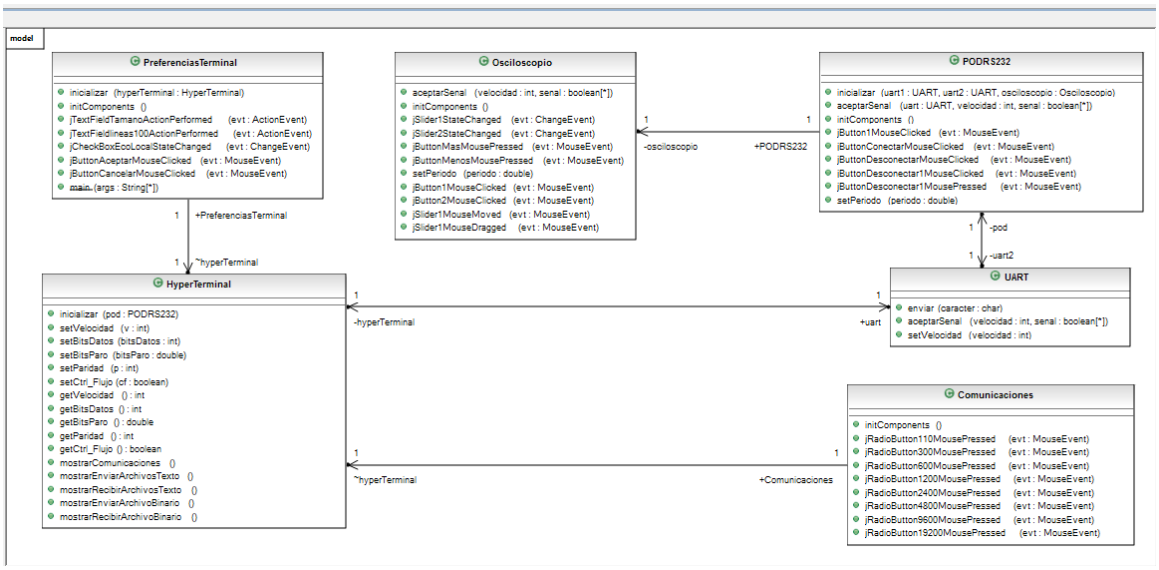


Figura 17. *Diagrama **UML** del proyecto completo.*

5.6 *Diagrama UML de la clase RS232 (RS232.java).*

Ésta es la clase principal desde donde se invocan a las otras clases mediante composición, es decir, la clase ***RS232*** está compuesta por los métodos ***init()*** e ***initComponents()***. La clase ***RS232***, es un *JApplet* (hereda de *JApplet*).

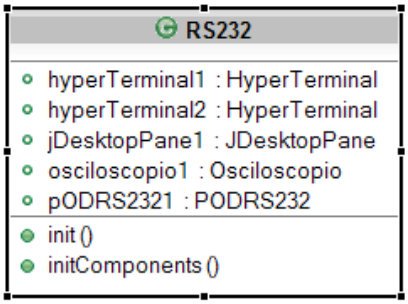


Figura 18. *Diagrama UML de la clase **RS232**.*

Los métodos de la clase ***RS232*** son:

- **init()**

  Es el método constructor, que además de configurar el tamaño de la ventana, también se llama cuando se carga el Applet, y además llama al método ***initComponent()***.

- **initComponent()**

  Éste método se llama desde el método init() para inicializar el formulario, además crea todos los controles del Hyperterminal, pODRS2321, JDesktopPanel y osciloscopio.

5.7 Diagrama UML de la clase HyperTerminal (*HyperTerminal.java*).

Es una clase que se extiende de *JInternalFrame*, y donde están todos los componentes para configurar las conexiones del ***Pod RS-232***, así como la interfaz gráfica.

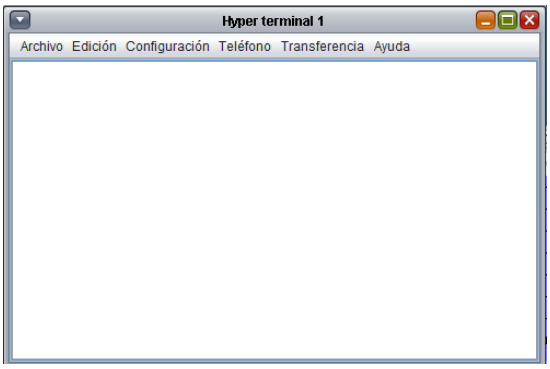


Figura 19. *HyperTerminal.*

Los métodos de la clase ***HyperTerminal*** son:

- **Inicializar(pod: PODRS232)**

  Con esta instrucción inicializamos un nuevo elemento, en concreto un elemento ***UART***, pasandole como parámetro el *pod* del tipo ***PODRS232***.

- **setVelocidad(v: int)**

  Con esta instrucción cambia el valor de la propiedad *velocidad*, este método se llama cada vez que se cambia el valor *velocidad* cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor en *baudios* de la *velocidad de transmisión*.

- **setBitsDatos(bitsDatos: int)**

  Con esta instrucción cambia el valor de la propiedad *bitsDatos*, con esta variable se cambia el valor de *Bits de Datos* cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor de *Bits de datos de transmisión* de datos.

- **setBitsParo(bitsParo: double)**

  Con esta instrucción cambia el valor de la propiedad *BitsParo*, con esta variable se cambia el valor de *Bits de Paro* cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor de *Bits de paro de transmisión* de datos.

- **setParidad(p: int)**

  Con esta instrucción cambia el valor de la propiedad *Paridad*, la *paridad* controla la forma en la que los módems controlan los errores, cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor de *Paridad* de la *transmisión* de datos.

- **setCtrl_Flujo(cf: boolean)**

  Con esta instrucción cambia el valor de la propiedad *Ctrl_Flujo*, el *Control de Flujo* es el método por el que un módem controla la *velocidad* a la que los restantes módems le envían los datos, cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor del *Control de Flujo* de la transmisión de datos.

- **getVelocidad(): int**

  Con esta instrucción cambia el valor de la propiedad *velocidad*, con esta variable se lee el valor de *velocidad* cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor en *baudios* de la *velocidad de transmisión*.

- **getBitsDatos(): int**

  Con esta instrucción cambia el valor de la propiedad *bitsDatos*, con esta variable se lee el valor de *Bits de Datos* cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor de *Bits de datos de transmisión* de datos.

- **getBitsParo(): int**

  Con esta instrucción lee el valor de la propiedad *BitsParo*, con esta variable se lee el valor de *Bits de Paro* cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor de *Bits de paro de transmisión* de datos.

- **getParidad(): int**

  Con esta instrucción se lee el valor de la propiedad *Paridad*, la *paridad* controla la forma en la que los módems controlan los errores, cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor de *Paridad* de la *transmisión* de datos.

- **getCtrl_Flujo(): boolean**

  Con esta instrucción lee el valor de la propiedad *Ctrl_Flujo*, el *Control de Flujo* es el método por el que un módem controla la *velocidad* a la que los restantes módems le envían los datos, cada vez que se llama a la clase ***Comunicaciones.java*** que es donde se selecciona el valor del *Control de Flujo* de la transmisión de datos.

- **mostrarComunicaciones()**

  Con esta instrucción se crea un menú de *Comunicaciones en el HyperTerminal*, que es donde se cambian todas las propiedades para configurar una conexión, además también le damos la propiedad de *setVisible* para poder hacer visible la ventana de *Comunicaciones* de la clase ***Comunicaciones.java***.

- **mostrarEnviarArchivosTexto()**

  Con esta instrucción se crea en el menú del *HyperTerminal* una pestaña de *Enviar Archivo de Texto*, que la utilizamos para poder seleccionar un archivo de nuestro PC para poder enviarlo mediante el protocolo *XModem* de la clase ***Comunicaciones.java***.

- **mostrarRecibirArchivosTexto()**

  Con esta instrucción se crea en el menú del *HyperTerminal* una pestaña de *Recibir Archivo de Texto*, que la utilizamos para poder seleccionar un archivo de nuestro PC para poder enviarlo mediante el protocolo *XModem* de la clase ***Comunicaciones.java***.

- **mostrarEnviarArchivoBinario()**

  Con esta instrucción se crea en el menú del *HyperTerminal* una pestaña de *Enviar Archivo de Binario*, que la utilizamos para poder seleccionar un archivo binario de nuestro PC para poder enviarlo mediante el protocolo *XModem* de la clase ***Comunicaciones.java***.

- **mostrarRecibirArchivoBinario()**

  Con esta instrucción se crea en el menú del *HyperTerminal* una pestaña de *Recibir Archivo de Binario*, que la utilizamos para poder seleccionar un archivo binario de nuestro PC para poder enviarlo mediante el protocolo *XModem* de la clase ***Comunicaciones.java***.

- **aceptarCaracter(carácter: char)**

  En el protocolo *XModem* cuando hay una transferencia de información, esta consta de 3 partes: *establecimiento*, *transmisión de datos* y *liberacion*. Para poder llevarlas a cabo, se hace necesaria la utilización de ciertos caracteres de control, y estos son: ***SOH***, ***ACK***, ***NACK***, ***CAN*** y ***EOT***, con este método aceptamos estos caracteres en la transmisión de un fichero.

- **enviarCaracterDeArchivo()**

  Con este método aceptamos los caracteres especiales dentro de una transmisión de ficheros.

- **initComponents()**

  Con esta instrucción *inicializamos* todos los componentes de la clase ***Comunicaciones.java*** que es donde instanciamos el aspecto final de la ventana ***Comunicaciones***.

- **jMenuItemEnvArchTextoMousePressed(MouseEvent evt)**

  Se active cada vez que hay un cambio de estado en el menú de configuración del *HyperTerminal*, poniendolo a *true* la variable del asistente, de tal forma que al pulsar el submenú *Enviar Archivo de Texto*, nos mostrará una nueva ventana para poder seleccionar un archivo de texto de nuestro PC para poder enviarlo.

- **jMenuItemPreferenciasMousePressed(MouseEvent evt)**

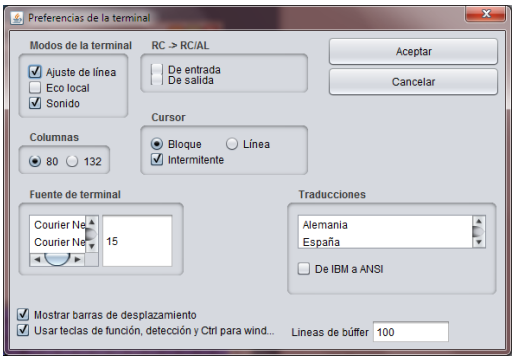
  Se active cada vez que hay un cambio de estado en el menú de *Configuración* del *HyperTerminal*, poniendolo a *true* la variable del asistente, de tal forma que al pulsar el submenú *Preferencias del Terminal*, nos mostrará una nueva ventana para poder seleccionar las *Preferencias del Terminal*, y así poder configurar el *eco local* el tipo de conversión de código *ASCII* de *retorno de carro* en *retorno de carro + avance de línea*.

- **jMenuItemRecArchTextoMousePressed(MouseEvent evt)**

  Se active cada vez que hay un cambio de estado en el menú de configuración del *HyperTerminal*, poniendolo a *true* la variable del asistente, de tal forma que al pulsar el submenú *Recibir Archivo de Texto*, nos mostrará una nueva ventana para poder seleccionar donde guardar el archivo seleccionado en nuestro PC para poder recibirlo.

5.8 Diagrama UML de la clase Prefencias de Terminal (*PreferenciasTerminal.java*).

A continuación se pasará a explicar el diagrama ***UML*** de la clase *PreferenciasTerminal*, es una clase que se extiende de *JDialog*, aquí es donde podemos configurar las variables, tales como: el ***eco Local***, que es el encargado de mostrar en un ***HyperTerminal*** los caracteres pulsados para una transmisión, y otro parámetro importante a configurar es el ***RC→RC/AL*** que es el encargado de en el ***HyperTerminal*** de lo que se ha escrito en una línea el momento de pulsar el ***retorno de carro*** en el ***HyperTerminal*** dependiendo de lo que selecciones nos mostrará una cosa u otra, por ejemplo: si tenemos seleccionado ***RC→RC/AL*** entrada, pues en el ***HyperTerminal*** del emisor el momento de que tecleemos el ***retorno de carro*** en nuestro ***HyperTerminal*** se mostrará en la siguiente línea.



Los métodos de la clase ***PreferenciasTerminal*** son:

- **inicializar(hyperTerminal: HyperTerminal)**

  Este método inicializa un componente ***HyperTerminal***, que es a partir del cual se lanza mediante el menú ***Configuración***, en el submenú ***PreferenciasTerminal*** una ventana para empezar a configurar los parámetros con los cuales vamos a sconfigurar las ***Preferencias del Terminal*** del ***HyperTerminal***.

- **initComponents()**

  Con esta instrucción *inicializamos* todos los componentes de la clase ***PreferenciasTerminal.java*** que es donde instanciamos el aspecto final de la ventana ***PreferenciasTerminal***.

- **jButtonAceptarMouseClicked(java.awt.event.MouseEvent evt)**

  Este método captura el click del ratón del ***botón*** *Aceptar* de la configuración de las ***Preferencias del Terminal***, para guardar la configuración seleccionada por el usuario.

- **jButtonCancelarMouseClicked(java.awt.event.MouseEvent evt)**

  Este método captura el click del ratón del ***botón*** *Cancelar* de la configuración de las ***Preferencias del Terminal***, para cancelar la configuración seleccionada por el usuario.

- **jCheckBoxEcoLocalStateChanged(javax.swing.event.ChangeEvent evt)**

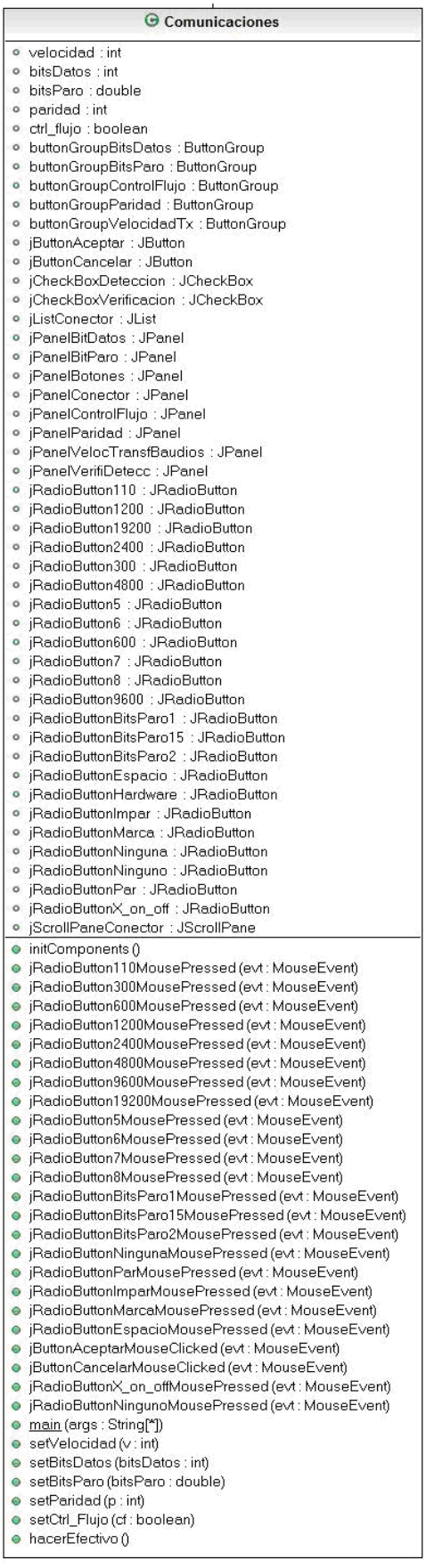
  Este método captura el click del ratón del *checkBox* para capturar el cambio de estado de la configuración de las ***Preferencias del Terminal***, para saber si seleccionamos la preferencia de si queremos ***eco Local*** o no queremos ***eco Local***.

- **main(args: String[*])**

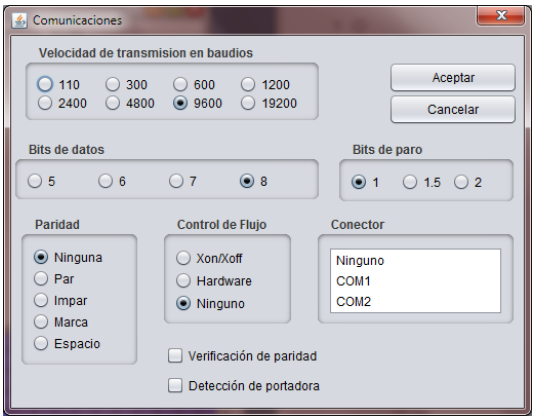
  En este método se creo un nuevo elemento ***dialog*** para crear un ***frame*** de la ventana ***PreferenciasTerminal***, el ***dialog*** añade una clase *addWindowsListener* **que se liga al *frame*** *windowsAdapter*. Otro evento que tiene este método es la clase *windowsClosing***,** que se invoca cuando el usuario intenta cerrar la ventana desde el menú del sistema de la ventana.

5.9 Diagrama UML de la clase Comunicaciones

(*Comunicaciones.java***).**



A continuación se pasará a explicar el diagrama ***UML*** de la clase *Comunicaciones*, es una clase que se extiende de *JDialog*, aquí es donde podemos configurar las variables, tales como: la ***velocidad de transmisión en baudios***, que es la encargada de configurar la velocidad a la cual se va a transmitir la señalde un ***HyperTerminal*** a otro, otro parámetro importante son los ***Bits de Datos*** que ya se explicaron en su momento, ***Bits de Paro***, ***Paridad***, ***Control de Flujo***, etc…



Los métodos de la clase ***Comunicaciones*** son:

- **hacerEfectivo()**

  Con este método lo que hacemos es captura el cambio de valor (*hacer efectivo el cambio*)de las variables ***HyperTerminal***, ***Bits de Paro***, ***Bits de Datos***, ***Paridad*** y ***Control de Flujo***, en la ventana de ***Comunicaciones***.

- **jButtonAceptarMouseClicked(java.awt.event.MouseEvent evt)**

  Este método captura el click del ratón del ***botón*** *Aceptar* de la configuración de la ventana de ***Configuración***, para guardar la configuración seleccionada por el usuario.

- **jButtonCancelarMouseClicked(java.awt.event.MouseEvent evt)**

  Este método captura el click del ratón del ***botón*** *Cancelar* de la configuración de la ventana de ***Comunicaciones***, para cancelar la configuración seleccionada por el usuario.

- **jRadioButton110MousePressed(java.awt.event.MouseEvent evt)**

  Este método se refiere a la selección de la *velocidad de transferencia en baudios***,** el momento en que el ***jRadioButton110MousePressed*** cambia de estado (*inicialmente jRadioButtonXXXMousePressed* = **false**), entonces la el *RadioButton* cambia de valor por el valor seleccionado por el usuario.

  Para los métodos: ***jRadioButton110MousePressed***, ***jRadioButton1200MousePressed, jRadioButton19200MousePressed, jRadioButton2400MousePressed, jRadioButton300MousePressed, jRadioButton4800MousePressed, jRadioButton5MousePressed, jRadioButton600MousePressed, jRadioButton6MousePressed, jRadioButton7MousePressed, jRadioButton8MousePressed, jRadioButton9600MousePressed, jRadioButtonBitsParo1MousePressed, jRadioButtonBitsParo15MousePressed, jRadioButtonBitsParo2MousePressed, jRadioButtonEspacioMousePressed, jRadioButtonImparMousePressed, jRadioButtonNingunaMousePressed, jRadioButtonNingunoMousePressed, jRadioButtonMarcaMousePressed, jRadioButtonParMousePressed, jRadioButtonX_on_offMousePressed,*** se hacen de manera análoga a la anterior (***jRadioButton110MousePressed***), solo que cambian el nombre del método por el correspondiente método.

5.10 Diagrama UML de la clase Osciloscopio (*Osciloscopio.java*).

A continuación se pasará a explicar el diagrama ***UML*** de la clase ***Osciloscopio***, es una clase que se extiende de ***JInternalFrame***, aquí es donde declaramos los métodos mediante los cuales funciona el ***Osciloscopio***, mediante las cuales se puede: *cambiar la amplitud de la señal* en el ***osciloscopio***, mediante unos botones que son los que ampliaran o reduciran la *amplitud* de la señal, una escala de tiempo en donde se puede ver el *tiempo de bit* que depende de la velocidad de transmisión en baudios, etc …

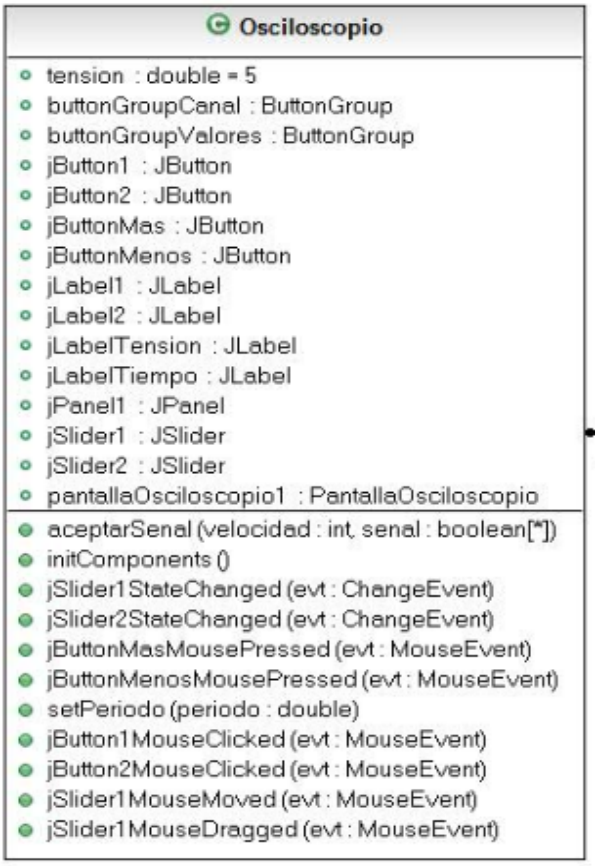


Figura 20. *Diagrama UML del Osciloscopio.*

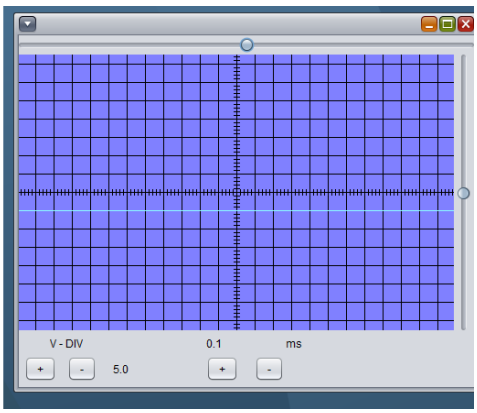


Figura 21. *Vista final del Osciloscopio.*

Los métodos de la clase ***Osciloscopio*** son:

- **aceptarSenal(velocidad: int, senal: boolean[*])**

  Método por el cual se captura la señal transmitida y se pinta en el ***osciloscopio***, esta señal estar configurada con la velocidad de transmisión que se ha configurado en el apartado ***Comunicaciones*** del menú ***configuración***.

- **jSlider1StateChanged(evt: ChangeEvent)**

  Con método realizado en ***jSlider1*** lo que hace es mover la señal transmitida en el ***osciloscopio*** de *izquierda* a *derecha* y viceversa, esto lo necesitaremos para centrar la señal y así poder medir bien la *tensión* de la señal con mas facilidad.

- **jSlider2StateChanged(evt: ChangeEvent)**

  Con método realizado en ***jSlider2*** lo que hace es mover la señal transmitida en el ***osciloscopio*** de *arriba* hacia *abajo* y viceversa, esto lo necesitaremos para centrar la señal y así poder medir bien la *amplitud* de la señal con mas facilidad.

- **jButtonMasMousePressed(evt: MouseEvent)**

  Con este método aplicado al botón lo que realizaremos será, aumentar el valor de *amplitud* de la señal transmitida, y además recalcularemos el valor de los *voltios por división* dependiendo del valor inicial.

- **jButtonMenosMousePressed(evt: MouseEvent)**

  Con este método aplicado al botón lo que realizaremos será, disminuiremos el valor de *amplitud* de la señal transmitida, y además recalcularemos el valor de los *voltios por división* dependiendo del valor inicial.

5.11 *Diagrama UML de la clase PantallaOsciloscopio* (*PantallaOsciloscopio.java*).

A continuación se pasará a explicar el diagrama ***UML*** de la clase *PantallaOsciloscopio*, es una clase que se extiende de *JPanel*, aquí es donde declaramos los métodos mediante los cuales funciona el ***Osciloscopio***, mediante las cuales se puede: *cambiar la amplitud de la señal* en el ***osciloscopio***, mediante unos botones que son los que ampliaran o reduciran la *amplitud* de la señal, una escala de tiempo en donde se puede ver el *tiempo de bit* que depende de la velocidad de transmisión en baudios, etc …

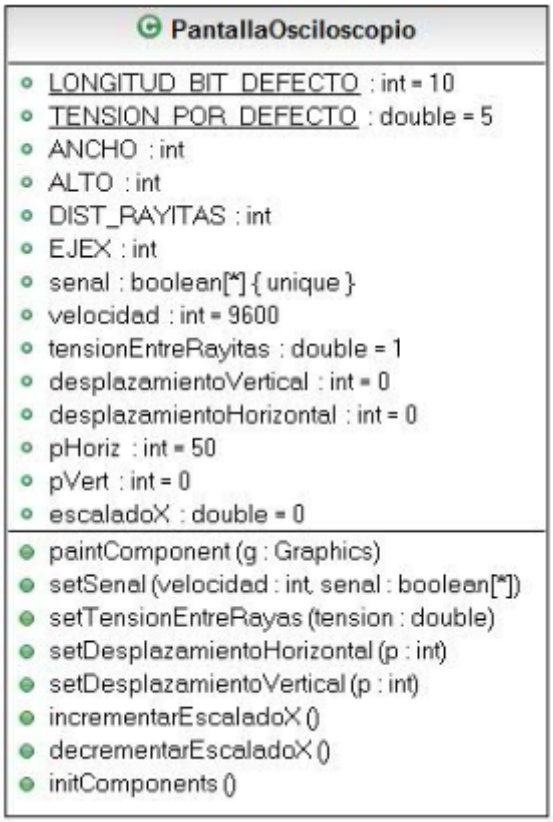


Figura 22. *Diagrama **UML** de la clase Pantalla de Oscilospcopio*

5.11 Diagrama UML de la clase PODRS232 (PODRS232.java).

A continuación se pasará a explicar el diagrama ***UML*** de la clase ***PODRS232***, es una clase que se extiende de ***JInternalFrame***, aquí es donde se inicializan los valores de la ***UART*** y el ***osciloscopio***, aceptaremos la señal que nos llega de la ***UART*** así como las ***configuración*** que se le ha dado a la misma, y además aquí es donde se ha generado dinámicamente la clase ***PodRS232Panel*** y así pintar todos los pines del ***Pod RS-232***.

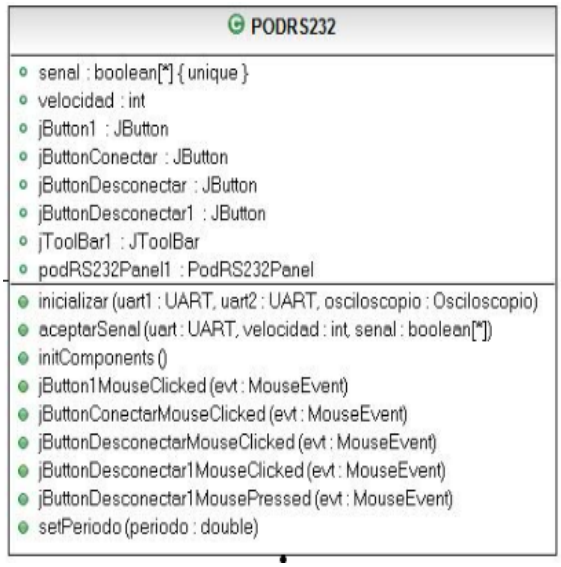


Figura 22. *Diagrama* **UML** de la clase Pod RS232.

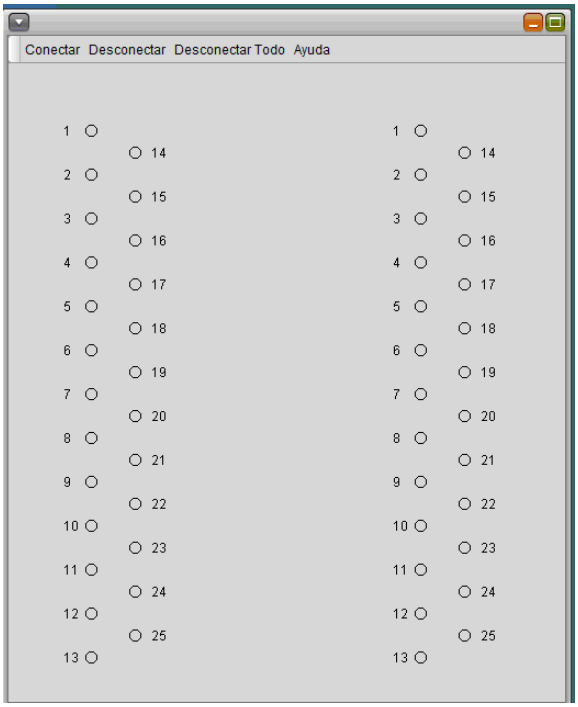


Figura 23. *Vista completa del Pod RS-232.*

5.12 Diagrama UML de la clase PodRS232Panel (*PodRS232Panel.java*).

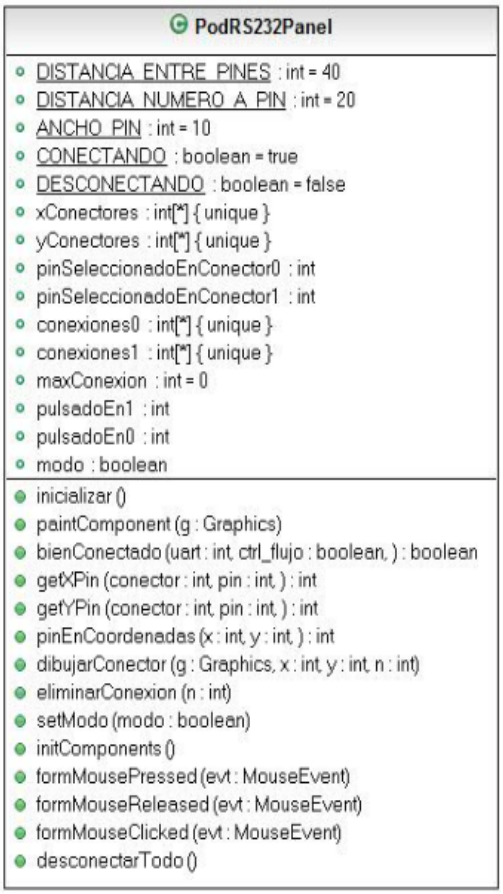


Figura 24. *Diagrama **UML** de la clase PodRS232.*

A continuación se pasará a explicar el diagrama ***UML*** de la clase ***PodRS232Panel***, es una clase que se extiende de ***JPanel***, aquí es donde se definen las variables para el control del método ***PODRS232***, estas variables son: la *distancia_entre_pines* que aquí es donde se define la distancia entre un pin y otro, la *distancia_numero_a_pin* que definimos la distancia entre el pin de la izquierda y un pin de conexión, el *ancho_pin*, *conectando, desconectando, xConectores, yConectores, maxConexiones* y *modo*.

- **inicializar()**

  Este método inicializa las variables *conexiones0[i]* y *conexiones1[i]* a un valor de -1 para que cuando aparezca por primera vez la imagen del **Pod** este se muestre con los pines desconectados '*no pulsados*'.

- **paintComponent(g: Graphics)**

  Con este método lo que hacemos es los conectores de los pines en las coordenadas pasadas por línea de código, y así dibujar los pines que se muestran en la imagen final del **Pod**.

- **bienConectado(uart: int, ctrl._flujo: boolean,): boolean**

  Con este método lo que hacemos es comprobar el correcto conexionado de manera correcta a la hora de configurar el modo de envio de la señal transmitida.

- **getXPin(conector: int, pin: int,): int**

  Con este método lo que hacemos es calcular las coordenadas '**x**' de los pines del conector, para poder dibujarlas en esas coordenadas el pin.

- **getYPin(conector: int, pin: int,): int**

  Con este método lo que hacemos es calcular las coordenadas '**y**' de los pines del conector, para poder dibujarlas en esas coordenadas el pin.

- **pinEnCoordenadas(x: int, y: int,): int**

  En este método lo que hacemos es mirar las coordenadas donde ha hecho click el usuario y mira si se corresponde a un pin y de que conector corresponde.

- **dibujarConector(g: Graphics, x: int, y: int, n: int)**

  Con este método los que haremos es crear una función que recibe el objeto *Graphics* que le pasamos del método *paintComponent*, esta función tiene por objetivo dibujar un conector en las coordenadas **x** e **y** calculadas anteriormente en los métodos *getXPin* e *getYPin* y las guarda en los array *xConectores[n]* e *yCoordenadas[n]*, y dibuja los conectores de orden '**n**', y en las posiciones de los array *xConectores[n]* e *yConectores[n]* guardo las coordenadas de los mismos, entendiendo por conector cada uno de los pines.

- **eliminarConexion(n: int)**

  Con este método lo que hacemos es, eliminar las conexiones del **Pod** una a una.

- **formMouseClicked(evt: MouseEvent)**

  Con este método compruebo si se ha hecho click en un pin, o sea si esta ese pin *conectado* o *desconectado*.

- **desconectarTodo()**

  Con este método hago la desconexión total de las conexiones del **Pod** y así si se ha realizado alguna conexión que no se haya querido hacer, pues se desconecta todo y así poder empezar desde cero el conexionado del mismo.

5.13 Diagrama UML de la clase UART (UART.java).

A continuación se pasará a explicar el diagrama ***UML*** de la clase ***UART***, en la cual se realiza toda la programación de las configuraciones guardadas en el método ***HyperTerminal***, y así poder enviarle la señal al ***Pod RS-232*** y esa señal así poder mostrarla en el ***osciloscopio***.

Figura 25. *Diagrama* **UML** *de la clase UART.*

Los métodos de la clase ***UART*** son:

- **enviar(carácter: char)**

  Este método lo que hace es enviar la señal que le ha llegado configurada del ***HyperTerminal*** hacia el ***POD RS-232***.

- **aceptarSeñal(velocidad: int, señal: boolean[*])**

  Este método lo que hace es aceptar la señal que le llega configurada del ***HyperTerminal*** y esta señal l atrata con los parámetros para enviarla al ***POD RS-232***.

Capítulo 6.

6 Uso académico de la aplicación.

En este capítulo vamos a aclarar los conceptos teóricos necesarios para el perfecto entendimiento de las *Comunicaciones Serie Asíncronas*.

6.1 Historia

En los años 60, cada fabricante usaba una interfaz diferente para comunicar un ***DTE*** (*Data Terminal Equipment*) y un ***DCE*** (*Data Comunication Equipment*). Cables, conectores y niveles de voltajr eran diferentes e incompatibles, por lo tanto, la interconexión entre equipos de diferentes fabricantes requería el uso de convertidores de niveles de voltaje y la fabricación de cables y conectores especiales.

En 1969, el ***EIA*** junto con Bell Laboratories y otros fabricantes establecieron un estándar para la interfaz entre ***DTE's*** y ***DCE's***. El objetivo de este estándar era simplificar la interconexión de equipos fabricados por diferentes firmas.

El estándar llegó a ser el ***RS-232*** (*Recommended Estándar number 232, revisión C from the Electronic Industry Association*). Un estándar similar fue desarrollado en Europa por el **CCITT** (*Comité Consultatif Internatile de Telegraphie et Telephonie*) conocido como **V.24** (*descripción funcional*) y **V.28** (*especificaciones eléctricas*). El ***RS-232-C*** fue adoptado por la mayor parte de fabricantes de terminales y equipamiento.

En 1980 la creciente industria de las **PC** encontró el estándar ***RS-232-C*** barato y apropiado para conectar periféricos a la **PC**. El ***RS-232-C*** llego a ser rápidamente un estándar para conectar a la **PC**: impresoras, cintas de backup, terminales y otras **PC**'s.

Como el estándar solamente soporta velocidades de transmisión hasta 20 Kbps y distancias de hasta 16 metros, se adoptaron nuevos estándares por la ***EIA***. El ***RS449*** (*decripción mecánica*) y ***RS423*** (*descripción eléctrica*) son compatibles con el ***RS-232-C*** y se puede operar a velocidades de hasta 10 Mbps y alcanzar distancias de hasta 1200 metros. Sin embargo, la adaptación de un nuevo estándar es un proceso largo y costoso. El estándar ***RS-232-C*** está muy extendido y por lo tanto le queda bastante vida.

6.2 Introducción.

Muchos microcontroladores poseen una interfaz ***UART*** o ***USART*** para comunicarse serial asíncrona, tipo ***RS-232***, que en un *PC* se denominan puertos '*COM*'. Si bien los microcontroladores poseen hardware para generar la secuencia de bits en los tiempos correctos, no son capaces de generar el voltaje especificado por el estándar ***RS-232***, por lo cual requieren de un chip externo que haga esta conversión de voltajes.

6.3 Aspectos importantes de la comunicación.

La comunicación entre computador y un dispositvo externo como tal es un aspecto fundamental que es necesario conocer con profundidad para poder solventar alguno de los problemas que puedan suceder durante ella. En la mayor parte de los casos, no se tiene en cuenta circunstancias exteriores (*ruido, vibración, etc.*) que puede dar lugar a una incorrecta comunicación y por lo tanto el proceso derivado de tal comunicación no se puede dar por bueno. Es por ello que es importante conocer las diversas posibilidades y limitaciones que la comunicación entre dispositivos como son los computadores y dispositivos externos.

En nuestro caso, se llevará a cabo una comunicación serie mediante un puerto de comunicaciones serie. Tal puerto es utilizado para la comunicación del computador con diversos sistemas como por ejemplo: impresoras, ratones, teclados, etc. Pero a nosotros nos interesa la utilzación de este puerto aplicado a este proceso de medición de forma exclusiva. Nos centraremos, por ello, en hablar en aspectos acerca de la comunicación serie y de sus principales características, así como poder explicar el funcionamiento de este dispositivo.

6.4 Descripción del estándar RS-232.

El estándar ***RS-232-C*** describe una interfaz entre un ***DTE*** y un ***DCE*** que emplea un intercambio en serie de datos binarios. En el se definen características eléctricas, mecánicas, funcionales de la interfaz y modos de conexión comunes. Las características eléctricas incluyen parámetros tales como niveles de montaje e impedancias del cable. La sección mecánica describe los pines. La descripción funcional define las funciones de las secciones eléctricas que se usan.

El ***RS-232*** permite la transmisión *síncrona* y *asíncrona*. La subnorma *asíncrona* es sin duda la más frecuente:

- En la transmisión ***síncrona*** el envio de un grupo de caracteres en un flujo contínuo de bits. Para lograr la sincronización de ambos dispositivos (*receptor y transmisor*) ambos dispositivos proveen una señal de reloj que se usa para establecer la velocidad de transmisión de datos para habilitar los dispositivos conectados a los módem para identificar los caracteres apropiados mientras estos son transmitidos o recibidos. Antes de iniciar la comunicación ambos dispositivos deben de establecer una sincronizaciòn entre ellos. Para esto, antes de enviar los datos se envían un grupo de caracteres especiales de sincronía. Una vez que se logra la sincronía, se puede empezar a transmitir datos.

  Por lo general, os dispositivos que transmiten en forma síncrona son mas caros que los asíncronos. Debido a que son más sofisticados en el hardware. A nivel mundial son más empleados los dispositivos asíncronos ya qie facilitan mejor la comunicación.

  La transmisión ***síncrona*** es una técnica que consiste en el envio de una trama de datos (*conjunto de caracteres*) que configura un bloque de información comenzando con un conjunto de ***bits*** de *sincronismo* (***SYN***) y terminando con un conjunto de bits de final de bloque (***ETB***). En este caso, los bits de sincronismo tienen la función de sincronizar los relojes

existentes tanto en el emisor como en el receptor, de tal forma que estos controlan la duración de cada bit y carácter.

Dicha transmisión se realiza con un ritmo que se genera centralizadamente en la red y es el mismo para el emisor como para el receptor. La información se transmite entre dos grupos, denminados delimitadores (*8 bits*).

A diferencia de la transmisión ***asíncrona***, en este tipo de transimisión no se utilizan bits de **inicio** o **parada**, aquí para evitar la desincronización lo que se usa son relojes que permiten que los bits se envieen a una velocidad constante que es dictada por los pulsos de reloj.
Cabe destacar que en este tipo de transmisión antes de enviar cualquier dato se debe primero enviar un grupo de caracteres de sincronía para que el receptor sepa que va a recibir un mensaje.

En transmisión es utilizada cuando se necesita bastante velocidad, y el hardware que se utiliza suele ser más costoso que el de la transmisión asíncrona.

Ejemplo:

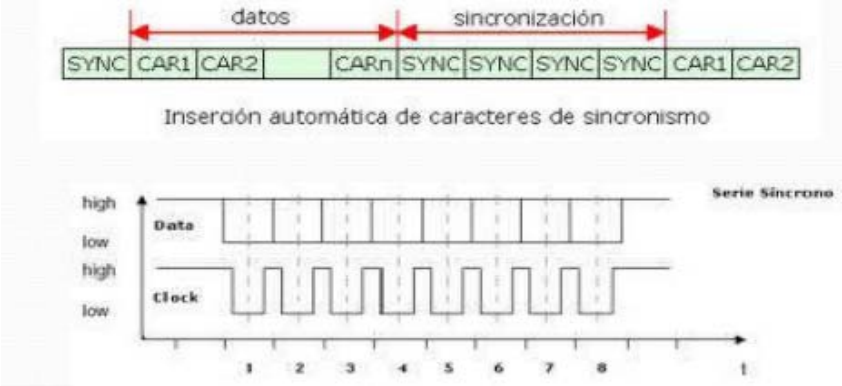


Figura 26. *Ejemplo de una comunicación síncrona.*

→ Características

Los bloques a ser transmitidos tienen un tamaño que oscila entre 128 y 1024 bytes. La señal de sincronismo en el extremo fuente, puede ser generada por el equipo terminal de datos o por el módem. Cuando se transmiten bloques de 1024 bytes y se usan no más de 10 bytes de cabecera y terminación, el rendimiento de transmisión supera el 99%.

→ Ventajas

- Posee un alto rendimiento en la transmisión.
- Los equipamientos son de tecnología más compleja y de costes más altos.
- Son aptos para transmisiones de altas velocidades (iguales o mayores a 1200 baudios de velocidad de modulación).
- El flujo de datos es más regular.

También llamada transmisión Síncrona. A todo conjunto de bits y datos se le denomina ***TRAMA***.

- La transmisión ***asíncrona*** es aquella que se transmite o recibe un carácter, bit por bit añadiendole bits de *inicio*, y bits que indican el término de un paquete de datos, para separar así los paquetes que se van *enviando*/*recibiendo* para sincronizar el receptor con el transmisor. El bit de inicio le indica al dispositivo receptor que sigue un carácter de datos; similarmente el bit de término indica que el carácter o paquete a sido completado.

  La transmisión ***asíncrona*** se da lugar cuando el proceso se sincronización entre emisor y receptor se realiza en cada palabra de código transmitido. Esta sincronización se lleva a cabo a través de unos bits especiales que definen el entorno de cada código.

  También se dice que se establece una relación ***asíncrona*** cuando no hay ninguna relación temporal entre la estación que transmite y la que recibe. Es decir, el ritmo de presentación de la información al destino no tiene por qué coincidir con el ritmo de presentación de la información por la fuente. En estas situaciones tampoco se necesita garantizar un ancho de banda determinado, suministrando solamente el que esté en ese momento disponible. Es un tipo de relación típica para la transmisión de datos.

  En este tipo de red el receptor nos abe con presición cuando recibirá un mensaje. Cada carácter a ser transmitido es delimitado por un bit de información denominado de cabecera o de arranque, y uno o dos bits denominados de ***terminación*** o ***parada***.

  i. El bit de arranque tiene dos funciones de sincronización de reloj del transmisor y del receptor.

  ii. El bit o bits de parada, se usan para separar un carácter del siguiente.

  Después de la transmisión de los bits de información se suelen agregar un bit de ***paridad*** (par o impar). Dicho bit sirve para comprobar que los datos se transfieran sin interrupción. El receptor revisa la paridad de cada unidad de entrada de datos.

  Partiendo desde la línea de transmisión en reposo, cuando tiene un nivel lógico **1**, el emisor informa al receptor de que va a llegar un carácter, para ello antepone un bit de ***arraque*** (*start*) con el valor lógico **0**. Una vez que el bit *start* llega al receptor este disparará un reloj interno y se quedará esperando por los sucesivos bits que contendrá la información del carácter transmitido por el emisor.

  Una vez que el receptor recibe todos los bits de información se añadirá al menos un bit de ***parada*** (*stop*) de nivel lógico **1**, que repondrán es su estado inicial a la línea de datos, dejándola así preparada para la siguiente transmisión del siguiente carácter. Es usada en velocidades de modulación de hasta 1200 baudios. El rendimiento se basa en el uso de un bit de ***arranque*** y dos de ***parada***, en una señal que use código de 7 bits más uno de ***paridad*** (8 bits sobre 11 transmitidos) es del 72 por 100.

En resumen:

- En esta transmisión el emisor decide cuando va a enviar el mensaje por la red, mientras que el receptor no sabe n que momento le puede llegar dicho mensaje, para esto se utiliza un bit de cabecera que va al inicio de cada carácter y uno o dos bits de parada que va al final de ese mismo carácter, esto se hace con la finalidad que tanto el emisor como el receptor puedan sincronizar sus relojes y poder decodificar el mensaje.

- Este tipo de transmisión es utilizada cuando no se necesita mucha velocidad, ya que cada carácter es transmitido de uno en uno y por lo tanto puede ser un poco lento, por otra parte, los equipos que se utilizan son económicos.

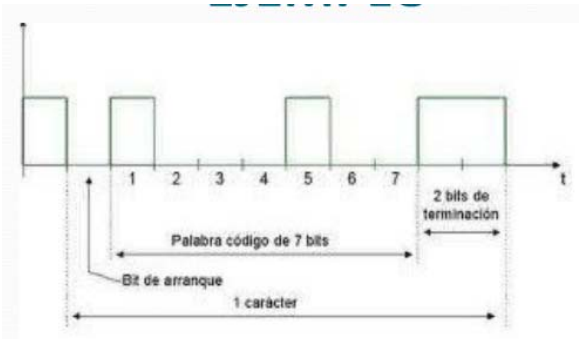


Figura 27. *Ejemplo de una comunicación asíncrona.*

Ventajas y **desventajas**:

- En caso de errores se pierde siempre una cantidad pequeña de caracteres, pues éstos se sincronizan y se transmiten de uno en uno.

- Bajo rendimiento de transmisión, dada la proporción de bits útiles y de bits de sincronismo, que hay que transmitir por cada carácter.

- Es un procedimiento que permite el uso de equipamiento más económico y de tecnológia menos sofisticada.

- Se adecua más fácilmente en aplicaciones, donde el flujo transmitido es más irregular.

- Son especialmente aptos, cuando no se necesitan lograr altas velocidades.

La transmisión asíncrona se lleva a cabo tal y como se describe en lo anterior explicado. En concreto además del bit de ***start*** utiliza:

- 5, 6, 7 y 8 son bits de ***datos***.

- 0 y 1 de ***paridad*** (la ***paridad*** puede ser “par” (*Even*), “impar” (*Odd*), “siempre a cero” (*Reset*) y “siempre a uno” (*Set*)).

- 1, 1.5 o 2 bits de ***stop***.

Para agilizar el lenguaje se suele emplear una nomenclatura abreviada como, por ejemplo, "8N1" que indica que la transmisión serie ***RS-232*** se ha configurado para transmitir 8 bits de datos, No ***paridad*** y un bit de ***stop***. Otro ejemplo sería "6E2" que indica 6 bits de ***datos***, ***paridad*** *par* y dos bits de ***stop***.

El nombre oficial del estándar es **EIA/TIA-232-E** y es un estándar completo, puesto que no sólo especifica los niveles de voltaje y señal, sino que además especifica la configuración de pines de los conectores y una cantidad mínima de información de control entre equipos. También especifica la forma y características físicas de los conectores.

Este estándar fue definido en 1962, antes de la lógica **TTL**, razón por la cuál no utiliza los niveles lógicos de 5 volts y tierra. Un nivel alto a la salida del transmisor está definido como un voltaje entre +5 y +15 volts, mientras que un nivel bajo está definido como un voltaje entre -5 y -15 volts.

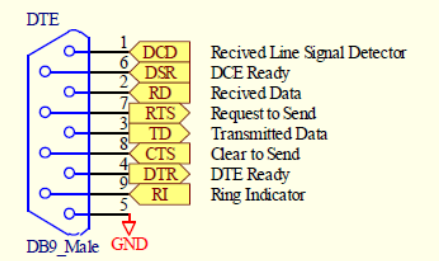
La lógica del receptor fue diseñada para permitir un nivel de ruido de 2 voltios. Así, un nivel alto para el receptor está definido en el rango +3 a +15 voltios, mientras que un nivel bajo va desde los -3 a los -15 voltios.

Es importante notar que un nivel alto está representado por un valor lógico '**0**', históricamente llamado ***spacing*** (*espacio*), mientras que un nivel bajo representa un valor lógico '**1**', históricamente referenciado como ***marking*** (*marca*).

Este estándar también define un máximo *slew rate* o **máxima variación de voltaje** de 30[V/µs] para evitar el *crosstalk*, que es la inducción de las señales que viajan por un cable en los cables adyacentes. Inicialmente, el estándar limitaba la velocidad de transferencia de datos a 20[kbps] (kilo bits por segundo). Actualmente los circuitos integrados soportan velocidades mucho mayores, de hasta 350[kbps], manteniendo el *slew rate*.

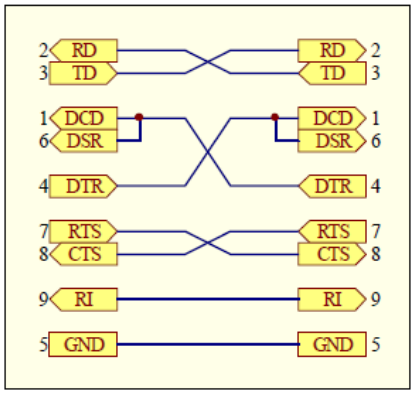
La carga vista por el transmisor se especificó en 3 a 7 [kΩ]. En un principio se estableció un largo máximo del cable de 15 metros, pero luego fue modificado por la revisión D del estándar. Ésta especifica una maxima capacitancia de 2500[pF], en vez de establecer un largo máximo. Así, el largo máximo depende de la capacitancia característica del cable utilizado.

El estándar estableció 4 grupos de señales: ***común***, ***datos***, ***control*** y ***temporizamiento***, sumando en total 24 señales. También especifica un conector de 25 pines llamado **DB25**, el cuál es capaz de incluir todas estas señales. Afortunadamente sólo muy pocos equipos utiliza esta gran cantidad de señales. La mayoría, además de la señal de tierra de referencia, requiere sólo 2 para datos y 2 para control, ó sólo el par de datos. Estos últimos suelen utilizar un conector **DB9S**, de 9 pines, el cuál permite acomodar las mínimas señales utilizadas por equipos modernos. La figura a continuación presenta las señales en un conector **DB9**. Este conector está visto desde fuera del computador. Las señales que apuntan hacia la derecha son señales que salen del computador, mientras que las que apuntan a la izquierda son entradas al computador.



Las mínimas señales utilizadas en una comunicación bidireccional son **TD** para *transmitir datos* y **RD** para *recibir datos*. Asimismo, si desea utilizarse *control de flujo por hardware*, se utilizan las señales ***RTS*** (*petición de transmisión*) y ***CTS*** (*habilitado para transmitir*). El *control de flujo* impide que un transmisor rápido sature a un receptor lento. Normalmente el PC podrá transmitir datos ininterrumpidamente, pero el equipo receptor puede ser más lento y no alcanzar a procesar todos los datos que le envía el PC.

La interfaz ***RS-232*** está pensada para conectar un terminal de datos (**DTE**) a un equipo tipo modem, llamado equipo de datos de terminación en circuito (**DCE**). El **DCE** es un equipo que hace la interfaz entre el **DTE** y el medio por el cuál se transmitirán los datos. Un ejemplo de **DCE** es un modem, el cuál hace de interfaz entre un **PC** y la línea telefónica. También pueden conectarse 2 **DTE** directamente a través de un puerto **RS-232**. Para ello se emplea un cable denominado ***null-modem***. Este cable es especial, ya que posee líneas de datos y control invertidas entre sus 2 conectores. A continuación se muestra la conexión interna de un cable ***null-modem***:



Las señales **RTS** y **CTS** también pueden utilizarse para establecer la dirección de comunicación en un sistema *half-duplex*. Esto es necesario cuando se utilizan conversores **RS-232** a **RS-485**, pues este útlimo utiliza un mismo par trenzado tanto para transmitir como para recibir, convirtiéndolo en un protocolo *half-duplex*.

6.5 Modos de transmisión.

Los distintos tipos de transmisión de un canal de comunicaciones pueden ser de tres clases:

i. Símplex.

ii. Semidúplex.

iii. Dúplix.

→ Método Símplex

Es aquel en el que una estación siempre actúa como fuente y la otra siempre como colector, este método permite la transmisión de información en un único sentido.

→ Método Semidúplex

Es aquel en el que una estación **A** en un momento de tiempo, actúa como fuente y otra estación corresponsal **B** actúa como colector, y en el momento siguiente, la estación **B** actuará como fuente y la **A** como colector. Permite la transmisión en ambas direcciones, aunque en momentos diferentes. Un ejemplo es la conversación entre dos radioaficionados, pero donde uno espera que el otro termine de hablar para continuar el diálogo.

→ Método Dúplex

En el que dos estaciones **A** y **B**, actúan como fuente y colector, transmitiendo y recibiendo información simultáneamente, permite la transmisión en ambas direcciones y de forma simultánea. Por ejemplo una conversación telefónica.

- **Comunicaciones *Half-Dúplex* y *Full-Dúplex***

Cuando dos equipos se comunican en una **LAN**, la información viaja normalmente en una sola dirección a la vez, dado que las redes en **banba base** usadas por las redes **LAN** admiten solo una señal. Esto de denomina comunicación ***half-duplex***. En cambio dos sistemas que se pueden comunicar simultaneamente en dos direcciónes están operando en modo ***full-duplex***. El ejemplo más comun de una red ***full-duplex*** es, una vez mas, el sistema telefónico. Ambas parte pueden hablar simultaneamente durante una llamada telefónica y cada parte puede oír a la otra a la vez. Un ejemplo de un sistema de comunicación ***half-duplex*** es la radio, como pueden ser los radiotransmisores, en los que solo una parte puede transmitir a la vez, y cada parte debe decir "cambio", para indicar que ha terminado de transmitir y está pasando de modo transmisión a modo recepción.

- **Modos de transmisión de datos**

Según el sentido de la transmisión podemos encontrarnos con tres tipos diferentes:

→ Simplex (sx)

Este modo de transmisión permite que la información discurra en un solo sentido y de forma permanente, con esta fórmula es difícil la corrección de errores causados por deficiencias de línea. Como ejemplos de la vida diaria tenemos, la televisión y la radio.

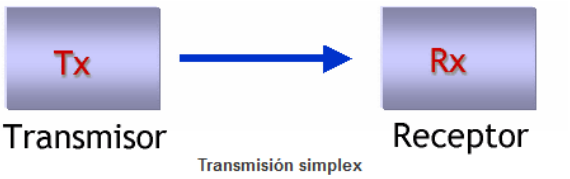


Figura 28. *Ejemplo de una comunicación Símplex.*

→ Half Dúplex (hdx)

En este modo, la transmisión fluye como en el anterior, o sea, en un único sentido de la transmisión de dato, pero no de una manera permanente, pues el sentido puede cambiar. Como ejemplo tenemos los Walkis Talkis.

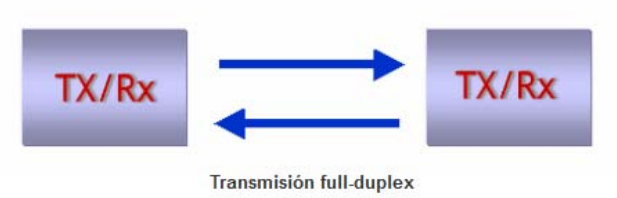


Figura 29. *Ejemplo de comunicación Half Dúplex.*

→ Full Dúplex (fdx)

Es el método de comunicación más aconsejable, puesto que en todo momento la comunicación puede ser en dos sentidos posibles y así pueden corregir los errores de manera instantánea y permanente. El ejemplo típico sería el teléfono.

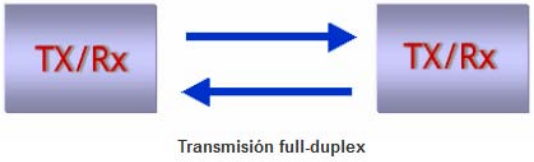


Figura 30. *Ejemplo de comunicaciones Full Dúplex.*

6.6 Características Eléctricas

La subnorma eléctrica de la **RS-232** es la **V28**. La norma fija una transmisión en modo común (*cada circuito tienen una referencia a tierra y esta es común para todos los circuitos*). Los circuitos son punto a punto, es decir, un driver con un sólo receptor de la señal.

La señal es bipolar con lógica invertida, utilizando los siguientes valores:

Señales de datos	"0"	"1"	
Emisor (necesario)	de 5 a 15	de -5 a -15	Voltios
Receptor (esperado)	de 3 a 25	de -3 a -25	Voltios
Señales de control	"Off"	"On"	
Emisor (necesario)	de -5 a -15	de 5 a 15	Voltios
Receptor (esperado)	de -3 a -25	de 3 a 25	Voltios

Figura 31. *Niveles de voltaje descritos en el estándar **RS-232.***

Puede verse que los voltajes del emisor y el receptor son diferentes. Esta definición de los niveles de voltaje compensa las perdidas de voltaje a través del cable. Las señales son atenuadas y distorsionadas a lo largo del cable. Este efecto es debido en gran parte a la capacidad del cable. En el estándar la capacidad máxima es de 2500 pf. La capacidad de un metro de cable es normalmente de 130 pf. Por lo tanto, la longitud máxima del cable esta limitada a unos 17 metros. Sin embargo, esta es una longitud nominal definida en el estándar y es posible llegar hasta los 30 metros con cables de baja capacidad o utilizando velocidades de transmisión bajas y mecanismos de corrección.

La **RS-232** es cortocircuitable. Esto quiere decir que, al menos teóricamente, los drivers de salida de las puertas disponen de un mecanismo de auto-protección contra sobrecalentamientos. La tensión máxima de operación es ±25voltios y la carga máxima es de 3Kohm a 7Kohm, con una corriente máxima de 500mA.

6.7 Características Mecánicas

En el estándar no se hace referencia al tipo de conector que debe usarse. Sin embargo los conectores más comunes son el **DB-25** (25 pines) y el **DB-9** (9 pines). El conector hembra debe estar asociado con el **DCE** y el macho con el **DTE.**

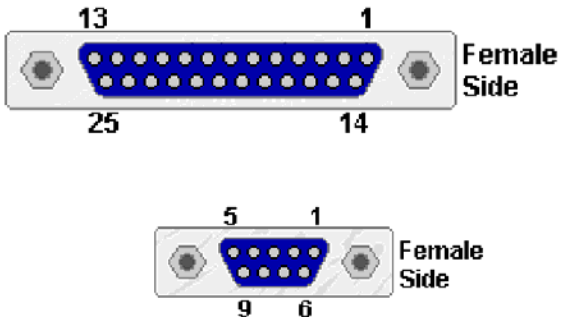
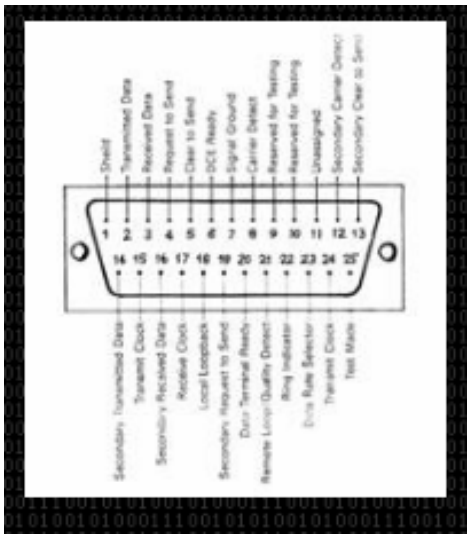


Figura 32. *Diagrama de los conectores **DB-25** y **DB-9**.*

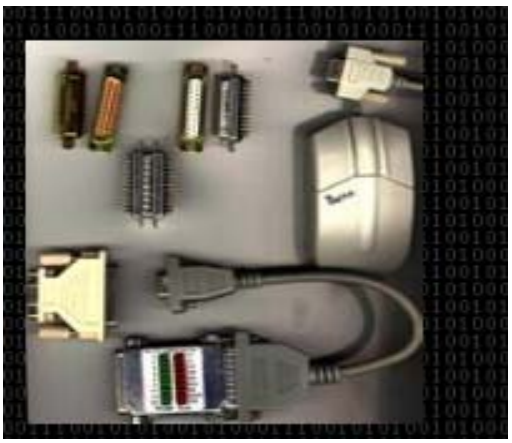


La conexión entre **DTE** y **DCE** es simple. Cada pin conecta con su par (*el 1 con el 1, el N con el N*) existen versiones de DB-25 para cable plano que simplifica el mecanizado de las conexiones. Cada pin tiene asignado una función tal y como se muestra en la figura anterior . Los nombres de las líneas están puestos desde el punto de

vista del **DTE**. Así, el pin 2 es la línea **TxD** (*transmisión de datos*) pero obviamente eso no es cierto en ambos equipos, sólo en el **DTE**. En el **DCE**, por el contrario, es la línea por la que recibe los datos del **DTE**.

Cuando sólo se utiliza la transmisión ***asíncrona***, sólo es necesario utilizar nueve líneas. Se puede utilizar el conector **DB-9**. Igualmente el macho es el **DTE** y la hembra el **DCE**. (*Desgraciadamente esta norma, y otras de la RS-232, no siempre es seguida por todos los fabricantes, razón por la cual no siempre es fácil manejar esta interface*).

La imagen muestra un conjunto de conectores **DB-25** machos (*arriba-izquierda*.) y hembras (*arriba-centro*). También pueden observarse conectores **DB-9** (*ratón*) y dispositivos adaptadores **DB-25** a **DB-9** (*abajo*), uno de ellos con un probador de **RS-232**.



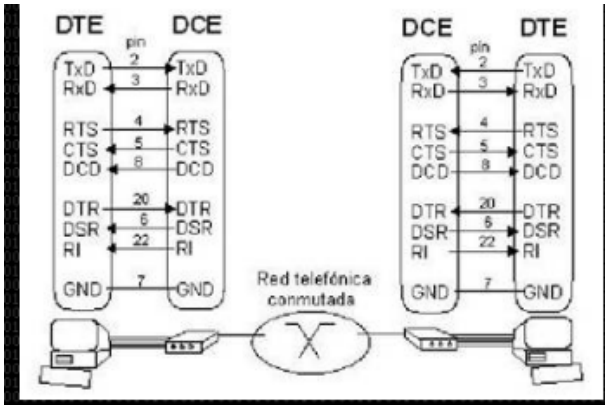
La longitud máxima del cable entre **DTE** y **DCE** depende de la calidad de éste y de la velocidad de transmisión utilizada. En principio la norma recomienda que no sea superior a 15 metros para una velocidad de 20Kbits/seg.

DB 25 Pin	DB 9 Pin	Nombre	EIA	CCITT	DTE - DCE	Nombre Formal
1		FG	AA	101		Protected Ground
2	3	TD	BA	103	→	Transmitted Data
3	2	RD	BB	104	←	Received Data
4	7	RTS	CA	105	→	Request to Send
5	8	CTS	CB	106	←	Clear to Send
6	6	DSR	CC	107	←	Data Set Ready
7	5	SG	AB	102		Signal Ground
8	1	DCD	CF	109	←	Data Carrie Detect
9						+ P
10						- P
11						No asignado
12		SDCD	SCF	122	→	Secondary Data Carrier
13		SCTS	SCB	121	→	Secondary Clear to Send
14		STD	SBA	118	←	Secondary Transmitted
15		TC	DB	114	←	Transmission Signal
16		SRD	SBB	119	←	Secondary Received Data
17		RC	DD	115	→	Received SIgnal Element Timing
18						No asignado
19		SRTS	SCA	120	→	Secondary Request to Send
20	4	DTR	CD	108.2	→	Data Terminal Ready
21		SQ	CG	110	←	Signal Quality Detector
22	9	RI	CE	125	←	Ring Indicator
23			CH/CI	111/112	← →	Data Signal Rate Selector
24			DA	113	←	Transmitter Signal
25						No asignado

Figura 33. *Relación de señales correspondientes a cada Pin.*

6.8 *Características Funcionales.*

La norma asíncrona la forman nueve líneas.



La línea **GND** conecta la masa de ambos equipos y no merece mayor comentario. Las restantes ocho líneas pueden ser agrupadas en tres bloques funcionales que se explican fácilmente si recordamos que la norma fue diseñada para conectar un PC (**DTE** típico) con un modem (**DCE** típico).

→ Primer Bloque

Lo denominaremos "*de establecimiento de conexión*". Está formado por las líneas:

- **DTR** (*Data Terminal Ready*). Terminal de datos preparado. (*El **PC** y su **RS232** están listos*).
- **DSR** (*Data Set Ready*). Equipo de comunicación preparado. (*El modem está listo*).
- **RI** (*Ring Indicator*). Indicador de llamada. (*El modem indica a su **PC** que ha recibido una llamada*).
- El objetivo es que ambos **PC**s sepan que se ha establecido un canal de comunicación (*normalmente a través de la línea telefónica*).
- Las líneas **DTR** y **DSR** del equipo local y del remoto deben estar activas (*set*) durante todo el proceso. *(De hecho cuando un **PC** desea dar por terminada una conexión basta con que, momentáneamente, desactive (reset) su **DTR***).
- La conexión se inicia manualmente (*el usuario llama con el teléfono al modem remoto*) o automáticamente (*el modem tiene capacidad de marcar un número de teléfono – dialling*) y se gestiona en los modems (*que negocian, de forma automática, los parámetros de transferencia como la velocidad, compresión, etc.*).
- Se asume que el usuario del **PC** que llama activará el proceso que va a utilizar la conexión (*un programa de transmisión de ficheros, por ejemplo*). En el **PC** llamado se asume que el proceso homólogo está ya activo (*porque, p.e., lo está permanentemente*) o se puede activar automáticamente al recibir de su modem la señal de **RI**. Sea como fuera, la conexión queda establecida. A partir de este momento los **PCs** pueden intercambiar información.

→ **Segundo Bloque** (*Control de Flujo*)

Estas líneas tienen sentido en el caso de que el canal de comunicación establecido tenga una gestión *half-duplex*.

Si el canal está establecido, el protocolo software de nivel de enlace de datos que se esté utilizando (*Xmodem*, *Ymodem*, *HDLC*,...) fijará cuál de los dos **DTEs** debe comenzar a hablar/transmitir.

Las líneas en este bloque son usadas de la siguiente manera:

- **RTS** (*Request To Send*). Petición de transmisión. El **PC** indica a su modem que quiere transmitir a la máquina remota.
- **CTS** (*Clear To Send*). Canal libre para la transmisión. El modem indica a su **PC** que puede transmitir. Previamente habrá transmitido una señal portadora por el canal de comunicación para avisar al otro modem que ocupa el canal.
- **DCD** (*Data Carrier Detected*). Detectada portadora. El modem indica a su **PC** que el canal de comunicación está ocupado por el equipo remoto.
- El **PC** que quiere transmitir activa **RTS**, entonces su modem manda una señal portadora (*sin modular, sin datos*) para avisar al modem remoto que se reserva el canal. Una vez reservado el canal comunica a su **DCE** que ya puede transmitir activando la línea **CTS**.
- Cuando un **PC** haya terminado de transmitir, desactivará **RTS**, el modem quitará la portadora y desactivará **CTS**. Entonces el otro modem podrá reservar el canal si su **PC** desea transmitir.
- En caso de que la gestión del canal sea *full-duplex* todo es más sencillo. Cuando un **PC** quiere transmitir activa su **RTS**. Automáticamente su modem le da paso activando **CTS**.

→ **Tercer Bloque (***Transmisión/Recepción de Datos***)**

El funcionamiento de las líneas de este bloque es obvio. Cuando un PC puede transmitir, lo hace por la línea:

- **TxD**. Transmisión de datos.

...y si está recibiendo datos lo hace por:

- **RxD**. Recepción de datos.

La transmisión serial de los datos, tal y como se ha explicado, con el bit de ***START***, de ***STOP***, etcétera, se produce en estas líneas.

6.9 Transmisión Serial.

- **Generalidades**

RS232 es el nombre del interfaz de comunicación serie más utilizado del mundo. La norma serie está disponible en prácticamente el 99% de los ordenadores.

La norma **RS232** fue originalmente diseñada para conectar terminales de datos con dispositivos de comunicación (como **modems** y **AITs**). Desde un principio, fue también utilizada para conectar casi cualquier dispositivo imaginable. Los usos de la **RS232** en el entorno doméstico son muchos y ampliamente conocidos. Desde la conexión del *ratón*, el *fax/modem*, *agendas electrónicas de bolsillo*, impresoras serie, grabadores de memoria (tipo **EPROM**), digitalizadores de *vídeo*, *radios* de **AM**/**FM**, etc. La lista sólo está limitada por la imaginación de los diseñadores.

En el entorno industrial el peso de la **RS232** es también muy importante. Si bien existen soluciones de comunicación serie más robustas y versátiles, como la **RS422** o la **RS475**, la **RS232** sigue siendo por su sencillez, su diseño económico y, sobre todo, por su gran difusión, la norma más frecuente. Así, es fácil ver cómo robots industriales, manipuladores, controles de todo tipo, utilizan la **RS232**. Existen hasta cafeterías industriales (*de las utilizadas en bares y restaurantes*) que disponen de una **RS232** para ser conectadas a un **PC** e informar de cuántos cafés han hecho en el transcurso del día, permitiendo al gerente de la empresa un control de caja, estadísticas de uso, etcétera.

- **Transmisión Serie/Paralelo**

Conceptualmente una transmisión *paralelo* consiste en utilizar simultáneamente varios circuitos de transmisión ***serie***. Dejando al margen problemas específicos de una transmisión en ***paralelo***, como puede ser el efecto ***crosstalk*** o interferencia inducida de símbolos, la transmisión *paralelo* es el recurso lógico cuando un solo circuito no proporciona un ancho de banda suficiente. Si en un diseño, un problema de transmisión puede resolverse (*a coste similar*) con una transmisión serie, esta opción es en principio deseable frente a una paralelo. Piénsese que en una transmisión con múltiples circuitos la probabilidad de fallo de línea y la necesidad de mantenimiento es proporcional al número de líneas utilizadas.

- **Transmisión Síncrona/Asíncrona**

Independientemente de si la transmisión es ***serie*** o ***paralelo***, ésta puede ser ***síncrona*** o ***asíncrona***. Para entender la diferencia es interesante fijarse en la etimología de las palabras. Ambas vienen del griego *cronos -tiempo* (*reloj*)-. ***Síncrona*** significa "*mismo reloj*" y *asíncrona* lo contrario, es decir, "*relojes distintos*".

Entre dos equipos, *emisor* y *receptor*, existe un problema básico en la identificación de los distintos símbolos (*bits en este caso*) que se transmiten por una línea de transmisión.

Supongamos dos computadores **A** y **B**, y una línea de transmisión por la que se comunican. Supongamos que **A** manda a **B** 50 bits a una velocidad de 1000 bits/segundo. Esto quiere decir que cada bit estará en la línea de transmisión una milésima de segundo. La máquina **B** necesita conocer este dato y necesita un reloj, o base de tiempos, que le permita medir con precisión esa milésima de segundo para saber cuándo está en la línea el segundo bit, el tercer bit, etcétera. El lector debe conocer que la forma normal en que el equipo receptor decide si un bit

es "**0**" o "**1**" es muestreando (*haciendo un muestreo de*) la línea de transmisión durante el intervalo del bit, preferiblemente a mitad del intervalo.

Es evidente que si el reloj utilizado por el *receptor* no mide el tiempo con precisión y la secuencia de bits es lo suficientemente larga, entonces cometerá un error en el muestreo de la línea e identificará una secuencia de bits incorrecta. Si, por ejemplo, el reloj receptor atrasa y cuando indica al sistema que ha pasado 1mseg en realidad ha pasado 1,1mseg (*un error del 10%, sin duda un poco exagerado*) entonces se producirá un primer error de muestreo en el 6º a 7º bit transmitido (*si asumimos que el primer bit lo muestreó correctamente en el centro del intervalo del bit*) (*ver figura*).

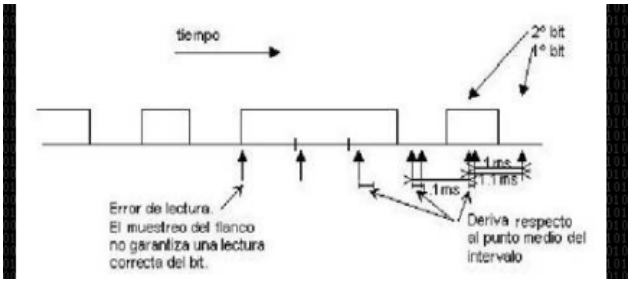
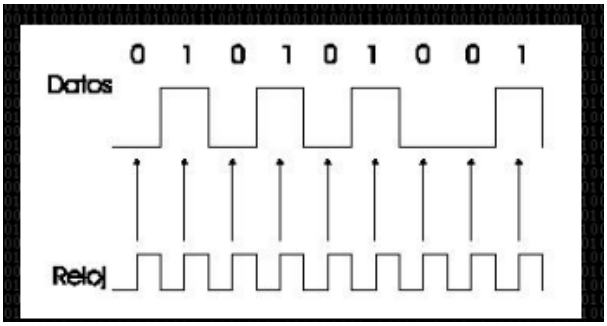


Figura 34.

La figura muestra claramente cuál es el problema. Debe quedar claro que aunque el planteamiento del ejemplo hace culpable al reloj del receptor, en una situación real encontraremos que, de usar dos relojes, es imposible garantizar que ambos midan el tiempo exactamente igual. Y aunque el error entre ambos sea mucho menor, nótese que si la secuencia de bits es lo suficientemente larga, el error de muestreo terminará por ocurrir.

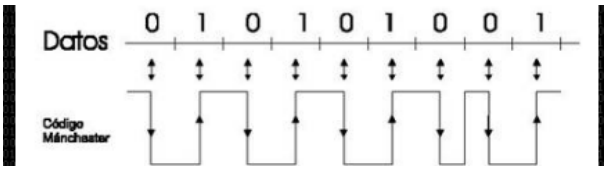
En la transmisión ***síncrona***... ¿Cómo se hace para que ambos computadores usen un único reloj?

i. Si se utiliza el reloj de una de las dos máquinas (*o un reloj tercero*) se puede transmitir la señal de reloj por una línea auxiliar a la otra máquina. La figura muestra como a partir de la señal de reloj el muestreo es siempre exacto.



ii. El emisor puede utilizar una codificación para los datos de las denominadas "***auto-reloj***", como por ejemplo el código Máncheste (*utilizado p.e. en redes locales Ethernet*).

La figura muestra una secuencia de bits codificada en código ***Mánchester***.



Se puede observar que la codificación ***Mánchester*** tiene la propiedad de que existe siempre una transmisión en la mitad del intervalo del bit. El receptor aprovecha esta propiedad para sincronizarse en cada bit. Es como si hubiera un reloj entre los datos que marca el ritmo del muestreó que debe hacer el receptor, de ahí el nombre de *auto-reloj*.

En la transmisión ***asíncrona***,... ¿Cómo se hace para sincronizar al principio de cada carácter?

Cuando el emisor no transmite, en el periodo entre caracteres, la línea se mantiene a "**1**" lógico. Cuando decide transmitir un carácter, primero transmite un "**0**" que se denomina bit de **START** y sirve para que el receptor sincronice (*empieza a contar tiempos desde ese momento*). El instante de sincronismo es el flanco de bajada de la señal (*ver figura*). Tras el bit de **START** se transmiten los bits de datos y después es obligatorio al menos un bit de **STOP** a "**1**" lógico. La secuencia se repite tantas veces como caracteres se transmitan. Obsérvese que este mecanismo de sincronización con el bit de **START** impide que la deriva de muestreo por diferencias entre los relojes continúe en el siguiente carácter. Se asume que la deriva de muestreo no debe ser tan grande que provoque un error de muestreo en los bits de cada carácter.

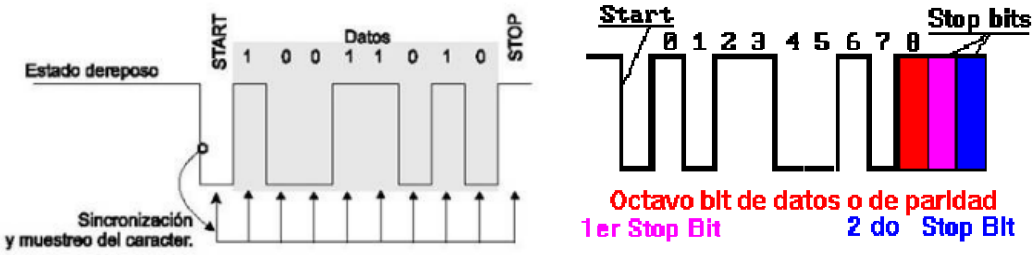


Figura 35. *Secuencia de bits **Start** y **Stop**.*

Hoy, se configuran casi siempre los equipos en 8 bits de *datos*, sin *paridad* y sin bit de *stop*.

6.10 Conexionado DTE-DTE: Null Módem.

Como ya se ha mencionado, es frecuente que la norma **RS-232** se utilice para otros propósitos distintos de los originales. Uno de los más frecuentes es conectar un **DTE** con otro **DTE**. En este caso el cableado normal **DTE** a **DCE** no tiene sentido y estaríamos cortocircuitando las líneas de salida. Existe una solución, un cableado cruzado que se conoce como **Null-modem**. Es fácil de entender si lo analizamos usando los tres bloques funcionales.

Bloque 1:

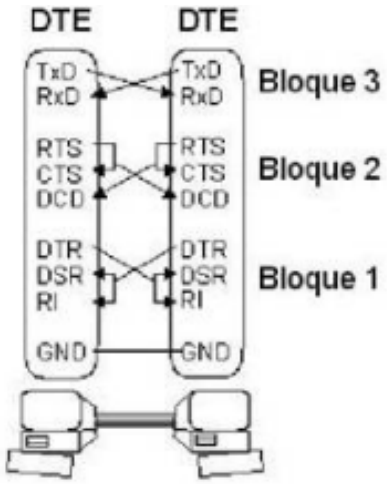
Conectar **DTR** con **DSR** remoto y **RI** remoto.

De esta manera cuando un **PC** activa su **RS-232** se lo comunica al remoto.

Bloque 2:

RTS con **CTS** local y **DCD** remoto.

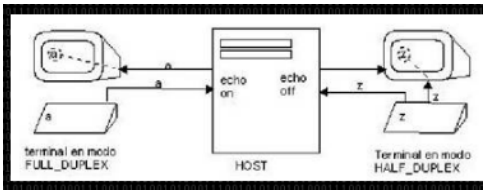
Para entender la lógica de este cableado debe observarse que hay dos líneas independientes de transmisión de datos, una en cada sentido. Por lo tanto la comunicación es potencialmente *Full-duplex*. Esto implica que cada **DTE** puede transmitir cuando lo desee, independientemente de que el otro **DTE** lo esté haciendo o no. Por lo tanto cuando un **PC** quiere transmitir activa su **RTS**, esto activa también su propia **CTS** lo que le permite transmitir inmediatamente. Además indica que está transmitiendo a la máquina remota activando el **DCD** remoto.



Bloque 3:

TxD con **RxD** remoto.

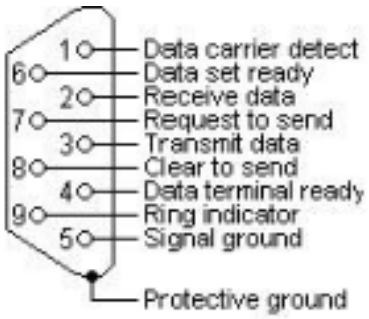
Como habíamos mencionado, la comunicación es potencialmente *Full-Duplex*.



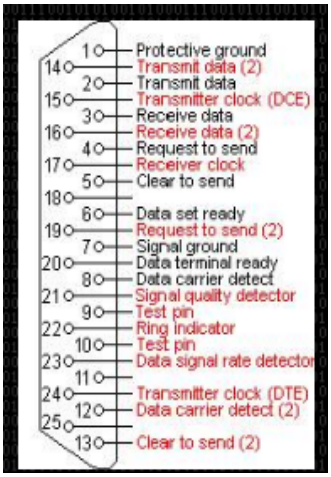
La ventaja del método **FD** es que permite al usuario comprobar si ha habido error de transmisión. Si el código en pantalla es el deseado, también es el que ha leído el host. Esto no pasa en **HD**. Si en **HD** existiera un error de transmisión y el código de la tecla pulsada no llega o llega mal al host, el usuario no puede, en principio, saberlo.

→ Diagramas de conexionado de los diferentes cables.

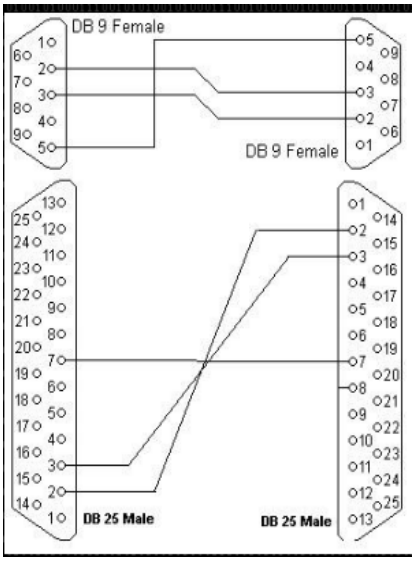
RS232 DB-9 Pin Assignment.



RS232 DB-25 Pin Assignment.

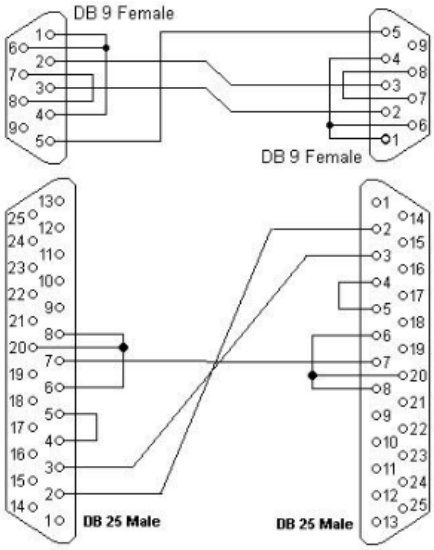


Simple Null-Módem without handshaking



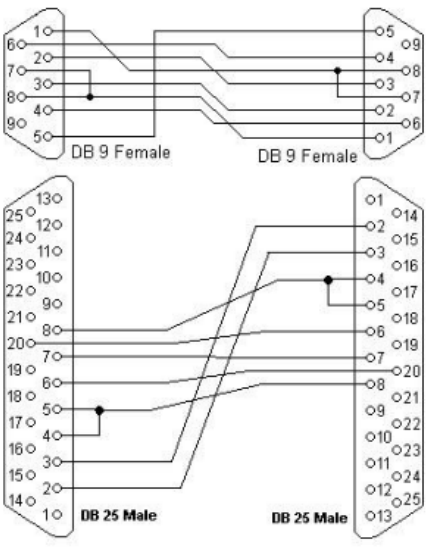
Computer 1			Computer 2		
Pin DB-25P	**Pin DB-9P**	**Signal**	**Signal**	**Pin DB-9P**	**Pin DB-25P**
2	3	TD	RD	2	3
3	2	RD	TD	3	2
7	5	SG	SG	5	7

Null módem whit loop back handshaking



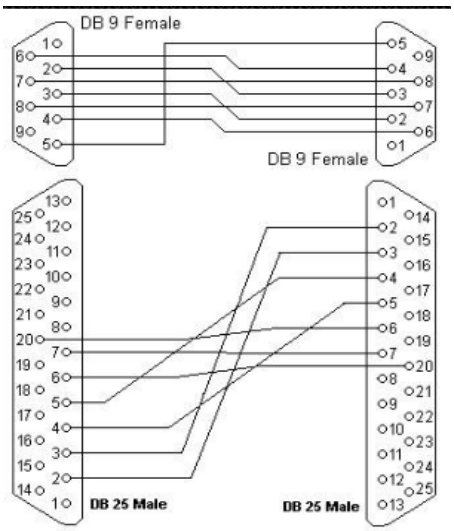
Computer 1			Computer 2		
Pin DB-25P	**Pin DB-9P**	**Signal**	**Signal**	**Pin DB-9P**	**Pin DB-25P**
2	3	TD	RD	2	3
3	2	RD	TD	3	2
7	5	SG	SG	5	7
Internal connections for each computer					
Pin DB-25P		**Pin DB-9P**		**Signal**	
4+5		7+8		RTS+CTS	
20+6+8		4+6+1		DTR+DSR+CD	

Null modem with partial handshaking



Computer 1			Computer 2		
Pin DB-25P	**Pin DB-9P**	**Pin DB-25P**	**Pin DB-9P**	**Pin DB-25P**	**Pin DB-9P**
2	3	2	3	2	3
3	2	3	2	3	2
7	5	7	5	7	5
4+5	7+8	RTS-CTS	CD	1	8
8	1	CD	RTS-CTS	7+8	4+5
20	4	DTR	DSR	6	6
6	6	DSR	DTR	4	20

Null modem with full handshaking



Computer 1			Computer 2		
Pin DB-25P	**Pin DB-9P**	**Pin DB-25P**	**Pin DB-9P**	**Pin DB-25P**	**Pin DB-9P**
2	3	TD	RD	2	3
3	2	RD	TD	3	2
7	5	SG	SG	5	7
4	7	RTS	CTS	8	5
5	8	CTS	RTS	7	4
20	4	DTR	DSR	6	6
6	6	DSR	DTR	4	20

6.11 Contro de Flujo.

Existen dos posibilidades de control de flujo de datos con la ***RS232***: Una *hardware* mediante las líneas **RTS**/**CTS** y otra *software* **XON**/**XOFF**.

RTS/**CTS**: la línea **CTS** indica al **PC** si puede transmitir o no. En aplicaciones como la conexión de un **PC** a una impresora serie (*dispositivo este normalmente bastante lento*) la línea **CTS** está gobernada por la impresora para impedir que el **PC** desborde su buffer de entrada.

XON/**XOFF**: Otra posibilidad es usar el protocolo software **XON**/**XOFF** que consiste en lo siguiente:

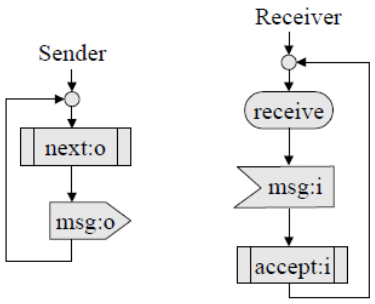
- Cuando la impresora está dispuesta para recibir datos (*buffer de entrada vacío o casi vacío*) transmite al **PC** la marca **XON** (***XON*** *y* ***XOFF*** *son códigos* ***ASCII*** *predefinidos*).
- Si el **PC** transmite demasiado rápido para la impresora y el buffer está próximo a llenarse, entonces se manda la marca **XOFF**.
- El **PC** transmite sólo si la última marca recibida fue **XON**.
- Dependiendo de las características de los equipos a conectar se puede hacer un control de flujo **RTS**/**CTS**, **XON**/**XOFF**, ambos o ninguno.

Cuando se utilizan ambos, normalmente es porque hay que controlar dos buffers de recepción, el del dispositivo físico (***UART***), que se hace por **RTS**/**CTS**, y el buffer de la aplicación que está recibiendo los datos, que se hace con **XON**/**XOFF**.

i. ¿Para qué sirve el control de flujo?.

- Asegurar que los datos no se transmiten más rápido de lo que se pueden procesar.
- Optimizar el uso del canal.
- Evitar la saturación de ciertos enlaces del canal.

ii. Modelo Básico: Protocolo sin control de flujo.



- Funciona si el emisor es más rápido que el receptor.
- Diseño de sistemas concurrentes: No hacer superposiciones de las velocidades relativas de procesos concurrentes.

iii. X-ON y X-OFF

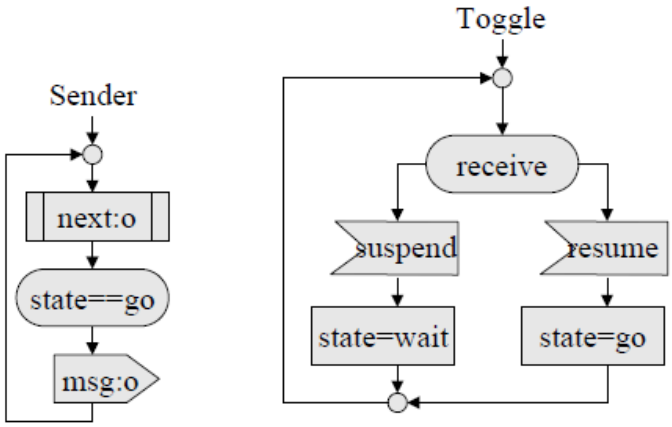
- Utiliza dos mensajes de control: resume resume y suspend.
- No requiere negociación previa.

Así, suponiendo un canal sin errores y el vocabulario:

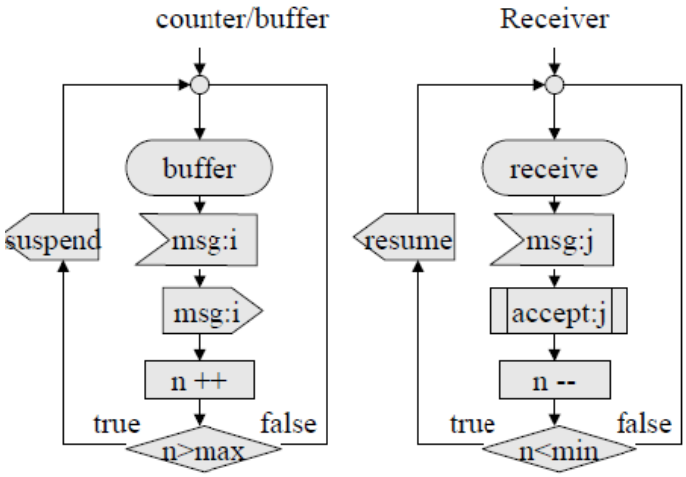
V = {*msg*, *suspend*, *resume*}

Las reglas de procedimiento para el *emisor* y para el *receptor* serán:

→ **Emisor:**



→ **Receptor:**



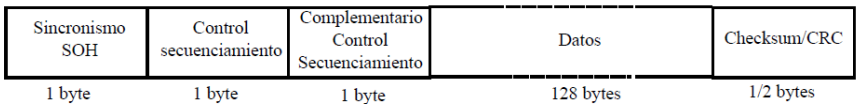
Los mensajes de datos pasan de *counter/buffer* a *receiver* a través de una cola interna.

Pegas: El protocolo depende de las propiedades del canal:

- Si un mensaje *suspend* se retrasa o pierde hay ***overflow***.
- Si un mensaje *resume* se pierde los 4 procesos se colapsan.

iv. X-MODEM

X-Modem es un protocolo ***ARQ*** (*automatic repeat request, solicitud de repetición automática*) de parada y espera (***Stop&Wait***) con un campo de datos de longitud fija (128 bytes). La trama del protocolo consta de cinco campos. Empieza con una cabecera de inicio de trama cuyo contenido es **SOH** (*Start Of Header*) de valor 01H. El segundo campo indica el número de secuencia de la trama y su valor se inicializa a 1. El contenido del tercer campo es el complemento a 1 del número de secuencia de la trama.



El último campo de la trama tiene como misión el control de errores. En X-Modem, durante el establecimiento de la conexión, los terminales negocian la técnica a utilizar entre las dos posibilidades siguientes:

- ***Checksum***. Se obtiene sumando el contenido del campo de datos y expresándolo en módulo 256. Su longitud será por tanto de 1 byte.

- ***CRC***. Consta de 2 bytes y se calcula a partir de un polinomio de grado 16 normalizado por el CCITT (x16 + x12 + x5 + 1).

En ***X-Modem*** una transferencia de información consta de tres fases: establecimiento, transferencia de datos y liberación. Para poder llevarlas a cabo, se hace necesaria la utilización de ciertos caracteres de control, incluidos dentro del código ***ASCII***: **SOH**, **ACK**, **NAK**,**CAN** y **EOT**.

El establecimiento de la comunicación corre a cargo del receptor, que es el responsable de gestionar la comunicación manteniendo un flujo de tramas. Inicialmente, el receptor genera continuamente **NAK**´s, permaneciendo a la espera de recibir la primera trama. En el momento en que el emisor reciba correctamente un **NAK**, enviará su primera trama. La fase de establecimiento finaliza si el receptor decodifica correctamente el **SOH** de esta primera trama.

Durante la fase de transferencia el emisor espera siempre un reconocimiento positivo (**ACK**) o negativo (**NAK**) de cada una de las tramas que va enviando (*de ahí el tipo de protocolo: parada y espera*). Este reconocimiento ha de ser enviado por el receptor, una vez comprobados los campos de secuenciamiento y de control de errores. La fase de transferencia se aborta bruscamente si el emisor genera un paquete **CAN** (*Cancel*).

La liberación de la conexión se realiza ordenadamente cuando el emisor genera un **EOT** (*End Of Transmission, 04H*), recibiendo un *ACK* por parte del receptor. Nuevas versiones del protocolo permiten enviar varios ficheros.

Se debe observar que durante la transferencia de un fichero mediante ***X-Modem***, el control de flujo es intrínseco al protocolo, ya que el receptor no permite al emisor que transmita más información hasta que no haya procesado la última trama. En realidad, el mecanismo de parada y espera es ya de por sí un método de control de flujo. De todas formas, algunas implementaciones del protocolo permiten combinar este control de flujo con el realizado mediante la técnica hardware (***RTS/CTS***).

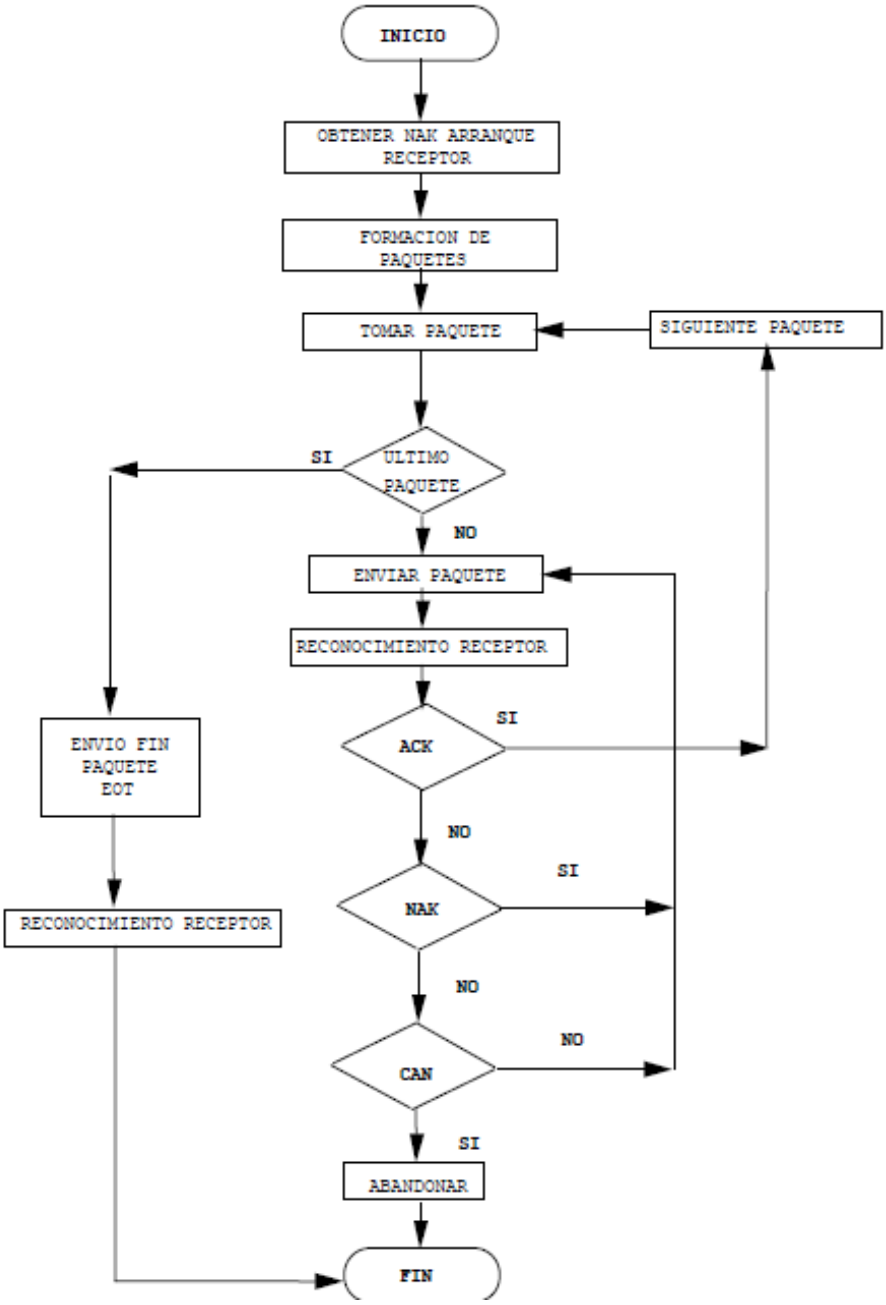


Figura 36. *X-MODEM Emisor.*

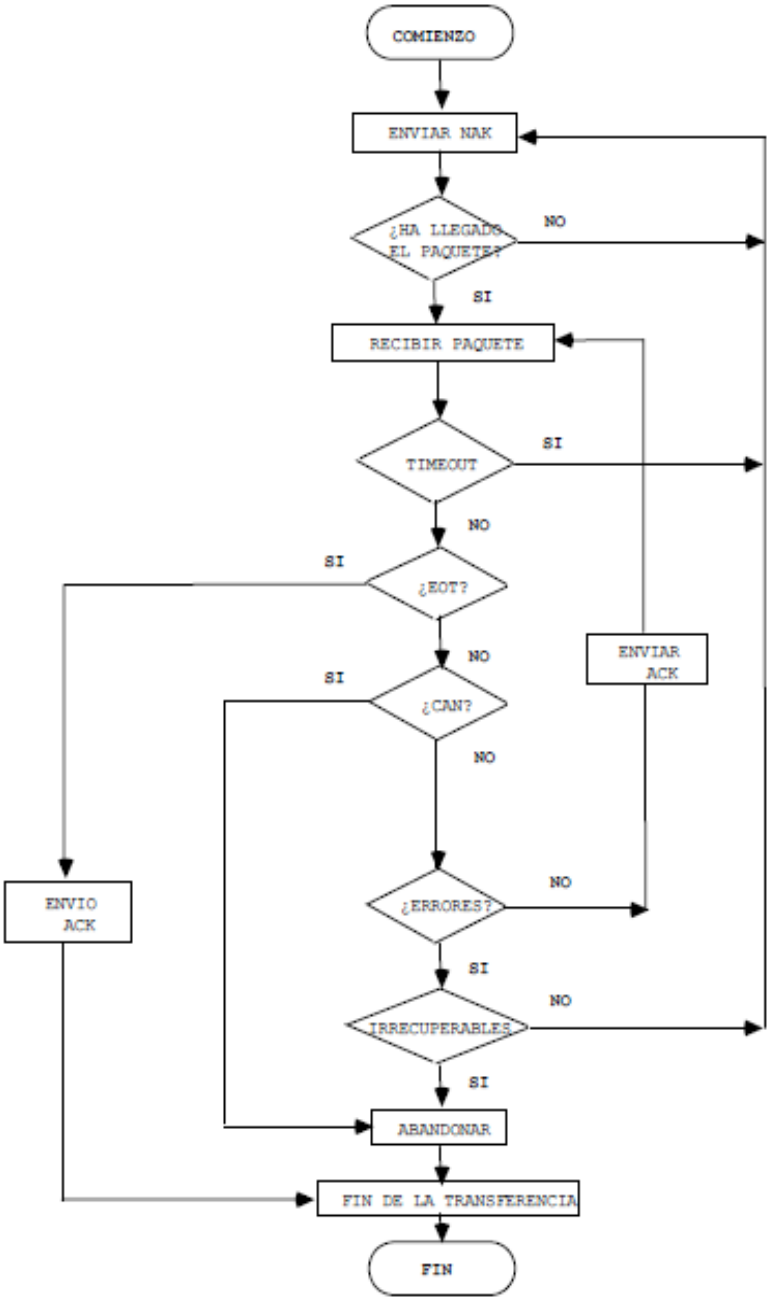


Figura 37. *X-Modem Receptor.*

Capítulo 7.

7 Conclusiones y líneas futuras.

La principal conclusión es que en este proyecto fin de carrera se han logrado todos los objetivos que inicialmente estaban expuestos al principio respecto de la práctica 1 de la asignatura Fundamentos de Telemática.

A modo de resumen, los objetivos alcanzados son:

- Se ha construido un simulador para poder entender los fundamentos de conexiones, tanto síncronas como asíncronas y el manejo del puerto serie.

- También se ha añadido un poco más de realismo a la hora de trabajar con el control de flujo. Al poder realizar la práctica tanto con control de flujo **Xon/Xoff** o **sin** control de flujo, y así poder entender mejor el concepto de control de flujo.

- Hemos creado una interfaz gráfica para poder realizar las prácticas desde casa, ya que no todo el mundo tiene en su poder un osciloscopio, PODRS232 y conectores de puerto serie para poder realizar físicamente las prácticas.

- La interfaz se ha tratado de hacer lo más real posible, para que se parezcan, tanto las prácticas realizadas en el laboratorio, como las simuladas en el PC.

- El código fuente se deja abierto y disponible, para así dejar el programa abierto para posibles mejoras futuras, y a la depuración de posibles errores.

- Se ha conseguido el objetivo de la portabilidad, ya que el programa al estar hecho en JAVA, se puede ejecutar en cualquier plataforma (Windows, Linux, Macintosh, Solaris, etc).

- Uno de los objetivos principales del proyecto que también se han cumplido, es que el APPLET es muy intuitivo a la hora de realizar, tanto el conexionado como a la hora de empezar a realizar simulaciones con el osciloscopio.

Las líneas futuras para este programa dependerán de las nuevas necesidades que se encuentren a los largo de las prácticas donde se utilice este simulador. En prácticas sucesivas de los cursos posteriores a la asignatura de Fundamentos de Telemática, tanto profesores como alumnos podrían ver que mejoras se pueden realizar en función de las necesidades didácticas, completando el ciclo de vida de este software.

La especificación de este Proyecto Final de Carrera consistió en una serie de objetivos claramente definidos y que se han conseguido en su totalidad. El definir unas líneas futuras para él va a depender de las necesidades didácticas que se puedan ir encontrando durante la utilización de la herramienta a lo largo de los cursos académicos. Se tiene que producir un proceso de retroalimentación en el que los usuarios (alumnos y profesores), dependiendo de hipotéticas necesidades futuras que el simulador pudiera ofrecer. Esta retroalimentación permitiría modificaciones en el programa para ajustarlo a necesidades futuras.

Capítulo 8.

8 Pasos para realizar un archivo ".jar".

Los archivos "***.java***" que contienen el código, al compilarlos generan los archivos "***.class***" que son los que hacen funcionar la aplicación. Para tener juntas estas clases y las imágenes (si las hubiera dentro del proyecto) del programa, lo mejor es empaquetarlo todo en un archivo "***.jar***", en el cuál solo hay que hacer doble click (como ejecutable que és) para que se inicie la aplicación.

Los pasos para crear el archivo "***.jar***" son los siguientes:

i. Entrar en las propiedades de "Mi PC", en "**Opciones Avanzadas → Variables de entorno**", y aquí creamos una variable llamada **Path**, donde su valor será el directorio *bin* del JDK que tengamos instalado.

ii. Editamos el archivo MANIFEST.MF (que he adjuntado con el CD del proyecto), y aquí indicamos cuál es la clase principal (main class) del programa. Por ejemplo:

```
Manifest-Version: 1.0
Main-Class: RS232
```

iii. Editar el archivo Comandos-jar.bat e indicamos cómo se llamará el archivo "*.jar*" que se desea construir, los archivos "*.class*" y todas las imágenes si hubiese inmágenes insertadas en el proyecto. La forma de editarlo es la siguiente:

```
jar cmvf manifest.mf RS232.jar *.class
```

El asterisco en .class quiere decir que se incluirán todos los archivos que contengan esta extensión.

iv. Finalmente, hacemos doble click en el archivo Comandos-jar.bat. Debe estar absolutamente todo en una misma carpeta, sino el archivo .bat no lo incluirá.

Después de hacer doble click, en el archivo "*.jar*" se encuentra construido en la misma carpeta donde estamos, listo para funcionar.

Recomendaciones:

- Tener instalado el JDK 1.7.0 o superior.
- Si en el sistema está instalado el WinRAR, desasociarlo con las extensiones "*.jar*", ya que si no lo hacemos, el archivo "*.jar*" en vez de ejecutarse, se abre para explorar lo que hay dentro, es decir, el sistema lo trata como un archivo comprimido a explorar, no como un ejecutable.

Capítulo 9.

9 Pasos para poder firmarlo con un certificado digital.

Como un Applet por defecto tiene permisos un tanto más restrictivos que una aplicación normal, la razón de esto es para evitar que un usuario malintencionado escriba Applet con código malicioso (por ejemplo, alguno que borre un disco duro sin autorización del usuario), los publique en Internet y posteriormente cualquier persona que ejecute dicho Applet se vea afectada.

En algunas ocasiones es necesario que un Applet acceda a ciertos recursos locales con el fin de realizar su función cabalmente, solo que debido a las restricciones anteriormente mencionadas, no va a ser posible a menos de que sea ***firmado digitalmente***.

Por este motivo para que el Applet pueda leer archivos del sistema de archivos de nuestro propio ordenador donde esta corriendo el navegador, hay que generar un archivo '***.jar***' que contenga el Applet y firmarlo con un certificado digital ('aunque sea auto-firmarlo'). Cuando se ejecuta se pide la aprobación del usuario para su ejecución y una vez atorgada dicha aprobación, ya tienes permisos para acceder al sistema de archivos de tu propio ordenador, y así poder realizar todos los puntos de de esta práctica.

¿Qué sucede si ejecuto un Applet no firmado en el navegador.?

Si tratamos de ejecutar en el navegador un Applet que accede a los recursos locales y que no esté firmado no se podrá ejecutar, y en la consola de la ***JVM*** aparecerá un mensaje de error:

```
java.security.AccessControlException: access denied
```

El firmado del Applet lo podemos realizar tanto en sistemas operativos como Linux como en Windows. Aquí dejaré comentado la manera de realizarlo en los dos sistemas operativos.

Pasos para realizar el firmado del Applet en Linux:

Para firmar el Applet en Linux seguiremos los siguientes pasos:

- Abrimos un Terminal.
- Ahora ejecutamos la siguiente línea de código:

```
xxx:~$ keytool –genkey –alias pod
```

keytool	para generar los certificados.
-genkey	
-alias pod	alias a utilizar para la firma del Applet.

- Tendrás que responder una serie de preguntas que te van a realizar.

- Una vez respondidas las preguntas anteriores, ejecutamos la siguiente línea de código (nos pedirá el password anteriormente introducido 'tiene que ser el mismo.'):

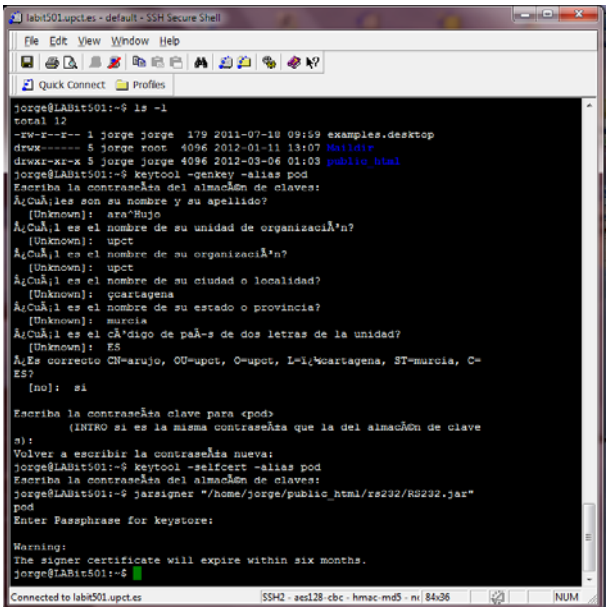
xxx:~$ keytool –selfcert –alias pod

- Y por último, ejecutamos el ***jarsigner*** que es el archivo mediante el cual se pueden firmar los archivos '***.jar***'.

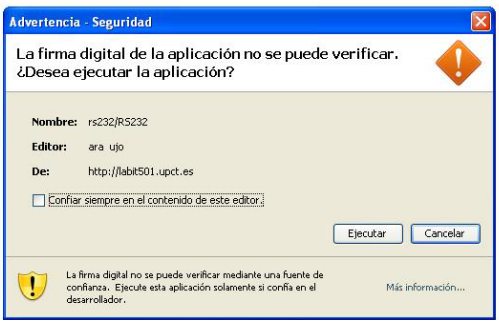
xxx:~$ jarsigner "/home/Jorge/public_html/rs232/RS232.jar" pod

jarsigner para firmar los archivos '.jar'.

- A continuación muestro la captura de pantalla del firmado del Applet.



- Cuando ejecutes la aplicación te aparecerá un mensaje parecido al siguiente:



Pasos para realizar el firmado del Applet en Windows:

Para firmar el Applet en Windows seguiremos los siguientes pasos:

- Para firmar digitalmente un Applet en Windows, afortunadamente, el mismo **jdk** nos provee de las herramientas necesarias para crearlo y posteriormente aplicarlo a nuestro archivo '***.jar***'.

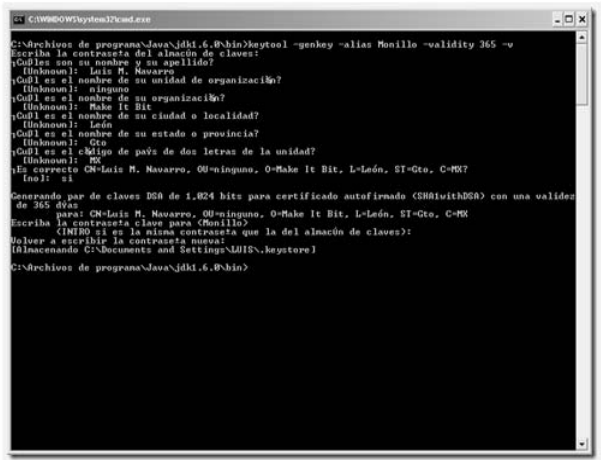
- Dentro de la carpeta **bin** que posee el **jdk** ('en Windows se encuentra normalmente en *c:\archivos de programa\jdkx.xx.xxx*'), existen un par de aplicaciones que nos servirán, dichas aplicaciones son las siguientes:

  keytool: para generar los certificados.
  Jarsigner: para firmar los archivos ***'.jar'***.

- Ahora para generar el certificado que posteriormente nos servirá para firmar los Applet solo basta con ejecutar el siguiente comando desde la línea de comandos (en Windows puedes abrir la línea de comandos desde inicio→Ejecutar→escribes **cmd** y pulsas **enter**):

  keytool –genkey –alias pod

- Después de ejecutar el código se te harán alguna preguntas (como lo muestro en la siguiente figura), debes contestarlas a tu preferencia.

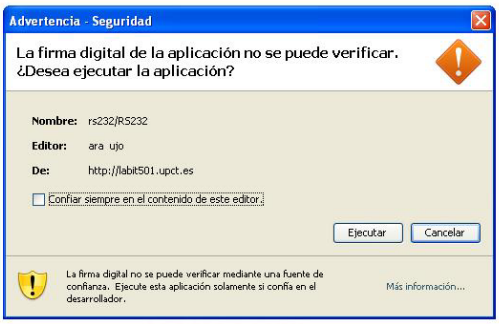


- Ahora ya tenemos nuestro certificado, ahora la pregunta es ¿**cómo lo utilizo para firmar el Applet**? Una vez que tenemos nuestro certificado basta con ejecutar el siguiente comando para firmarlo:

  jarsigner.exe RS232.jar pod –verbose

- Después de todo esto te pedirá escribir la contraseña que definimos anteriormente, si es correcto el Applet quedará firmado y listo para ejecutarse en el navegador.

- Cuando ejecutemos nuestra aplicación nos aparecerá el siguiente mensaje, y ya está listo para ejecutarse.



RECOMENDACIONES:

- Para el correcto funcionamiento de la aplicación, teneis que descargar el software de:

  http://java.sun.com/j2se

- También la máquina virtual JAVA:

  http://java.com/es/download/windows_xpi.jsp?locale=es

Anexos.

Anexo A.
Manual de usuario.

A. Introducción

El programa ha sido creado mediante *JAVA*, por lo que en los ordenadores donde se utilice se deberá tener instalado el ***JRE***(*Java Runtime Environment*) para que pueda ejecutarse, y éste incluye la *Java Virtual Machine* (***JVM***) y la **API.** Todo está incluido en el ***J2SE*** (*Java 2 Standard Edition*), donde puede instalarse cada cosa por separado.

Si lo que desea es modificar el código fuente del programa, no es suficiente tener instalado el ***JRE***, sino que se necesita el ***J2SE***.

El software está disponible gratuitamente en http://java.sun.com/

La aplicación consiste en un único archivo con extensión "*.jar*" (***RS232.jar***), donde se encuentran las clases que componen el programa.

B. Objetivos

El objetivo final de esra práctica es que el alumno entienda el funcionamiento de las comunicaciones serie, en este caso el de las comunicaciones asíncronas, y así poder ayudar a los alumnos al correcto entendimiento de las comunicaciones.

C. Anexo

Uno de los interfaces de nivel físico más utilizados es el **RS232** que fue estandarizado por **EIA** en 1962. El interfaz consta de 25 contactos formando circuitos que trabajan en modo no balanceado, manejando todos ellos una masa común. La transmisión de información se realiza en serie tanto en modo asíncrono como síncrono.

Las especificaciones eléctricas definen con un margen de tensión entre -5 y -15 voltios el nivel lógico 1 ó marca para los circuitos de datos, y **OFF** para las líneas de control. El rango de tensión entre +5 y +15 voltios define el nivel lógico 0 ó espacio para los circuitos de datos y **ON** para los circuitos de control. El tiempo de tránsito entre estados tiene que ser menor que 1 mseg. En estado de reposo (Idle) la línea permanece en el estado lógico 1.

D. Página Principal

El programa se inicia desde una web (http://labit501.upct.es/~jorge/RS232.html), desde aquí tendremos acceso a todo el conjunto de ventanas de las cuales dispone nuestro Applet.

La página tiene el siguiente aspecto:

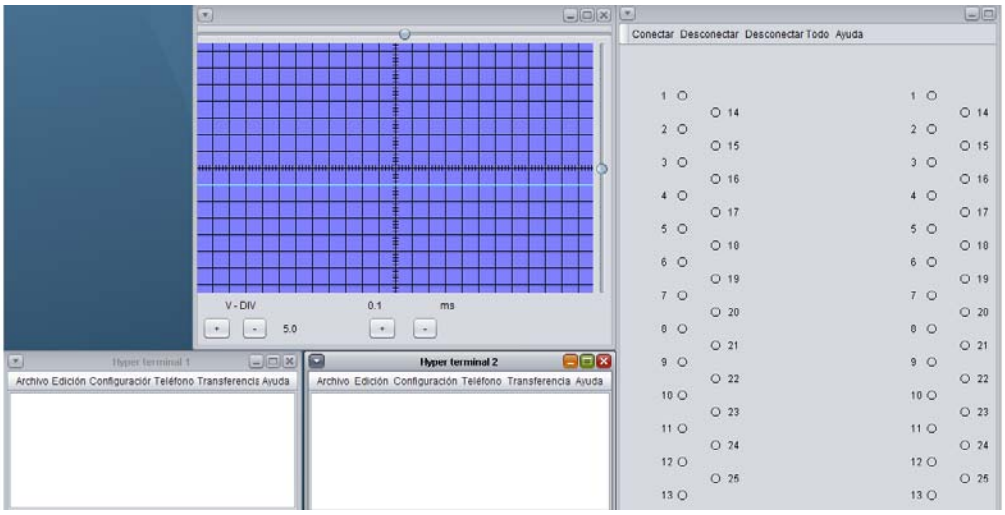
Si se elige la opción de EJECUTAR APPLET, nos mostrará el entorno del Applet a ejecutar, y a partir de eso momento empezar a realizar la práctica en cuestión.

Si se elige la opción de PDF de la práctica, esta opción nos da la posibilidad de descargarnos el enunciado de la práctica en versión .pdf para así poder realizar las prácticas.

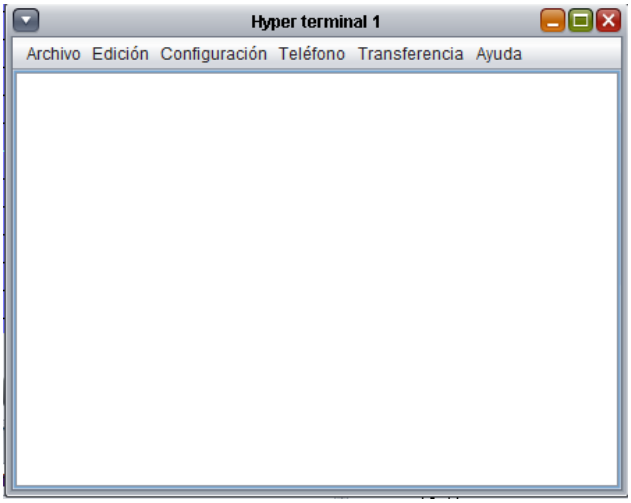
Si se elige la opción de MARCO TEÓRICO, te redirige a una página Web, en la cual te muestra el enunciado de la práctica en versión on-line, por si no quiere descargarse el enunciado de la práctica en versión pdf.

Una ves hecho click en el *botón* ***EJECUTAR APPLET*** nos redirige a otra ventana web en la cual ya se ve el aspecto del *Applet* y desde el cual empezar a realizar las prácticas.

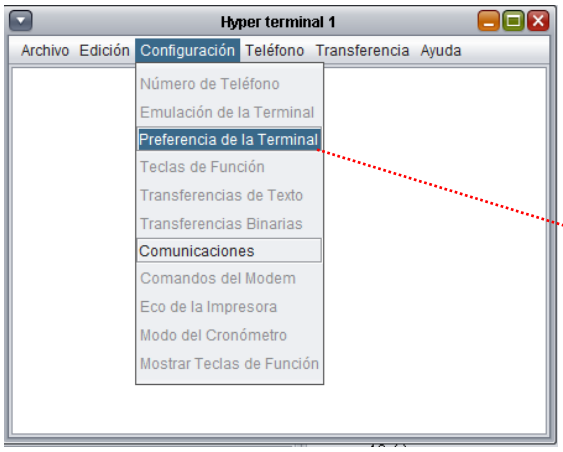
El aspecto del *Applet* es el siguiente:



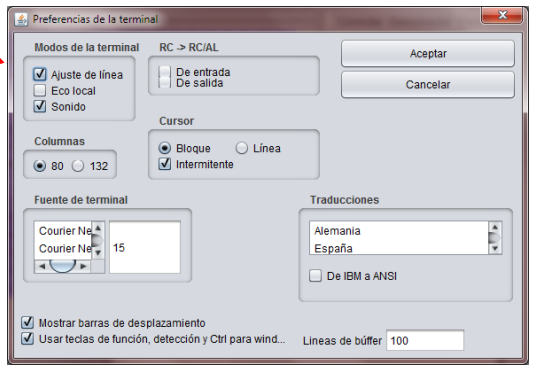
E. HyperTerminal

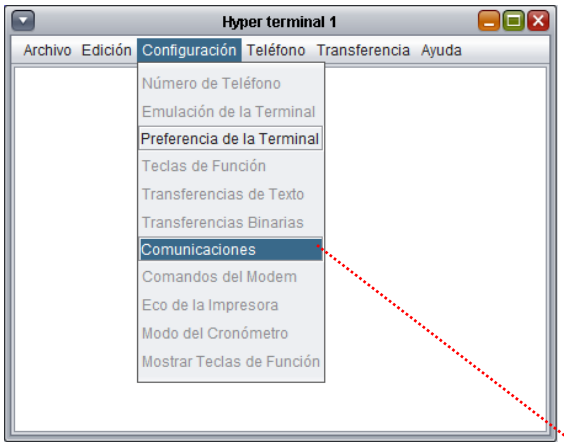


El objetivo de esta opción, es la configuración de la comunicación para que se lleve a cabo una buena transferencia de información.

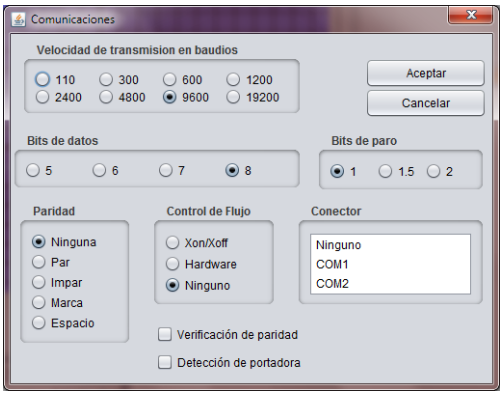


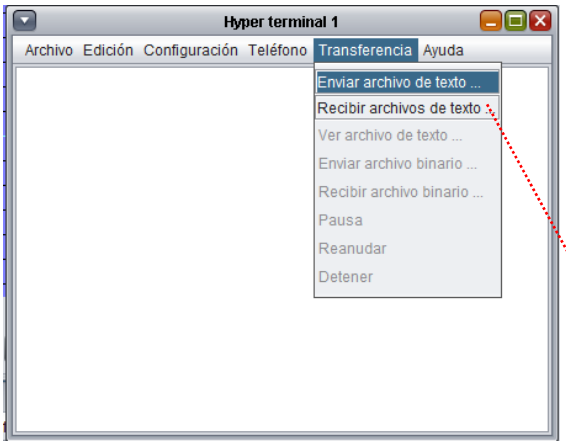
Si hacemos click en ***Configuración*** y nos vamos a ***Preferencia de la Terminal***, con esto nos permite fijar los parámetros de visualización.



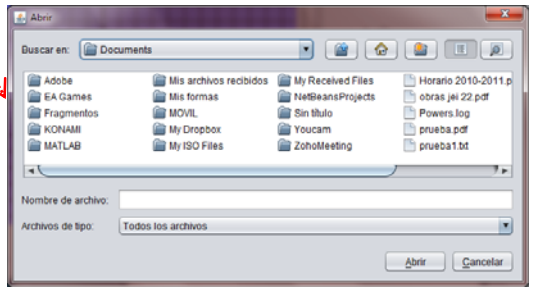


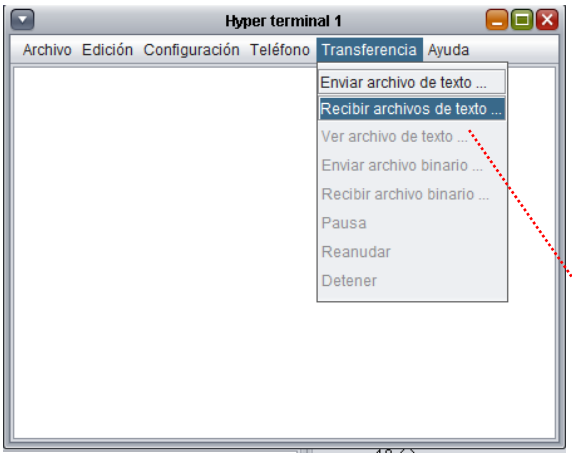
Si hacemos click en ***Configuración*** y nos vamos a ***Comunicaciones***, con esto para poder utilizar correctamente el programa terminal, tendremos que configurar los parámetros de la comunicación serie, así como el puerto que se usará. Deben de conecerse a priori los parámetros de transmisión que utiliza el dispositivo al cuál nos vamos a conectar.



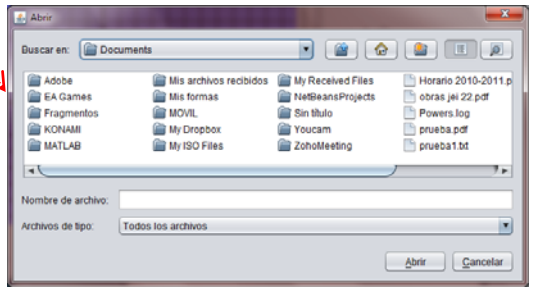


Si hacemos click en ***Transferencia*** y nos vamos a ***Enviar archivo de texto***, con esto nos permite enviar un archivo de texto. Aparece una ventana en la que pregunta por un nombre de fichero y si se desea realizar algun tipo de mapeo.

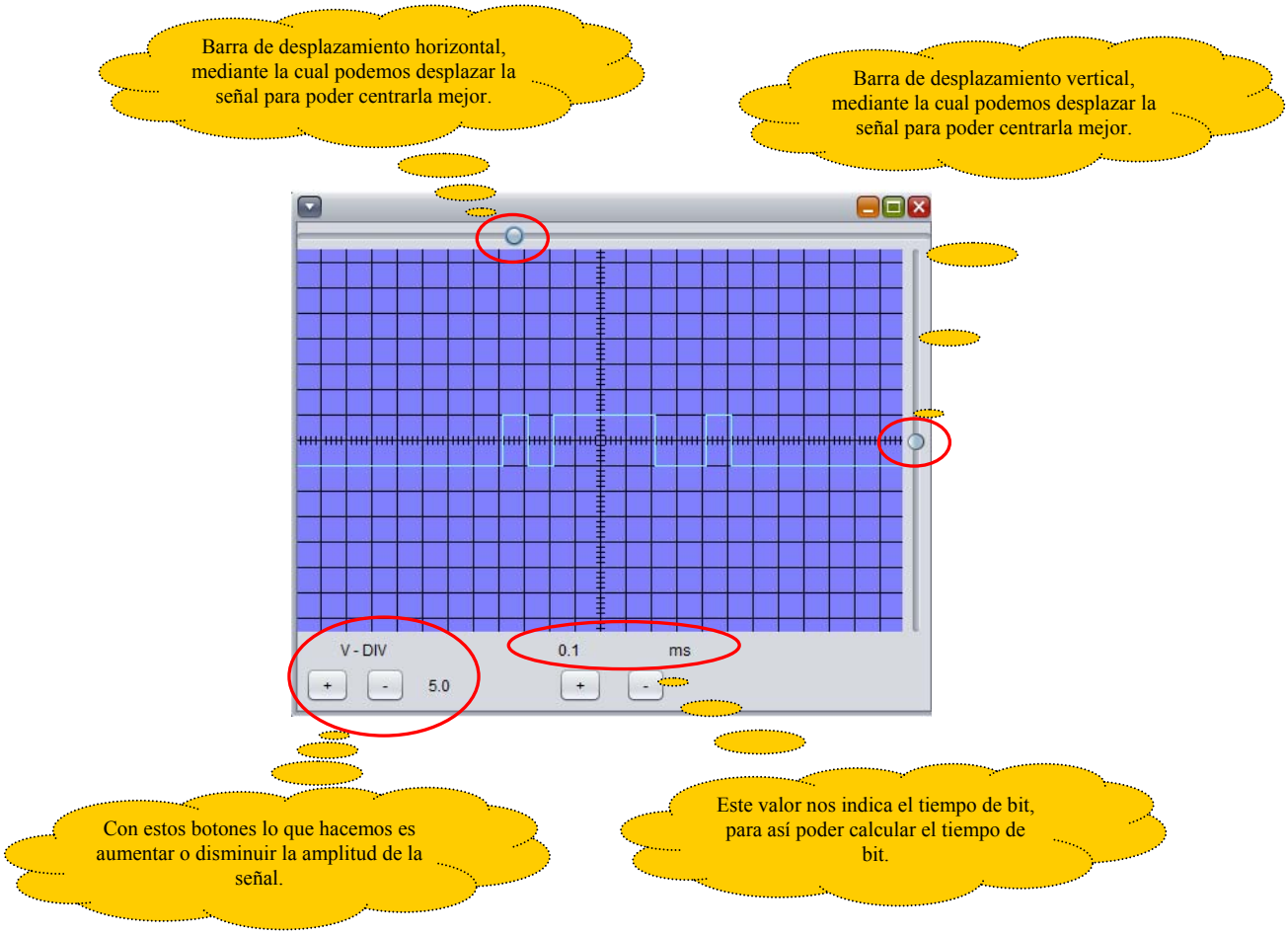




Si hacemos click en ***Transferencia*** y nos vamos a ***Recibir archivo de texto***, con esto no spermite especificar el nombre de un fichero donde se recibirá un fichero de texto.



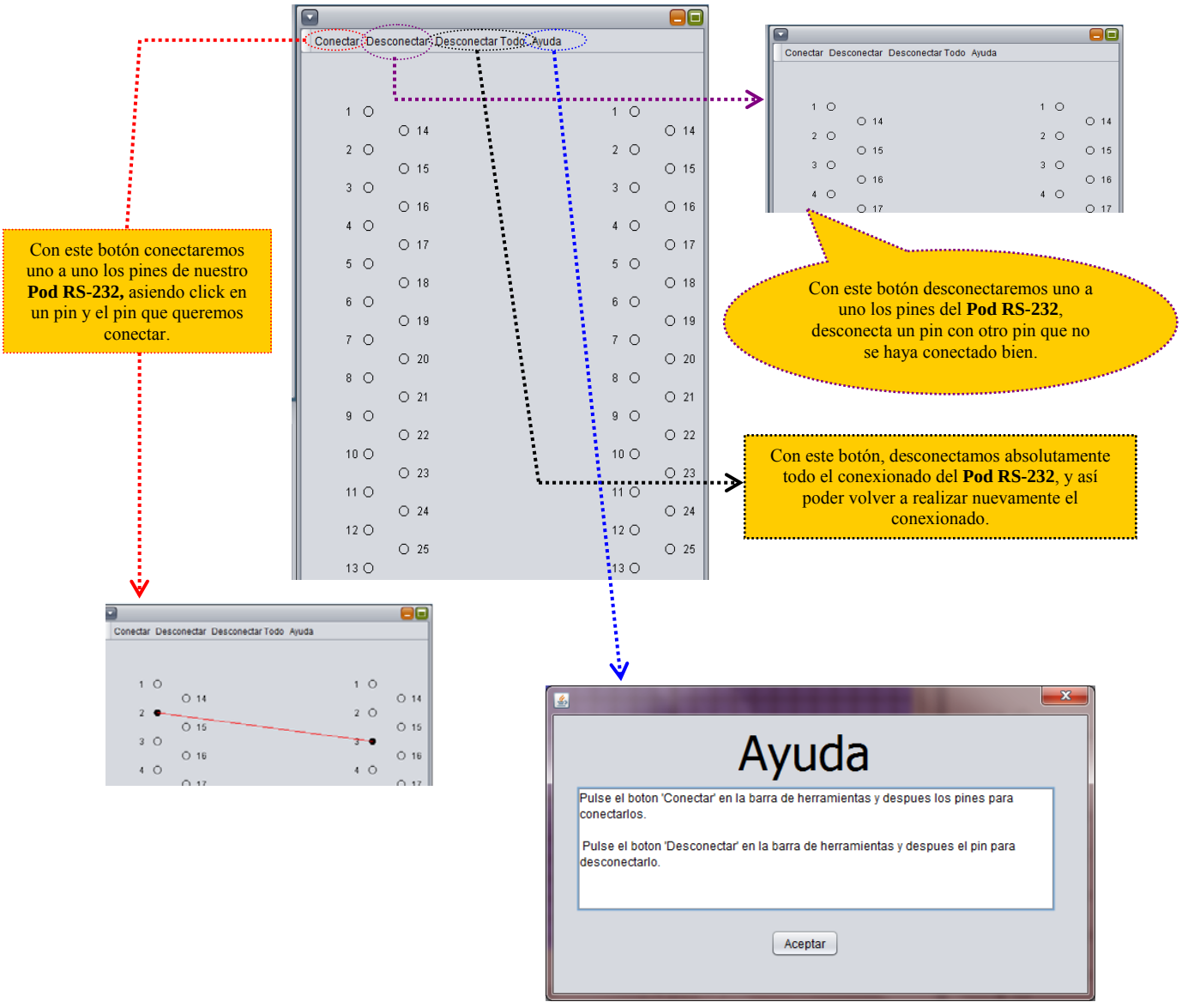
F. Osciloscopio



En el ***Osciloscopio*** no hay mucho que explicar, porque toda la programación es interna, aquí no necesitamos conectar nada (*en referencia a* ***masa****'tierra' y al* ***canal***), porque todos los parámetros que hay dentro del ***osciloscopio*** vienen de configuraciones, tanto del ***HyperTerminal*** como del conexionado del ***Pod RS-232***.

G. Pod RS-232

Una vez finalizada la configuración, se pasará a realizar los distintos conexionados entre los puertos serie **ETD**s formando un enlace punto a punto vía **RS-232**, para ello, utilicese el **Pod RS-232**.



Bibliografía.

→ Manual de prácticas de Fundamentos de Telemática.

http://labit501.upct.es/~fburrull/docencia/FundamentosTelematica/practicas/P1_ComunicacionesSerieAsincronas-NivelFisicoydeEnlace.pdf

→ Manuales de clase Java. Clase Graphics:

http://www.javaya.com.ar/detalleconcepto.php?codigo=130&inicio=40

→ Manuales de métodos de JAVA.

http://objetos2.wordpress.com/2010/04/24/clase-window-listener/

→ Apuntes de clase de la asignatura Fundamentos de Telemática, de 1º curso de Ingeniería Telemática de la Universidad Politécnica de Cartagena.

→ Apuntes de clase de la asignatura de Ingeniería de Protocolos, de 3º curso de Ingeniería de Telemática de la Universidad Polotécnica de Cartagena.

→ Apuntes de clase de la asignatura de Fundamentos de Programación, de 2º curso de Ingeniería Telemática de la Universidad Politécnica de Cartagena.

→ Transparencias de D. Jorge Martínez Bauset, Departamento de Comunicaciones de la Universidad Politécnica de Valencia.

→ Stallings, W.,"Comunicaciones y Redes de Computadores". Prentice-Hall Iberia, 2000 (6ª Ed.). ISBN: 84-205-2986-9.

→ Halsall, F., "Data Communications, Computer Networks and Open Systems". Ed. Addison-Wesley, 1996 (4ª Ed.).

→ Seifert, “The Switch Book”. Ed. Jonh Willey & Sons, 2000. ISBN: 0-471-34586-5.

Anexo B.
Código fuente mas significativo.

El código fuente de la aplicación está disponible en el *CD-ROM* del proyecto.

Sin embargo, aquí se describe parte del código fuente de la aplicación que puede resultar más interesante.

B.1 RS-232.

```
package rs232;


public class RS232 extends javax.swing.JApplet {

    /** Initializes the applet RS232 */

    public void init() {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(RS232.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(RS232.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(RS232.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(RS232.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }

        /* Create and display the applet */
        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {

                public void run() {
                    initComponents();
                    hyperTerminal1.setTitle("Hyper terminal 1");
                    hyperTerminal2.setTitle("Hyper terminal 2");
```

```
                hyperTerminal1.inicializar(pODRS2321);
                hyperTerminal2.inicializar(pODRS2321);
                pODRS2321.inicializar(hyperTerminal1.uart,hyperTerminal2.uart,osciloscopio1);
            }
        });
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

    jDesktopPane1 = new javax.swing.JDesktopPane();
    hyperTerminal1 = new rs232.HyperTerminal();
    hyperTerminal2 = new rs232.HyperTerminal();
    pODRS2321 = new rs232.PODRS232();
    osciloscopio1 = new rs232.Osciloscopio();

    hyperTerminal1.setVisible(true);
    hyperTerminal1.setBounds(20, 20, 496, 331);
    jDesktopPane1.add(hyperTerminal1, javax.swing.JLayeredPane.DEFAULT_LAYER);

    hyperTerminal2.setVisible(true);
    hyperTerminal2.setBounds(50, 90, 496, 331);
    jDesktopPane1.add(hyperTerminal2, javax.swing.JLayeredPane.DEFAULT_LAYER);

    pODRS2321.setVisible(true);
    pODRS2321.setBounds(610, 10, 519, 638);
    jDesktopPane1.add(pODRS2321, javax.swing.JLayeredPane.DEFAULT_LAYER);

    osciloscopio1.setVisible(true);
    osciloscopio1.setBounds(70, 200, 510, 415);
    jDesktopPane1.add(osciloscopio1, javax.swing.JLayeredPane.DEFAULT_LAYER);
```

```
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jDesktopPane1,javax.swing.GroupLayout.Alignment.TRAILING,
            javax.swing.GroupLayout.DEFAULT_SIZE, 939, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jDesktopPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 559, Short.MAX_VALUE)
        );

    // Variables declaration - do not modify
    private rs232.HyperTerminal hyperTerminal1;
    private rs232.HyperTerminal hyperTerminal2;
    private javax.swing.JDesktopPane jDesktopPane1;
    private rs232.Osciloscopio osciloscopio1;
    private rs232.PODRS232 pODRS2321;
    // End of variables declaration
}
```

B.2 UART.

```
package rs232;

public class UART {
    static int PARIDAD_NINGUNA=0;
    static int PARIDAD_PAR=1;
    static int PARIDAD_IMPAR=2;
    static int PARIDAD_MARCA=3;
    static int PARIDAD_ESPACIO=4;

    boolean ctrl_flujo=false; //False se refiere al modo X-Modem y si es true para Null-Modem

    public int velocidad=9600;
    public int bitsDatos=8;
    public double bitsParo=1.0;
    public int paridad ;

    private char caracterRecibido;
    private char caracterEnviado;

    private boolean errorEnRecepcion=false;

    private PODRS232 pod;
    private HyperTerminal hyperTerminal;

    public UART(PODRS232 pod,HyperTerminal hyperTerminal){
        this.pod=pod;
        this.hyperTerminal=hyperTerminal;
    }

    public void enviar(char caracter){
        caracterEnviado=caracter;

        boolean[] senal=new boolean[24];
```

```
boolean[] senalDefinitiva;
for(int i=0;i<24;i++){
    senal[i]=false;
}

int ascii=(int)caracter;
senal[0]=false;
senal[1]=false;
int nBits=0;
int n=2;
for(int i=0;i<bitsDatos;i++){
    senal[n]=(ascii%2==1);
    senal[n+1]=(ascii%2==1);
    nBits=nBits+ascii%2;
    ascii=ascii/2;
    n=n+2;
}

if(paridad==PARIDAD_PAR){
    senal[n]=nBits%2==1;
    senal[n+1]=nBits%2==1;
    n=n+2;
}
if(paridad==PARIDAD_IMPAR){
    senal[n]=nBits%2==0;
    senal[n+1]=nBits%2==0;
    n=n+2;
}

if(paridad==PARIDAD_MARCA){
    senal[n]=true;
    senal[n+1]=true;
    n=n+2;
}
if(paridad==PARIDAD_ESPACIO){
    senal[n]=false;
    senal[n+1]=false;
```

```
            n=n+2;
        }
        if(bitsParo==1.0){
            senal[n]=true;
            senal[n+1]=true;
            n=n+2;
        }
        if(bitsParo==1.5){
            senal[n]=true;
            senal[n+1]=true;
            senal[n+2]=true;
            senal[n+3]=true;
            n=n+4;
        }
        if(bitsParo==2.0){
            senal[n]=true;
            senal[n+1]=true;
            senal[n+2]=true;
            senal[n+3]=true;
            n=n+4;
        }
        senalDefinitiva=new boolean[n];
        for(int i=0;i<n;i++){
            senalDefinitiva[i]=senal[i];
        }
        pod.aceptarSenal(this,velocidad,senalDefinitiva);

    }

    public void aceptarSenal(int velocidad,boolean[] senal){
        errorEnRecepcion=false;
        if(this.velocidad==velocidad){
            int ascii=0;
            int nBits=0;
            int orden=1;
            int n=2;
            for(int i=0;i<bitsDatos;i++){
```

```
    if(senal[n+1]){
        ascii=ascii+orden;
        nBits++;
    }
    orden=orden*2;
    n=n+2;
}
if(paridad==PARIDAD_PAR || paridad==PARIDAD_IMPAR){
    if(senal[n]){
        nBits++;
        n=n+2;
    }
}

if(paridad==PARIDAD_PAR){
    if(nBits%2!=0){
        errorEnRecepcion=true;
    }
}
if(paridad==PARIDAD_IMPAR){
    if(nBits%2!=1){
        errorEnRecepcion=true;
    }
}
if(paridad==PARIDAD_MARCA){
    if(!senal[n-2]){
        errorEnRecepcion=true;
    }
}
if(paridad==PARIDAD_ESPACIO){
    if(senal[n-2]){
        errorEnRecepcion=true;
    }
}

if(bitsParo==1.0){
    if(!senal[n] || !senal[n+1]){
```

```
                    errorEnRecepcion=true;
                }
                n=n+2;
            }
            if(bitsParo==1.5){
                if(!senal[n] || !senal[n+1] || !senal[n+2]){
                    errorEnRecepcion=true;
                }
                n=n+3;
            }
            if(bitsParo==2.0){
                if(!senal[n] || !senal[n+1] || !senal[n+2] || !senal[n+2]){
                    errorEnRecepcion=true;
                }
                n=n+4;
            }
            if(!errorEnRecepcion){
                caracterRecibido=(char)ascii;
                hyperTerminal.aceptarCaracter(caracterRecibido);
            }
        }
        else{
            errorEnRecepcion=true;
        }
    }

    public void setVelocidad(int velocidad){
        this.velocidad=velocidad;
        pod.setPeriodo(1.0/(double)velocidad);
    }
}
```

B.3 HyperTerminal.

```
package rs232;


import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.BufferedWriter;

public class HyperTerminal extends javax.swing.JInternalFrame {
    private static int IDLE=0;
    private static int ESPERANDO_TRANSMITIR_ARCHIVO=1;
    private static int TRANSMITIENDO_ARCHIVO=2;
    private static int ESPERANDO_RECIBIR_ARCHIVO=3;
    private static int RECIBIENDO_ARCHIVO=4;
    private char SOH=(char)1;
    private char ACK=(char)6;
    private char NAK=(char)21;
    private char CAN=(char)24;
    private char EOT=(char)4;

    public UART uart;
    public boolean ecoLocal;
    private File archivo;
    private FileReader fr;
    private BufferedReader br;
    private FileWriter fw;
    private BufferedWriter bw;
```

```
private int estado=IDLE;


/** Creates new form HyperTerminal */
public HyperTerminal() {
    initComponents();
}

public void inicializar(PODRS232 pod){
    uart=new UART(pod,this);
}

public void setVelocidad(int v){
    uart.setVelocidad(v);
}

public void setBitsDatos(int bitsDatos){
    uart.bitsDatos=bitsDatos;
}

public void setBitsParo(double bitsParo){
    uart.bitsParo=bitsParo;
}

public void setParidad(int p){
    uart.paridad=p;
}

public void setCtrl_Flujo(boolean cf){
    uart.ctrl_flujo=cf;
}
```

```
public int getVelocidad(){
   return uart.velocidad;
}

public int getBitsDatos(){
   return uart.bitsDatos;
}

public double getBitsParo(){
   return uart.bitsParo;
}

public int getParidad(){
   return uart.paridad;
}

public boolean getCtrl_Flujo(){
   return uart.ctrl_flujo;
}

private void mostrarComunicaciones(){
   Comunicaciones com=new Comunicaciones(null,true,this);
   com.setVisible(true);
}

private void mostrarEnviarArchivosTexto(){
   javax.swing.JFileChooser fc=new javax.swing.JFileChooser();
   fc.setFileSelectionMode(javax.swing.JFileChooser.FILES_ONLY);
   int ret=fc.showOpenDialog(null);
   archivo=fc.getSelectedFile();
```

```
    if(ret==javax.swing.JFileChooser.APPROVE_OPTION && archivo!=null){
      try{
        fr=new FileReader(archivo);
        br=new BufferedReader(fr);
        estado=ESPERANDO_TRANSMITIR_ARCHIVO;
        uart.enviar(SOH);
      }
      catch(java.io.IOException e){
        System.out.println(e.getMessage());
      }
    }
}

private void mostrarRecibirArchivosTexto(){
    javax.swing.JFileChooser fc=new javax.swing.JFileChooser();
    fc.setFileSelectionMode(javax.swing.JFileChooser.FILES_ONLY);
    int ret=fc.showOpenDialog(null);
    archivo=fc.getSelectedFile();
    if(ret==javax.swing.JFileChooser.APPROVE_OPTION && archivo!=null){
      try{
        fw=new java.io.FileWriter(archivo);
        bw=new BufferedWriter(fw);
        estado=RECIBIENDO_ARCHIVO;
        uart.enviar(ACK);
      }
      catch(java.io.IOException e){ System.out.println("Error al escribir archivo :"+e.getMessage());
      }
    }
}

private void mostrarEnviarArchivoBinario(){
    mostrarEnviarArchivosTexto();
```

```
}

private void mostrarRecibirArchivoBinario(){
   mostrarRecibirArchivosTexto();
}

public void aceptarCaracter(char caracter){
   if(estado==IDLE){
      if(caracter==SOH){
         estado=ESPERANDO_RECIBIR_ARCHIVO;
      }
      else{
         jTextPane1.setText(jTextPane1.getText()+String.valueOf(caracter));
      }
   }
   if(estado==RECIBIENDO_ARCHIVO){
      if(caracter==EOT){
         try{
            bw.close();
            fw.close();
            estado=IDLE;
         }
         catch(java.io.IOException e){ System.out.println(e.getMessage());
         }
      }
      else{
         try{
            bw.write(caracter);
         }
         catch(java.io.IOException e){
            System.out.println(e.getMessage());
         }
      }
```

```
    }
    if(estado==ESPERANDO_TRANSMITIR_ARCHIVO && caracter==ACK){
        estado=TRANSMITIENDO_ARCHIVO;
        char c;
        int i;
        try{
            i=br.read();
            while(i!=-1){
                c=(char)i;
                uart.enviar(c);
                i=br.read();
            }
            uart.enviar(EOT);
            estado=IDLE;
        }
        catch(java.io.IOException e){
            System.out.println(e.getMessage());
        }
    }
}

private void enviarCaracterDeArchivo(){
    try{
        char c=(char)br.read();
        if(c!=-1){
            uart.enviar((char)br.read());
        }
        else{
            uart.enviar(EOT);
            estado=IDLE;
            br.close();
            fr.close();
        }
```

```
    }
    catch(java.io.IOException e){
        System.out.println(e.getMessage());
    }
}
@SuppressWarnings("unchecked")

private void initComponents() {

    jScrollPane1 = new javax.swing.JScrollPane();
    jTextPane1 = new javax.swing.JTextPane();
    jMenuBarHyperTerminal = new javax.swing.JMenuBar();
    jMenuArchivo = new javax.swing.JMenu();
    jMenuItemNuevo = new javax.swing.JMenuItem();
    jMenuItemAbrir = new javax.swing.JMenuItem();
    jMenuItemGuardar = new javax.swing.JMenuItem();
    jMenuItemGuardarComo = new javax.swing.JMenuItem();
    jMenuItemEspecificarImpresora = new javax.swing.JMenuItem();
    jMenuItemSalir = new javax.swing.JMenuItem();
    jMenuEdicion = new javax.swing.JMenu();
    jMenuItemCopiar = new javax.swing.JMenuItem();
    jMenuItemPegar = new javax.swing.JMenuItem();
    jMenuItemEnviar = new javax.swing.JMenuItem();
    jMenuItemSeleccionarTodo = new javax.swing.JMenuItem();
    jMenuItemLimpiarBuffer = new javax.swing.JMenuItem();
    jMenuConfiguracion = new javax.swing.JMenu();
    jMenuItemNumTlfn = new javax.swing.JMenuItem();
    jMenuItemEmulaTreminal = new javax.swing.JMenuItem();
    jMenuItemPreferencias = new javax.swing.JMenuItem();
    jMenuItemTeclasFuncion = new javax.swing.JMenuItem();
    jMenuItemTransTexto = new javax.swing.JMenuItem();
    jMenuItemTransBinarias = new javax.swing.JMenuItem();
    jMenuItemComunicaciones = new javax.swing.JMenuItem();
```

```
jMenuItemComModem = new javax.swing.JMenuItem();
jMenuItemEcoImpresora = new javax.swing.JMenuItem();
jMenuItemModoCronometro = new javax.swing.JMenuItem();
jMenuItemMostrarTeclasFuncion = new javax.swing.JMenuItem();
jMenuTelefono = new javax.swing.JMenu();
jMenuItemMarcar = new javax.swing.JMenuItem();
jMenuItemColgar = new javax.swing.JMenuItem();
jMenuTransferencia = new javax.swing.JMenu();
jMenuItemEnvArchTexto = new javax.swing.JMenuItem();
jMenuItemRecArchTexto = new javax.swing.JMenuItem();
jMenuItemVerArchTexto = new javax.swing.JMenuItem();
jMenuItemEnvArchBinario = new javax.swing.JMenuItem();
jMenuItemRecArchBinario = new javax.swing.JMenuItem();
jMenuItemPausa = new javax.swing.JMenuItem();
jMenuItemReanudar = new javax.swing.JMenuItem();
jMenuItemDetener = new javax.swing.JMenuItem();
jMenuAyuda = new javax.swing.JMenu();
jMenuItemIndice = new javax.swing.JMenuItem();
jMenuItemBuscarAyudaSobre = new javax.swing.JMenuItem();
jMenuItemUsoAyuda = new javax.swing.JMenuItem();
jMenuItemAcercaTerminal = new javax.swing.JMenuItem();

setClosable(true);
setIconifiable(true);
setMaximizable(true);
setResizable(true);
setTitle("Hyper terminal");

jTextPane1.setEditable(false);
jTextPane1.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        jTextPane1KeyPressed(evt);
    }
```

```
});
jScrollPane1.setViewportView(jTextPane1);

jMenuArchivo.setText("Archivo");

jMenuItemNuevo.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_N, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemNuevo.setBackground(new java.awt.Color(204, 204, 204));
jMenuItemNuevo.setText("Nuevo");
jMenuItemNuevo.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemNuevo.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
jMenuItemNuevo.setEnabled(false);
jMenuArchivo.add(jMenuItemNuevo);

jMenuItemAbrir.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_O, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemAbrir.setText("Abrir");
jMenuItemAbrir.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemAbrir.setEnabled(false);
jMenuArchivo.add(jMenuItemAbrir);

jMenuItemGuardar.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_G, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemGuardar.setText("Guardar");
jMenuItemGuardar.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemGuardar.setEnabled(false);
jMenuArchivo.add(jMenuItemGuardar);

jMenuItemGuardarComo.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_S,
java.awt.event.InputEvent.CTRL_MASK));
jMenuItemGuardarComo.setText("Guardar como ...");
jMenuItemGuardarComo.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemGuardarComo.setEnabled(false);
jMenuArchivo.add(jMenuItemGuardarComo);
```

```
jMenuItemEspecificarImpresora.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_P,
java.awt.event.InputEvent.CTRL_MASK));
jMenuItemEspecificarImpresora.setText("Especificar impresora ...");
jMenuItemEspecificarImpresora.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemEspecificarImpresora.setEnabled(false);
jMenuArchivo.add(jMenuItemEspecificarImpresora);

jMenuItemSalir.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_Q, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemSalir.setText("Salir");
jMenuItemSalir.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemSalir.setEnabled(false);
jMenuArchivo.add(jMenuItemSalir);

jMenuBarHyperTerminal.add(jMenuArchivo);

jMenuEdicion.setText("Edición");

jMenuItemCopiar.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_C, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemCopiar.setText("Copiar");
jMenuItemCopiar.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemCopiar.setEnabled(false);
jMenuEdicion.add(jMenuItemCopiar);

jMenuItemPegar.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_V, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemPegar.setText("Pegar");
jMenuItemPegar.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemPegar.setEnabled(false);
jMenuEdicion.add(jMenuItemPegar);
```

```
jMenuItemEnviar.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_INSERT,
java.awt.event.InputEvent.SHIFT_MASK | java.awt.event.InputEvent.CTRL_MASK));
jMenuItemEnviar.setText("Enviar");
jMenuItemEnviar.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemEnviar.setEnabled(false);
jMenuEdicion.add(jMenuItemEnviar);

jMenuItemSeleccionarTodo.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_E,
java.awt.event.InputEvent.CTRL_MASK));
jMenuItemSeleccionarTodo.setText("Seleccionar todo");
jMenuItemSeleccionarTodo.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemSeleccionarTodo.setEnabled(false);
jMenuEdicion.add(jMenuItemSeleccionarTodo);

jMenuItemLimpiarBuffer.setText("Limpiar búffer");
jMenuItemLimpiarBuffer.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemLimpiarBuffer.setEnabled(false);
jMenuEdicion.add(jMenuItemLimpiarBuffer);

jMenuBarHyperTerminal.add(jMenuEdicion);

jMenuConfiguracion.setText("Configuración");

jMenuItemNumTlfn.setText("Número de Teléfono");
jMenuItemNumTlfn.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemNumTlfn.setEnabled(false);
jMenuConfiguracion.add(jMenuItemNumTlfn);

jMenuItemEmulaTreminal.setText("Emulación de la Terminal");
jMenuItemEmulaTreminal.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemEmulaTreminal.setEnabled(false);
jMenuConfiguracion.add(jMenuItemEmulaTreminal);
```

```
jMenuItemPreferencias.setText("Preferencia de la Terminal");
jMenuItemPreferencias.setBorder(javax.swing.BorderFactory.createEtchedBorder());
jMenuItemPreferencias.setBorderPainted(true);
jMenuItemPreferencias.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jMenuItemPreferenciasMousePressed(evt);
    }
});
jMenuItemPreferencias.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItemPreferenciasActionPerformed(evt);
    }
});
jMenuConfiguracion.add(jMenuItemPreferencias);

jMenuItemTeclasFuncion.setText("Teclas de Función");
jMenuItemTeclasFuncion.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemTeclasFuncion.setEnabled(false);
jMenuConfiguracion.add(jMenuItemTeclasFuncion);

jMenuItemTransTexto.setText("Transferencias de Texto");
jMenuItemTransTexto.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemTransTexto.setEnabled(false);
jMenuItemTransTexto.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jMenuItemTransTextoMousePressed(evt);
    }
});
jMenuItemTransTexto.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItemTransTextoActionPerformed(evt);
    }
});
```

```
jMenuConfiguracion.add(jMenuItemTransTexto);

jMenuItemTransBinarias.setText("Transferencias Binarias");
jMenuItemTransBinarias.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemTransBinarias.setEnabled(false);
jMenuItemTransBinarias.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jMenuItemTransBinariasMousePressed(evt);
    }
});
jMenuConfiguracion.add(jMenuItemTransBinarias);

jMenuItemComunicaciones.setText("Comunicaciones");
jMenuItemComunicaciones.setBorder(javax.swing.BorderFactory.createEtchedBorder());
jMenuItemComunicaciones.setBorderPainted(true);
jMenuItemComunicaciones.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jMenuItemComunicacionesMousePressed(evt);
    }
});
jMenuItemComunicaciones.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItemComunicacionesActionPerformed(evt);
    }
});
jMenuConfiguracion.add(jMenuItemComunicaciones);

jMenuItemComModem.setText("Comandos del Modem");
jMenuItemComModem.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemComModem.setEnabled(false);
jMenuConfiguracion.add(jMenuItemComModem);
```

```
jMenuItemEcoImpresora.setText("Eco de la Impresora");
jMenuItemEcoImpresora.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemEcoImpresora.setEnabled(false);
jMenuConfiguracion.add(jMenuItemEcoImpresora);

jMenuItemModoCronometro.setText("Modo del Cronómetro");
jMenuItemModoCronometro.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemModoCronometro.setEnabled(false);
jMenuConfiguracion.add(jMenuItemModoCronometro);

jMenuItemMostrarTeclasFuncion.setText("Mostrar Teclas de Función");
jMenuItemMostrarTeclasFuncion.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemMostrarTeclasFuncion.setEnabled(false);
jMenuConfiguracion.add(jMenuItemMostrarTeclasFuncion);

jMenuBarHyperTerminal.add(jMenuConfiguracion);

jMenuTelefono.setText("Teléfono");

jMenuItemMarcar.setText("Marcar");
jMenuItemMarcar.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemMarcar.setEnabled(false);
jMenuTelefono.add(jMenuItemMarcar);

jMenuItemColgar.setText("Colgar");
jMenuItemColgar.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemColgar.setEnabled(false);
jMenuTelefono.add(jMenuItemColgar);

jMenuBarHyperTerminal.add(jMenuTelefono);

jMenuTransferencia.setText("Transferencia");
```

```
jMenuItemEnvArchTexto.setText("Enviar archivo de texto ...");
jMenuItemEnvArchTexto.setBorder(javax.swing.BorderFactory.createEtchedBorder());
jMenuItemEnvArchTexto.setBorderPainted(true);
jMenuItemEnvArchTexto.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jMenuItemEnvArchTextoMousePressed(evt);
    }
});
jMenuTransferencia.add(jMenuItemEnvArchTexto);

jMenuItemRecArchTexto.setText("Recibir archivos de texto ...");
jMenuItemRecArchTexto.setBorder(javax.swing.BorderFactory.createEtchedBorder());
jMenuItemRecArchTexto.setBorderPainted(true);
jMenuItemRecArchTexto.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jMenuItemRecArchTextoMousePressed(evt);
    }
});
jMenuTransferencia.add(jMenuItemRecArchTexto);

jMenuItemVerArchTexto.setText("Ver archivo de texto ...");
jMenuItemVerArchTexto.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemVerArchTexto.setEnabled(false);
jMenuTransferencia.add(jMenuItemVerArchTexto);

jMenuItemEnvArchBinario.setText("Enviar archivo binario ...");
jMenuItemEnvArchBinario.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemEnvArchBinario.setEnabled(false);
jMenuItemEnvArchBinario.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jMenuItemEnvArchBinarioMousePressed(evt);
    }
});
```

```
jMenuTransferencia.add(jMenuItemEnvArchBinario);

jMenuItemRecArchBinario.setText("Recibir archivo binario ...");
jMenuItemRecArchBinario.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemRecArchBinario.setEnabled(false);
jMenuItemRecArchBinario.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jMenuItemRecArchBinarioMousePressed(evt);
    }
});
jMenuTransferencia.add(jMenuItemRecArchBinario);

jMenuItemPausa.setText("Pausa");
jMenuItemPausa.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemPausa.setEnabled(false);
jMenuTransferencia.add(jMenuItemPausa);

jMenuItemReanudar.setText("Reanudar");
jMenuItemReanudar.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemReanudar.setEnabled(false);
jMenuTransferencia.add(jMenuItemReanudar);

jMenuItemDetener.setText("Detener");
jMenuItemDetener.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemDetener.setEnabled(false);
jMenuTransferencia.add(jMenuItemDetener);

jMenuBarHyperTerminal.add(jMenuTransferencia);

jMenuAyuda.setText("Ayuda");
```

```
jMenuItemIndice.setText("Indice");
jMenuItemIndice.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemIndice.setEnabled(false);
jMenuAyuda.add(jMenuItemIndice);

jMenuItemBuscarAyudaSobre.setText("Buscar ayuda sobre ...");
jMenuItemBuscarAyudaSobre.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemBuscarAyudaSobre.setEnabled(false);
jMenuAyuda.add(jMenuItemBuscarAyudaSobre);

jMenuItemUsoAyuda.setText("Uso de la ayuda ...");
jMenuItemUsoAyuda.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
jMenuItemUsoAyuda.setEnabled(false);
jMenuAyuda.add(jMenuItemUsoAyuda);

jMenuItemAcercaTerminal.setText("Acerca de terminal ...");
jMenuItemAcercaTerminal.setBorder(javax.swing.BorderFactory.createEtchedBorder());
jMenuItemAcercaTerminal.setBorderPainted(true);
jMenuAyuda.add(jMenuItemAcercaTerminal);

jMenuBarHyperTerminal.add(jMenuAyuda);

setJMenuBar(jMenuBarHyperTerminal);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
     .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE, 480,
     Short.MAX_VALUE)
);
```

```
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE, 277,
         Short.MAX_VALUE)
    );

    pack();
}// </editor-fold>

private void jMenuItemComunicacionesMousePressed(java.awt.event.MouseEvent evt) {
    Comunicaciones com=new Comunicaciones(null,true,this);
    com.setVisible(true);
}

private void jMenuItemPreferenciasMousePressed(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    PreferenciasTerminal pref=new PreferenciasTerminal(null,true);
    pref.inicializar(this);
    pref.setVisible(true);
    this.ecoLocal=pref.ecoLocal;
    if(ecoLocal){
        jTextPane1.setEditable(true);
    }
    else{
        jTextPane1.setEditable(false);
    }
}

private void jTextPane1KeyPressed(java.awt.event.KeyEvent evt) {
    uart.enviar(evt.getKeyChar());
}
```

```
private void jMenuItemEnvArchTextoMousePressed(java.awt.event.MouseEvent evt) {
    mostrarEnviarArchivosTexto();
}

private void jMenuItemRecArchTextoMousePressed(java.awt.event.MouseEvent evt) {
   mostrarRecibirArchivosTexto();
}

// Variables declaration - do not modify
private javax.swing.JMenu jMenuArchivo;
private javax.swing.JMenu jMenuAyuda;
private javax.swing.JMenuBar jMenuBarHyperTerminal;
private javax.swing.JMenu jMenuConfiguracion;
private javax.swing.JMenu jMenuEdicion;
private javax.swing.JMenuItem jMenuItemAbrir;
private javax.swing.JMenuItem jMenuItemAcercaTerminal;
private javax.swing.JMenuItem jMenuItemBuscarAyudaSobre;
private javax.swing.JMenuItem jMenuItemColgar;
private javax.swing.JMenuItem jMenuItemComModem;
private javax.swing.JMenuItem jMenuItemComunicaciones;
private javax.swing.JMenuItem jMenuItemCopiar;
private javax.swing.JMenuItem jMenuItemDetener;
private javax.swing.JMenuItem jMenuItemEcoImpresora;
private javax.swing.JMenuItem jMenuItemEmulaTreminal;
private javax.swing.JMenuItem jMenuItemEnvArchBinario;
private javax.swing.JMenuItem jMenuItemEnvArchTexto;
private javax.swing.JMenuItem jMenuItemEnviar;
private javax.swing.JMenuItem jMenuItemEspecificarImpresora;
private javax.swing.JMenuItem jMenuItemGuardar;
private javax.swing.JMenuItem jMenuItemGuardarComo;
private javax.swing.JMenuItem jMenuItemIndice;
private javax.swing.JMenuItem jMenuItemLimpiarBuffer;
private javax.swing.JMenuItem jMenuItemMarcar;
```

```
    private javax.swing.JMenuItem jMenuItemModoCronometro;
    private javax.swing.JMenuItem jMenuItemMostrarTeclasFuncion;
    private javax.swing.JMenuItem jMenuItemNuevo;
    private javax.swing.JMenuItem jMenuItemNumTlfn;
    private javax.swing.JMenuItem jMenuItemPausa;
    private javax.swing.JMenuItem jMenuItemPegar;
    private javax.swing.JMenuItem jMenuItemPreferencias;
    private javax.swing.JMenuItem jMenuItemReanudar;
    private javax.swing.JMenuItem jMenuItemRecArchBinario;
    private javax.swing.JMenuItem jMenuItemRecArchTexto;
    private javax.swing.JMenuItem jMenuItemSalir;
    private javax.swing.JMenuItem jMenuItemSeleccionarTodo;
    private javax.swing.JMenuItem jMenuItemTeclasFuncion;
    private javax.swing.JMenuItem jMenuItemTransBinarias;
    private javax.swing.JMenuItem jMenuItemTransTexto;
    private javax.swing.JMenuItem jMenuItemUsoAyuda;
    private javax.swing.JMenuItem jMenuItemVerArchTexto;
    private javax.swing.JMenu jMenuTelefono;
    private javax.swing.JMenu jMenuTransferencia;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextPane jTextPane1;
    // End of variables declaration
}
```

B.4 Comunicaciones.

```
package rs232;

public class Comunicaciones extends javax.swing.JDialog {
    HyperTerminal hyperTerminal;
    private int velocidad;
    private int bitsDatos;
    private double bitsParo;
    private int paridad;

    private boolean ctrl_flujo;
    /** Creates new form Comunicaciones **/
    public Comunicaciones(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }

    public Comunicaciones(java.awt.Frame parent, boolean modal,HyperTerminal hyperTerminal){
        super(parent,modal);
        initComponents();
        this.hyperTerminal=hyperTerminal;
        velocidad=hyperTerminal.getVelocidad();
        bitsParo=hyperTerminal.getBitsParo();
        bitsDatos=hyperTerminal.getBitsDatos();
        paridad=hyperTerminal.getParidad();
        ctrl_flujo=hyperTerminal.getCtrl_Flujo();

        if(ctrl_flujo==true){
            jRadioButtonX_on_off.setSelected(true);
        }

        if(ctrl_flujo==false){
            jRadioButtonNinguno.setSelected(true);
        }
```

```
switch(velocidad){
    case 110:
        jRadioButton110.setSelected(true);
        break;
    case 300:
        jRadioButton300.setSelected(true);
        break;
    case 600:
        jRadioButton600.setSelected(true);
        break;
    case 1200:
        jRadioButton1200.setSelected(true);
        break;
    case 2400:
        jRadioButton2400.setSelected(true);
        break;
    case 4800:
        jRadioButton4800.setSelected(true);
        break;
    case 9600:
        jRadioButton9600.setSelected(true);
        break;
    case 19200:
        jRadioButton19200.setSelected(true);
        break;
}
switch(bitsDatos){
    case 5:
        jRadioButton5.setSelected(true);
        break;
    case 6:
        jRadioButton6.setSelected(true);
        break;
    case 7:
        jRadioButton7.setSelected(true);
        break;
    case 8:
        jRadioButton8.setSelected(true);
        break;
}
```

```
        if(bitsParo==1.0){
            jRadioButtonBitsParo1.setSelected(true);
        }
        if(bitsParo==1.5){
            jRadioButtonBitsParo15.setSelected(true);
        }
        if(bitsParo==2.0){
            jRadioButtonBitsParo2.setSelected(true);
        }
        if(paridad==UART.PARIDAD_NINGUNA){
            jRadioButtonNinguna.setSelected(true);
        }
        if(paridad==UART.PARIDAD_ESPACIO){
            jRadioButtonEspacio.setSelected(true);
        }
        if(paridad==UART.PARIDAD_IMPAR){
            jRadioButtonImpar.setSelected(true);
        }
        if(paridad==UART.PARIDAD_MARCA){
            jRadioButtonMarca.setSelected(true);
        }
        if(paridad==UART.PARIDAD_PAR){
            jRadioButtonPar.setSelected(true);
        }

    }

    @SuppressWarnings("unchecked")

    private void initComponents() {

        buttonGroupVelocidadTx = new javax.swing.ButtonGroup();
        buttonGroupBitsDatos = new javax.swing.ButtonGroup();
        buttonGroupBitsParo = new javax.swing.ButtonGroup();
        buttonGroupParidad = new javax.swing.ButtonGroup();
        buttonGroupControlFlujo = new javax.swing.ButtonGroup();
        jPanelVelocTransfBaudios = new javax.swing.JPanel();
        jRadioButton110 = new javax.swing.JRadioButton();
        jRadioButton300 = new javax.swing.JRadioButton();
        jRadioButton600 = new javax.swing.JRadioButton();
```

```
jRadioButton1200 = new javax.swing.JRadioButton();
jRadioButton2400 = new javax.swing.JRadioButton();
jRadioButton4800 = new javax.swing.JRadioButton();
jRadioButton9600 = new javax.swing.JRadioButton();
jRadioButton19200 = new javax.swing.JRadioButton();
jPanelBotones = new javax.swing.JPanel();
jButtonAceptar = new javax.swing.JButton();
jButtonCancelar = new javax.swing.JButton();
jPanelBitDatos = new javax.swing.JPanel();
jRadioButton5 = new javax.swing.JRadioButton();
jRadioButton6 = new javax.swing.JRadioButton();
jRadioButton7 = new javax.swing.JRadioButton();
jRadioButton8 = new javax.swing.JRadioButton();
jPanelBitParo = new javax.swing.JPanel();
jRadioButtonBitsParo1 = new javax.swing.JRadioButton();
jRadioButtonBitsParo15 = new javax.swing.JRadioButton();
jRadioButtonBitsParo2 = new javax.swing.JRadioButton();
jPanelParidad = new javax.swing.JPanel();
jRadioButtonNinguna = new javax.swing.JRadioButton();
jRadioButtonPar = new javax.swing.JRadioButton();
jRadioButtonImpar = new javax.swing.JRadioButton();
jRadioButtonMarca = new javax.swing.JRadioButton();
jRadioButtonEspacio = new javax.swing.JRadioButton();
jPanelControlFlujo = new javax.swing.JPanel();
jRadioButtonX_on_off = new javax.swing.JRadioButton();
jRadioButtonHardware = new javax.swing.JRadioButton();
jRadioButtonNinguno = new javax.swing.JRadioButton();
jPanelConector = new javax.swing.JPanel();
jScrollPaneConector = new javax.swing.JScrollPane();
jListConector = new javax.swing.JList();
jPanelVerifiDetecc = new javax.swing.JPanel();
jCheckBoxVerificacion = new javax.swing.JCheckBox();
jCheckBoxDeteccion = new javax.swing.JCheckBox();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("Comunicaciones");

jPanelVelocTransfBaudios.setBorder(javax.swing.BorderFactory.createTitledBorder("Velocidad de transmision en baudios"));
jPanelVelocTransfBaudios.setLayout(new java.awt.GridLayout(2, 4));
```

```
buttonGroupVelocidadTx.add(jRadioButton110);
jRadioButton110.setText("110");
jRadioButton110.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton110MousePressed(evt);
    }
});
jPanelVelocTransfBaudios.add(jRadioButton110);

buttonGroupVelocidadTx.add(jRadioButton300);
jRadioButton300.setText("300");
jRadioButton300.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton300MousePressed(evt);
    }
});
jPanelVelocTransfBaudios.add(jRadioButton300);

buttonGroupVelocidadTx.add(jRadioButton600);
jRadioButton600.setText("600");
jRadioButton600.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton600MousePressed(evt);
    }
});
jPanelVelocTransfBaudios.add(jRadioButton600);

buttonGroupVelocidadTx.add(jRadioButton1200);
jRadioButton1200.setText("1200");
jRadioButton1200.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton1200MousePressed(evt);
    }
});
jPanelVelocTransfBaudios.add(jRadioButton1200);
```

```
buttonGroupVelocidadTx.add(jRadioButton2400);
jRadioButton2400.setText("2400");
jRadioButton2400.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton2400MousePressed(evt);
    }
});
jPanelVelocTransfBaudios.add(jRadioButton2400);

buttonGroupVelocidadTx.add(jRadioButton4800);
jRadioButton4800.setText("4800");
jRadioButton4800.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton4800MousePressed(evt);
    }
});
jPanelVelocTransfBaudios.add(jRadioButton4800);

buttonGroupVelocidadTx.add(jRadioButton9600);
jRadioButton9600.setSelected(true);
jRadioButton9600.setText("9600");
jRadioButton9600.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton9600MousePressed(evt);
    }
});
jPanelVelocTransfBaudios.add(jRadioButton9600);

buttonGroupVelocidadTx.add(jRadioButton19200);
jRadioButton19200.setText("19200");
jRadioButton19200.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton19200MousePressed(evt);
    }
});
jPanelVelocTransfBaudios.add(jRadioButton19200);

jPanelBotones.setLayout(new java.awt.GridLayout(2, 1));
```

```
jButtonAceptar.setText("Aceptar");
jButtonAceptar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButtonAceptarMouseClicked(evt);
    }
});
jPanelBotones.add(jButtonAceptar);

jButtonCancelar.setText("Cancelar");
jButtonCancelar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButtonCancelarMouseClicked(evt);
    }
});
jPanelBotones.add(jButtonCancelar);

jPanelBitDatos.setBorder(javax.swing.BorderFactory.createTitledBorder("Bits de datos"));
jPanelBitDatos.setLayout(new java.awt.GridLayout(1, 4));

buttonGroupBitsDatos.add(jRadioButton5);
jRadioButton5.setText("5");
jRadioButton5.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton5MousePressed(evt);
    }
});
jPanelBitDatos.add(jRadioButton5);

buttonGroupBitsDatos.add(jRadioButton6);
jRadioButton6.setText("6");
jRadioButton6.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton6MousePressed(evt);
    }
});
jPanelBitDatos.add(jRadioButton6);
```

```
buttonGroupBitsDatos.add(jRadioButton7);
jRadioButton7.setText("7");
jRadioButton7.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton7MousePressed(evt);
    }
});
jPanelBitDatos.add(jRadioButton7);

buttonGroupBitsDatos.add(jRadioButton8);
jRadioButton8.setSelected(true);
jRadioButton8.setText("8");
jRadioButton8.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButton8MousePressed(evt);
    }
});
jPanelBitDatos.add(jRadioButton8);

jPanelBitParo.setBorder(javax.swing.BorderFactory.createTitledBorder("Bits de paro"));
jPanelBitParo.setLayout(new java.awt.GridLayout(1, 0));

buttonGroupBitsParo.add(jRadioButtonBitsParo1);
jRadioButtonBitsParo1.setSelected(true);
jRadioButtonBitsParo1.setText("1");
jRadioButtonBitsParo1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonBitsParo1MousePressed(evt);
    }
});
jPanelBitParo.add(jRadioButtonBitsParo1);

buttonGroupBitsParo.add(jRadioButtonBitsParo15);
jRadioButtonBitsParo15.setText("1.5");
jRadioButtonBitsParo15.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonBitsParo15MousePressed(evt);
    }
});
```

```
jPanelBitParo.add(jRadioButtonBitsParo15);

buttonGroupBitsParo.add(jRadioButtonBitsParo2);
jRadioButtonBitsParo2.setText("2");
jRadioButtonBitsParo2.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonBitsParo2MousePressed(evt);
    }
});
jPanelBitParo.add(jRadioButtonBitsParo2);

jPanelParidad.setBorder(javax.swing.BorderFactory.createTitledBorder("Paridad"));
jPanelParidad.setLayout(new java.awt.GridLayout(5, 1));

buttonGroupParidad.add(jRadioButtonNinguna);
jRadioButtonNinguna.setSelected(true);
jRadioButtonNinguna.setText("Ninguna");
jRadioButtonNinguna.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonNingunaMousePressed(evt);
    }
});
jPanelParidad.add(jRadioButtonNinguna);

buttonGroupParidad.add(jRadioButtonPar);
jRadioButtonPar.setText("Par");
jRadioButtonPar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonParMousePressed(evt);
    }
});
jPanelParidad.add(jRadioButtonPar);

buttonGroupParidad.add(jRadioButtonImpar);
jRadioButtonImpar.setText("Impar");
jRadioButtonImpar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonImparMousePressed(evt);
    }
});
```

```
jPanelParidad.add(jRadioButtonImpar);

buttonGroupParidad.add(jRadioButtonMarca);
jRadioButtonMarca.setText("Marca");
jRadioButtonMarca.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonMarcaMousePressed(evt);
    }
});
jPanelParidad.add(jRadioButtonMarca);

buttonGroupParidad.add(jRadioButtonEspacio);
jRadioButtonEspacio.setText("Espacio");
jRadioButtonEspacio.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonEspacioMousePressed(evt);
    }
});
jPanelParidad.add(jRadioButtonEspacio);

jPanelControlFlujo.setBorder(javax.swing.BorderFactory.createTitledBorder("Control de Flujo"));
jPanelControlFlujo.setLayout(new java.awt.GridLayout(3, 1));

buttonGroupControlFlujo.add(jRadioButtonX_on_off);
jRadioButtonX_on_off.setText("Xon/Xoff");
jRadioButtonX_on_off.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonX_on_offMousePressed(evt);
    }
});
jPanelControlFlujo.add(jRadioButtonX_on_off);

buttonGroupControlFlujo.add(jRadioButtonHardware);
jRadioButtonHardware.setText("Hardware");
jPanelControlFlujo.add(jRadioButtonHardware);
```

```
buttonGroupControlFlujo.add(jRadioButtonNinguno);
jRadioButtonNinguno.setSelected(true);
jRadioButtonNinguno.setText("Ninguno");
jRadioButtonNinguno.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jRadioButtonNingunoMousePressed(evt);
    }
});
jPanelControlFlujo.add(jRadioButtonNinguno);

jPanelConector.setBorder(javax.swing.BorderFactory.createTitledBorder("Conector"));
jPanelConector.setLayout(new java.awt.GridLayout(1, 1));

jListConector.setModel(new javax.swing.AbstractListModel() {
    String[] strings = { "Ninguno", "COM1", "COM2" };
    public int getSize() { return strings.length; }
    public Object getElementAt(int i) { return strings[i]; }
});
jListConector.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
jScrollPaneConector.setViewportView(jListConector);

jPanelConector.add(jScrollPaneConector);

jPanelVerifiDetecc.setLayout(new java.awt.GridLayout(2, 1));

jCheckBoxVerificacion.setText("Verificación de paridad");
jPanelVerifiDetecc.add(jCheckBoxVerificacion);

jCheckBoxDeteccion.setText("Detección de portadora");
jPanelVerifiDetecc.add(jCheckBoxDeteccion);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                    .addGroup(layout.createSequentialGroup()
```

```
                                .addGap(10, 10, 10)
                                .addComponent(jPanelVelocTransfBaudios, javax.swing.GroupLayout.PREFERRED_SIZE, 294,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addComponent(jPanelBitDatos, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(jPanelBitParo, javax.swing.GroupLayout.PREFERRED_SIZE, 178,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jPanelBotones, javax.swing.GroupLayout.PREFERRED_SIZE, 126,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jPanelParidad, javax.swing.GroupLayout.PREFERRED_SIZE, 121,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(18, 18, 18)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
                                .addComponent(jPanelControlFlujo, javax.swing.GroupLayout.PREFERRED_SIZE, 132,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addGap(18, 18, 18)
                                .addComponent(jPanelConector, javax.swing.GroupLayout.DEFAULT_SIZE, 193, Short.MAX_VALUE))
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jPanelVerifiDetecc, javax.swing.GroupLayout.PREFERRED_SIZE, 153,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addGap(190, 190, 190)))))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jPanelVelocTransfBaudios, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jPanelBotones, javax.swing.GroupLayout.PREFERRED_SIZE, 63,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanelBitDatos, javax.swing.GroupLayout.PREFERRED_SIZE, 65,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                    .addComponent(jPanelBitParo, javax.swing.GroupLayout.DEFAULT_SIZE, 67, Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanelParidad, javax.swing.GroupLayout.PREFERRED_SIZE, 153,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jPanelConector, javax.swing.GroupLayout.DEFAULT_SIZE, 113, Short.MAX_VALUE)
                            .addComponent(jPanelControlFlujo, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 113, Short.MAX_VALUE))
                        .addGap(11, 11, 11)
                        .addComponent(jPanelVerifiDetecc, javax.swing.GroupLayout.PREFERRED_SIZE, 59,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void jRadioButton110MousePressed(java.awt.event.MouseEvent evt) {
        setVelocidad(110);
    }

    private void jRadioButton300MousePressed(java.awt.event.MouseEvent evt) {
        setVelocidad(300);
    }

    private void jRadioButton600MousePressed(java.awt.event.MouseEvent evt) {
        setVelocidad(600);
    }

    private void jRadioButton1200MousePressed(java.awt.event.MouseEvent evt) {
        setVelocidad(1200);
    }

    private void jRadioButton2400MousePressed(java.awt.event.MouseEvent evt) {
        setVelocidad(2400);
    }

    private void jRadioButton4800MousePressed(java.awt.event.MouseEvent evt) {
        setVelocidad(4800);
```

```
}

private void jRadioButton9600MousePressed(java.awt.event.MouseEvent evt) {
    setVelocidad(9600);
}

private void jRadioButton19200MousePressed(java.awt.event.MouseEvent evt) {
    setVelocidad(19200);
}

private void jRadioButton5MousePressed(java.awt.event.MouseEvent evt) {
    setBitsDatos(5);
}

private void jRadioButton6MousePressed(java.awt.event.MouseEvent evt) {
    setBitsDatos(6);
}

private void jRadioButton7MousePressed(java.awt.event.MouseEvent evt) {
    setBitsDatos(7);
}

private void jRadioButton8MousePressed(java.awt.event.MouseEvent evt) {
    setBitsDatos(8);
}

private void jRadioButtonBitsParo1MousePressed(java.awt.event.MouseEvent evt) {
    setBitsParo(1.0);
}

private void jRadioButtonBitsParo15MousePressed(java.awt.event.MouseEvent evt) {
    setBitsParo(1.5);
}

private void jRadioButtonBitsParo2MousePressed(java.awt.event.MouseEvent evt) {
    setBitsParo(2.0);
}

private void jRadioButtonNingunaMousePressed(java.awt.event.MouseEvent evt) {
    setParidad(UART.PARIDAD_NINGUNA);
}
```

```
private void jRadioButtonParMousePressed(java.awt.event.MouseEvent evt) {
    setParidad(UART.PARIDAD_PAR);
}

private void jRadioButtonImparMousePressed(java.awt.event.MouseEvent evt) {
    setParidad(UART.PARIDAD_IMPAR);
}

private void jRadioButtonMarcaMousePressed(java.awt.event.MouseEvent evt) {
    setParidad(UART.PARIDAD_MARCA);
}

private void jRadioButtonEspacioMousePressed(java.awt.event.MouseEvent evt) {
    setParidad(UART.PARIDAD_ESPACIO);
}

private void jButtonAceptarMouseClicked(java.awt.event.MouseEvent evt) {
    hacerEfectivo();
    setVisible(false);
}

private void jButtonCancelarMouseClicked(java.awt.event.MouseEvent evt) {
    setVisible(false);
}

private void jRadioButtonX_on_offMousePressed(java.awt.event.MouseEvent evt) {
    setCtrl_Flujo(true);
}

private void jRadioButtonNingunoMousePressed(java.awt.event.MouseEvent evt) {
    setCtrl_Flujo(false);
}

public static void main(String args[]) {
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
```

```
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(Comunicaciones.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(Comunicaciones.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(Comunicaciones.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(Comunicaciones.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the dialog */
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                Comunicaciones dialog = new Comunicaciones(new javax.swing.JFrame(), true);
                dialog.addWindowListener(new java.awt.event.WindowAdapter() {

                    @Override
                    public void windowClosing(java.awt.event.WindowEvent e) {
                        System.exit(0);
                    }
                });
                dialog.setVisible(true);
            }
        });
    }

    public void setVelocidad(int v){
        velocidad=v;
    }

    public void setBitsDatos(int bitsDatos){
        this.bitsDatos=bitsDatos;
    }

    public void setBitsParo(double bitsParo){
        this.bitsParo=bitsParo;
    }
```

```
public void setParidad(int p){
    this.paridad=p;
}

    public void setCtrl_Flujo(boolean cf){
    this.ctrl_flujo=cf;
}

public void hacerEfectivo(){
    hyperTerminal.setVelocidad(velocidad);
    hyperTerminal.setBitsDatos(bitsDatos);
    hyperTerminal.setBitsParo(bitsParo);
    hyperTerminal.setParidad(paridad);
    hyperTerminal.setCtrl_Flujo(ctrl_flujo);
}

// Variables declaration - do not modify
private javax.swing.ButtonGroup buttonGroupBitsDatos;
private javax.swing.ButtonGroup buttonGroupBitsParo;
private javax.swing.ButtonGroup buttonGroupControlFlujo;
private javax.swing.ButtonGroup buttonGroupParidad;
private javax.swing.ButtonGroup buttonGroupVelocidadTx;
private javax.swing.JButton jButtonAceptar;
private javax.swing.JButton jButtonCancelar;
private javax.swing.JCheckBox jCheckBoxDeteccion;
private javax.swing.JCheckBox jCheckBoxVerificacion;
private javax.swing.JList jListConector;
private javax.swing.JPanel jPanelBitDatos;
private javax.swing.JPanel jPanelBitParo;
private javax.swing.JPanel jPanelBotones;
private javax.swing.JPanel jPanelConector;
private javax.swing.JPanel jPanelControlFlujo;
private javax.swing.JPanel jPanelParidad;
private javax.swing.JPanel jPanelVelocTransfBaudios;
private javax.swing.JPanel jPanelVerifiDetecc;
private javax.swing.JRadioButton jRadioButton110;
private javax.swing.JRadioButton jRadioButton1200;
private javax.swing.JRadioButton jRadioButton19200;
private javax.swing.JRadioButton jRadioButton2400;
private javax.swing.JRadioButton jRadioButton300;
```

```
    private javax.swing.JRadioButton jRadioButton4800;
    private javax.swing.JRadioButton jRadioButton5;
    private javax.swing.JRadioButton jRadioButton6;
    private javax.swing.JRadioButton jRadioButton600;
    private javax.swing.JRadioButton jRadioButton7;
    private javax.swing.JRadioButton jRadioButton8;
    private javax.swing.JRadioButton jRadioButton9600;
    private javax.swing.JRadioButton jRadioButtonBitsParo1;
    private javax.swing.JRadioButton jRadioButtonBitsParo15;
    private javax.swing.JRadioButton jRadioButtonBitsParo2;
    private javax.swing.JRadioButton jRadioButtonEspacio;
    private javax.swing.JRadioButton jRadioButtonHardware;
    private javax.swing.JRadioButton jRadioButtonImpar;
    private javax.swing.JRadioButton jRadioButtonMarca;
    private javax.swing.JRadioButton jRadioButtonNinguna;
    private javax.swing.JRadioButton jRadioButtonNinguno;
    private javax.swing.JRadioButton jRadioButtonPar;
    private javax.swing.JRadioButton jRadioButtonX_on_off;
    private javax.swing.JScrollPane jScrollPaneConector;
    // End of variables declaration
}
```

B.5 Preferencial del Terminal.

```
package rs232;

public class PreferenciasTerminal extends javax.swing.JDialog {
    HyperTerminal hyperTerminal;
    boolean ecoLocal=false;
    boolean ecoLocalInicio=false;
    /** Creates new form PreferenciasTerminal */
    public PreferenciasTerminal(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }

    public void inicializar(HyperTerminal hyperTerminal){
        this.hyperTerminal=hyperTerminal;
        ecoLocal=hyperTerminal.ecoLocal;
        ecoLocalInicio=ecoLocal;
        if(ecoLocal){
            jCheckBoxEcoLocal.setSelected(true);
        }
    }

    @SuppressWarnings("unchecked")

    private void initComponents() {

        buttonGroupColumnas = new javax.swing.ButtonGroup();
        buttonGroupCursor = new javax.swing.ButtonGroup();
        jPanelModosTerminal = new javax.swing.JPanel();
        jCheckBoxAjusteLinea = new javax.swing.JCheckBox();
        jCheckBoxEcoLocal = new javax.swing.JCheckBox();
        jCheckBoxSonido = new javax.swing.JCheckBox();
        jPanelRC = new javax.swing.JPanel();
        jCheckBoxEntrada = new javax.swing.JCheckBox();
        jCheckBoxSalida = new javax.swing.JCheckBox();
        jPanelBotones = new javax.swing.JPanel();
```

```
jButtonAceptar = new javax.swing.JButton();
jButtonCancelar = new javax.swing.JButton();
jPanelColumnas = new javax.swing.JPanel();
jRadioButton80 = new javax.swing.JRadioButton();
jRadioButton132 = new javax.swing.JRadioButton();
jPanelCursor = new javax.swing.JPanel();
jRadioButtonBloque = new javax.swing.JRadioButton();
jRadioButtonLinea = new javax.swing.JRadioButton();
jCheckBoxIntermitente = new javax.swing.JCheckBox();
jPanelFuenteTerminal = new javax.swing.JPanel();
jScrollPaneFuente = new javax.swing.JScrollPane();
jListFuentes = new javax.swing.JList();
jTextFieldTamano = new javax.swing.JTextField();
jPanelTraducciones = new javax.swing.JPanel();
jScrollPanePais = new javax.swing.JScrollPane();
jListPais = new javax.swing.JList();
jCheckBoxIBM_ANSI = new javax.swing.JCheckBox();
jPanelopciones = new javax.swing.JPanel();
jCheckBoxMostrarBarras = new javax.swing.JCheckBox();
jCheckBoxUsarTeclas = new javax.swing.JCheckBox();
jPanelOpciones1 = new javax.swing.JPanel();
jLabelLineas = new javax.swing.JLabel();
jTextFieldlineas100 = new javax.swing.JTextField();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("Preferencias de la terminal");

jPanelModosTerminal.setBorder(javax.swing.BorderFactory.createTitledBorder("Modos de la terminal"));
jPanelModosTerminal.setLayout(new java.awt.GridLayout(3, 1));

jCheckBoxAjusteLinea.setSelected(true);
jCheckBoxAjusteLinea.setText("Ajuste de línea");
jPanelModosTerminal.add(jCheckBoxAjusteLinea);

jCheckBoxEcoLocal.setText("Eco local");
jCheckBoxEcoLocal.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        jCheckBoxEcoLocalStateChanged(evt);
    }
});
jPanelModosTerminal.add(jCheckBoxEcoLocal);
```

```
jCheckBoxSonido.setSelected(true);
jCheckBoxSonido.setText("Sonido");
jPanelModosTerminal.add(jCheckBoxSonido);

jPanelRC.setBorder(javax.swing.BorderFactory.createTitledBorder("RC -> RC/AL"));
jPanelRC.setLayout(new java.awt.GridLayout(2, 1));

jCheckBoxEntrada.setText("De entrada");
jPanelRC.add(jCheckBoxEntrada);

jCheckBoxSalida.setText("De salida");
jPanelRC.add(jCheckBoxSalida);

jPanelBotones.setLayout(new java.awt.GridLayout(2, 1));

jButtonAceptar.setText("Aceptar");
jButtonAceptar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButtonAceptarMouseClicked(evt);
    }
});
jPanelBotones.add(jButtonAceptar);

jButtonCancelar.setText("Cancelar");
jButtonCancelar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButtonCancelarMouseClicked(evt);
    }
});
jPanelBotones.add(jButtonCancelar);

jPanelColumnas.setBorder(javax.swing.BorderFactory.createTitledBorder("Columnas"));
jPanelColumnas.setLayout(new java.awt.GridLayout(1, 2));

buttonGroupColumnas.add(jRadioButton80);
jRadioButton80.setSelected(true);
jRadioButton80.setText("80");
jPanelColumnas.add(jRadioButton80);

buttonGroupColumnas.add(jRadioButton132);
```

```
jRadioButton132.setText("132");
jPanelColumnas.add(jRadioButton132);

jPanelCursor.setBorder(javax.swing.BorderFactory.createTitledBorder("Cursor"));
jPanelCursor.setLayout(new java.awt.GridLayout(2, 2));

buttonGroupCursor.add(jRadioButtonBloque);
jRadioButtonBloque.setSelected(true);
jRadioButtonBloque.setText("Bloque");
jPanelCursor.add(jRadioButtonBloque);

buttonGroupCursor.add(jRadioButtonLinea);
jRadioButtonLinea.setText("Línea");
jPanelCursor.add(jRadioButtonLinea);

jCheckBoxIntermitente.setSelected(true);
jCheckBoxIntermitente.setText("Intermitente");
jPanelCursor.add(jCheckBoxIntermitente);

jPanelFuenteTerminal.setBorder(javax.swing.BorderFactory.createTitledBorder("Fuente de terminal"));
jPanelFuenteTerminal.setLayout(new java.awt.GridLayout(1, 2));

jListFuentes.setModel(new javax.swing.AbstractListModel() {
    String[] strings = { "Courier New Ground", "Courier New Trouble", "Fixedsys", "Times New Roman" };
    public int getSize() { return strings.length; }
    public Object getElementAt(int i) { return strings[i]; }
});
jListFuentes.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
jListFuentes.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
jScrollPaneFuente.setViewportView(jListFuentes);

jPanelFuenteTerminal.add(jScrollPaneFuente);

jTextFieldTamano.setEditable(false);
jTextFieldTamano.setText("15");
jTextFieldTamano.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextFieldTamanoActionPerformed(evt);
    }
});
jPanelFuenteTerminal.add(jTextFieldTamano);
```

```
jPanelTraducciones.setBorder(javax.swing.BorderFactory.createTitledBorder("Traducciones"));
jPanelTraducciones.setLayout(new java.awt.GridLayout(2, 1));

jListPais.setModel(new javax.swing.AbstractListModel() {
    String[] strings = { "Alemania", "España", "Italia", "Inglaterra", "Francia", "Ecuador", "Colombia", "Suiza" };
    public int getSize() { return strings.length; }
    public Object getElementAt(int i) { return strings[i]; }
});
jListPais.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
jScrollPanePais.setViewportView(jListPais);

jPanelTraducciones.add(jScrollPanePais);

jCheckBoxIBM_ANSI.setText("De IBM a ANSI");
jPanelTraducciones.add(jCheckBoxIBM_ANSI);

jPanelopciones.setLayout(new java.awt.GridLayout(2, 1));

jCheckBoxMostrarBarras.setSelected(true);
jCheckBoxMostrarBarras.setText("Mostrar barras de desplazamiento");
jPanelopciones.add(jCheckBoxMostrarBarras);

jCheckBoxUsarTeclas.setSelected(true);
jCheckBoxUsarTeclas.setText("Usar teclas de función, detección y Ctrl para windows");
jPanelopciones.add(jCheckBoxUsarTeclas);

jPanelOpciones1.setLayout(new java.awt.GridLayout(1, 2));

jLabelLineas.setText("Lineas de búffer");
jPanelOpciones1.add(jLabelLineas);

jTextFieldlineas100.setEditable(false);
jTextFieldlineas100.setText("100");
jTextFieldlineas100.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextFieldlineas100ActionPerformed(evt);
    }
});
jPanelOpciones1.add(jTextFieldlineas100);
```

```
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jPanelColumnas, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jPanelModosTerminal, javax.swing.GroupLayout.PREFERRED_SIZE, 135,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                            .addComponent(jPanelRC, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                            .addComponent(jPanelCursor, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                        .addGap(18, 18, 18)
                        .addComponent(jPanelBotones, javax.swing.GroupLayout.DEFAULT_SIZE, 164, Short.MAX_VALUE))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jPanelFuenteTerminal, javax.swing.GroupLayout.PREFERRED_SIZE, 199,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(112, 112, 112)
                        .addComponent(jPanelTraducciones, javax.swing.GroupLayout.DEFAULT_SIZE, 250, Short.MAX_VALUE))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jPanelopciones, javax.swing.GroupLayout.PREFERRED_SIZE, 302,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(18, 18, 18)
                        .addComponent(jPanelOpciones1, javax.swing.GroupLayout.PREFERRED_SIZE, 185,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
```

```
                        .addComponent(jPanelModosTerminal, javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jPanelColumnas, javax.swing.GroupLayout.PREFERRED_SIZE, 59,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(jPanelBotones, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 71, Short.MAX_VALUE)
                            .addComponent(jPanelRC, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.PREFERRED_SIZE, 71, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(jPanelCursor, javax.swing.GroupLayout.DEFAULT_SIZE, 83, Short.MAX_VALUE)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanelTraducciones, javax.swing.GroupLayout.PREFERRED_SIZE, 132,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jPanelFuenteTerminal, javax.swing.GroupLayout.PREFERRED_SIZE, 110,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(8, 8, 8)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanelopciones, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(15, 15, 15)
                        .addComponent(jPanelOpciones1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void jCheckBoxEcoLocalStateChanged(javax.swing.event.ChangeEvent evt) {
        ecoLocal=this.jCheckBoxEcoLocal.isSelected();
    }

    private void jButtonAceptarMouseClicked(java.awt.event.MouseEvent evt) {
        this.setVisible(false);
    }
```

```
    private void jButtonCancelarMouseClicked(java.awt.event.MouseEvent evt) {
        ecoLocal=ecoLocalInicio;
        this.setVisible(false);
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(PreferenciasTerminal.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(PreferenciasTerminal.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(PreferenciasTerminal.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(PreferenciasTerminal.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        }
        //</editor-fold>

        /* Create and display the dialog */
```

```
    java.awt.EventQueue.invokeLater(new Runnable() {

        public void run() {
            PreferenciasTerminal dialog = new PreferenciasTerminal(new javax.swing.JFrame(), true);
            dialog.addWindowListener(new java.awt.event.WindowAdapter() {

                @Override
                public void windowClosing(java.awt.event.WindowEvent e) {
                    System.exit(0);
                }
            });
            dialog.setVisible(true);
        }
    });
}
// Variables declaration - do not modify
private javax.swing.ButtonGroup buttonGroupColumnas;
private javax.swing.ButtonGroup buttonGroupCursor;
private javax.swing.JButton jButtonAceptar;
private javax.swing.JButton jButtonCancelar;
private javax.swing.JCheckBox jCheckBoxAjusteLinea;
private javax.swing.JCheckBox jCheckBoxEcoLocal;
private javax.swing.JCheckBox jCheckBoxEntrada;
private javax.swing.JCheckBox jCheckBoxIBM_ANSI;
private javax.swing.JCheckBox jCheckBoxIntermitente;
private javax.swing.JCheckBox jCheckBoxMostrarBarras;
private javax.swing.JCheckBox jCheckBoxSalida;
private javax.swing.JCheckBox jCheckBoxSonido;
private javax.swing.JCheckBox jCheckBoxUsarTeclas;
private javax.swing.JLabel jLabelLineas;
private javax.swing.JList jListFuentes;
private javax.swing.JList jListPais;
private javax.swing.JPanel jPanelBotones;
private javax.swing.JPanel jPanelColumnas;
private javax.swing.JPanel jPanelCursor;
private javax.swing.JPanel jPanelFuenteTerminal;
private javax.swing.JPanel jPanelModosTerminal;
private javax.swing.JPanel jPanelOpciones1;
private javax.swing.JPanel jPanelRC;
```

```
    private javax.swing.JPanel jPanelTraducciones;
    private javax.swing.JPanel jPanelopciones;
    private javax.swing.JRadioButton jRadioButton132;
    private javax.swing.JRadioButton jRadioButton80;
    private javax.swing.JRadioButton jRadioButtonBloque;
    private javax.swing.JRadioButton jRadioButtonLinea;
    private javax.swing.JScrollPane jScrollPaneFuente;
    private javax.swing.JScrollPane jScrollPanePais;
    private javax.swing.JTextField jTextFieldTamano;
    private javax.swing.JTextField jTextFieldlineas100;
    // End of variables declaration
}
```

B.6 Pod RS-232 Panel.

```
package rs232;

import java.awt.Graphics;
import java.awt.Color;

public class PodRS232Panel extends javax.swing.JPanel {
    private static int DISTANCIA_ENTRE_PINES=40;
    private static int DISTANCIA_NUMERO_A_PIN=20;
    private static int ANCHO_PIN=10;
    public static boolean CONECTANDO=true;
    public static boolean DESCONECTANDO=false;

    private int xConectores[]=new int[2];
    private int yConectores[]=new int[2];
    private int pinSeleccionadoEnConector0=-1;
    private int pinSeleccionadoEnConector1=-1;

    public int[] conexiones0=new int[25];
    public int[] conexiones1=new int[25];
    public int maxConexion=0;
    public int pulsadoEn1=-1;
    public int pulsadoEn0=-1;


    public boolean modo=CONECTANDO;

    /** Creates new form PodRS232Panel */
    public PodRS232Panel() {
        initComponents();
        inicializar();
    }
```

```
public void inicializar(){

    for(int i=0;i<25;i++){
        conexiones0[i]=-1;
        conexiones1[i]=-1;
    }
}



public void paintComponent(Graphics g){
    super.paintComponent(g);
    int x0,y0,x1,y1;
    dibujarConector(g,50,50,0);
    dibujarConector(g,350,50,1);
    for(int i=0;i<maxConexion;i++){
            x0=getXPin(0,conexiones0[i])+ANCHO_PIN/2;
            y0=getYPin(0,conexiones0[i])+ANCHO_PIN/2;
            x1=getXPin(1,conexiones1[i])+ANCHO_PIN/2;
            y1=getYPin(1,conexiones1[i])+ANCHO_PIN/2;
            g.setColor(Color.red);
            g.drawLine(x0, y0, x1, y1);
    }
}

public  boolean bienConectado(int uart,boolean ctrl_flujo){
    boolean b=false;
    if(ctrl_flujo==false){
        boolean existe023=false;
        boolean existe123=false;
        boolean existe045=false;
        boolean existe145=false;
        boolean existe077=false;
        boolean existe068=false;
        boolean existe0620=false;
        boolean existe168=false;
        boolean existe1620=false;
        for(int i=0;i<maxConexion;i++){
            existe023=existe023 || (conexiones0[i]==1 && conexiones1[i]==2);
```

```
                existe123=existe123 || (conexiones1[i]==1 && conexiones0[i]==2);
                existe045=existe045 || (conexiones0[i]==3 && conexiones1[i]==4);
                existe145=existe145 || (conexiones1[i]==3 && conexiones0[i]==4);
                existe077=existe077 || (conexiones0[i]==6 && conexiones1[i]==6);
                existe068=existe068 || (conexiones0[i]==5 && conexiones1[i]==7);
                existe0620=existe0620 || (conexiones0[i]==5 && conexiones1[i]==19);
                existe168=existe168 || (conexiones1[i]==5 && conexiones0[i]==7);
                existe1620=existe1620 || (conexiones1[i]==5 && conexiones0[i]==19);
            }
            b=existe023 && existe123 && existe045 && existe145 && existe077 && existe068 && existe0620 && existe168 &&
existe1620;
        }
        else{
            boolean existe023=false;
            boolean existe123=false;
            boolean existe045=false;
            boolean existe145=false;
            boolean existe077=false;

            for(int i=0;i<maxConexion;i++){
                existe023=existe023 || (conexiones0[i]==1 && conexiones1[i]==2);
                existe123=existe123 || (conexiones1[i]==1 && conexiones0[i]==2);
                existe045=existe045 || (conexiones0[i]==3 && conexiones1[i]==4);
                existe145=existe145 || (conexiones1[i]==3 && conexiones0[i]==4);
                existe077=existe077 || (conexiones0[i]==6 && conexiones1[i]==6);
            }
            b=((existe023 && existe123 && !existe045 && !existe145) || (uart==1 && existe023 && existe123 && existe045) ||
(uart==2 && existe023 && existe123 && existe145)) && existe077;
         }
         return b;
    }

    private int getXPin(int conector,int pin){
        int r=0;
        int x0=xConectores[conector];
        if(pin<14){
            r=x0+DISTANCIA_NUMERO_A_PIN;
        }
        else{
            r=x0+DISTANCIA_NUMERO_A_PIN+DISTANCIA_ENTRE_PINES;
        }
```

```
    return r;}

private int getYPin(int conector,int pin){
    int r=0;
    int y0=yConectores[conector];
    if(pin<14){
        r=y0+pin*DISTANCIA_ENTRE_PINES;
    }
    else{
        r=y0+(pin-14)*DISTANCIA_ENTRE_PINES+DISTANCIA_ENTRE_PINES*3/2;
    }
    return r;
}

private int[] pinEnCoordenadas(int x,int y){
    int d[]=new int[2];
    d[0]=-1;
    d[1]=-1;
    boolean r=false;
    int conector;
    int pin;
    conector=0;
    int xc,yc;

    while(!r && conector<2){
        pin=0;
        while(!r && pin<25){
            xc=getXPin(conector,pin);
            yc=getYPin(conector,pin);
            r=(x<=xc+ANCHO_PIN && x>=xc && y<=yc+ANCHO_PIN && y>=yc);
            if(r){
                d[0]=conector;
                d[1]=pin;
            }
            pin++;
        }
        conector++;
    }
    return d;
}
```

```
    private void dibujarConector(Graphics g,int x,int y,int n){
        xConectores[n]=x;
        yConectores[n]=y;
        int[] con;
        int pul;
        if(n==0){
            con=conexiones0;
            pul=pulsadoEn0;
        }
        else{
            con=conexiones1;
            pul=pulsadoEn1;
        }
        int i,j,k;
        for(i=0;i<13;i++){
            k=0;
            while(k<25 && con[k]!=i)k++;
            if(k<25 || i==pul){
                g.fillArc(x+DISTANCIA_NUMERO_A_PIN, y+i*DISTANCIA_ENTRE_PINES, ANCHO_PIN, ANCHO_PIN,0,360);
            }
            else{
                g.drawArc(x+DISTANCIA_NUMERO_A_PIN, y+i*DISTANCIA_ENTRE_PINES, ANCHO_PIN, ANCHO_PIN,0,360);
            }
            g.drawString(String.valueOf(i+1), x, y+i*DISTANCIA_ENTRE_PINES+10);

        }
        for(j=13;j<25;j++){
            i=j-14;
            k=0;
            while(k<25 && con[k]!=j)k++;
            if(k<25 || j==pul){
                g.fillArc(x+DISTANCIA_NUMERO_A_PIN+DISTANCIA_ENTRE_PINES, y+i*DISTANCIA_ENTRE_PINES+DISTANCIA_ENTRE_PINES*3/2,
ANCHO_PIN, ANCHO_PIN,0,360);
            }
            else{
                g.drawArc(x+DISTANCIA_NUMERO_A_PIN+DISTANCIA_ENTRE_PINES, y+i*DISTANCIA_ENTRE_PINES+DISTANCIA_ENTRE_PINES*3/2,
ANCHO_PIN, ANCHO_PIN,0,360);
            }
            g.drawString(String.valueOf(j+1), x+2*DISTANCIA_NUMERO_A_PIN+DISTANCIA_ENTRE_PINES,
y+i*DISTANCIA_ENTRE_PINES+DISTANCIA_ENTRE_PINES*3/2+10);
```

```
    }
}

private void eliminarConexion(int n){
    for(int i=n+1;i<conexiones0.length;i++){
        conexiones0[i-1]=conexiones0[i];
        conexiones1[i-1]=conexiones1[i];
    }
    maxConexion--;
}

public void setModo(boolean modo){
    this.modo=modo;
}

@SuppressWarnings("unchecked")

private void initComponents() {

    addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            formMouseReleased(evt);
        }
        public void mouseClicked(java.awt.event.MouseEvent evt) {
            formMouseClicked(evt);
        }
        public void mousePressed(java.awt.event.MouseEvent evt) {
            formMousePressed(evt);
        }
    });

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
    this.setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 357, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 408, Short.MAX_VALUE)
```

```
    );
}// </editor-fold>

private void formMouseClicked(java.awt.event.MouseEvent evt) {
    int d[];
    int i;

    d=pinEnCoordenadas(evt.getX(),evt.getY());
    if(modo==CONECTANDO){
        if(d[0]!=-1){
            if(d[0]==0){
                pulsadoEn0=d[1];
            }
            else{
                pulsadoEn1=d[1];
            }
            if(pulsadoEn0!=-1 && pulsadoEn1!=-1){
                boolean b=false;
                for(i=0;i<maxConexion;i++){
                    b=b || (conexiones0[i]==pulsadoEn0 && conexiones1[i]==pulsadoEn1);
                }
                if(!b){
                    conexiones0[maxConexion]=pulsadoEn0;
                    conexiones1[maxConexion]=pulsadoEn1;
                    maxConexion++;
                }
                pulsadoEn0=-1;
                pulsadoEn1=-1;
            }
        }
    }
    else{
        if(d[0]!=-1){
            if(d[0]==0){
                boolean b=false;
                for(i=0;i<maxConexion;i++){
                    b=conexiones0[i]==d[1];
                    if(b)break;
                }
                if(b){
                    eliminarConexion(i);
```

```
                    }
                }
                else{
                    boolean b=false;
                    for(i=0;i<maxConexion;i++){
                        b=conexiones1[i]==d[1];
                        if(b)break;
                    }
                    if(b){
                        eliminarConexion(i);
                    }
                }
                pulsadoEn0=-1;
                pulsadoEn1=-1;
            }
        }

        this.repaint();

    }

    public void desconectarTodo(){
        for(int i=0;i<conexiones0.length;i++){
            conexiones0[i]=-1;
            conexiones1[i]=-1;
        }
                maxConexion=0;
        this.repaint();
    }
    // Variables declaration - do not modify
    // End of variables declaration
}
```

B.7 POD RS-232.

```
package rs232;

public class PODRS232 extends javax.swing.JInternalFrame {
    public boolean[] senal;
    public int velocidad;

    private UART uart1;
    private UART uart2;
    private Osciloscopio osciloscopio;



    /** Creates new form PODRS232 */
    public PODRS232() {
        initComponents();

    }

    public void inicializar(UART uart1,UART uart2,Osciloscopio osciloscopio){
        this.uart1=uart1;
        this.uart2=uart2;
        this.osciloscopio=osciloscopio;
    }

    public void aceptarSenal(UART uart,int velocidad,boolean[] senal){
        this.senal=senal;
        this.velocidad=velocidad;
        boolean ctrl_flujo=(uart1.ctrl_flujo && uart2.ctrl_flujo);
        if(uart==uart1){
            if(this.podRS232Panel1.bienConectado(1,ctrl_flujo)){
                uart2.aceptarSenal(velocidad, senal);
                osciloscopio.aceptarSenal(velocidad,senal);
            }
        }
```

```
    else{
        if(this.podRS232Panel1.bienConectado(2,ctrl_flujo)){
            uart1.aceptarSenal(velocidad, senal);
            osciloscopio.aceptarSenal(velocidad,senal);
        }
    }


}

@SuppressWarnings("unchecked")

private void initComponents() {

    podRS232Panel1 = new rs232.PodRS232Panel();
    jToolBar1 = new javax.swing.JToolBar();
    jButtonConectar = new javax.swing.JButton();
    jButtonDesconectar = new javax.swing.JButton();
    jButtonDesconectar1 = new javax.swing.JButton();
    jButton1 = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
    setIconifiable(true);
    setMaximizable(true);
    setResizable(true);

    jToolBar1.setRollover(true);

    jButtonConectar.setText("Conectar");
    jButtonConectar.setFocusable(false);
    jButtonConectar.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    jButtonConectar.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
    jButtonConectar.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(java.awt.event.MouseEvent evt) {
            jButtonConectarMouseClicked(evt);
        }
    });
    jToolBar1.add(jButtonConectar);

    jButtonDesconectar.setText("Desconectar");
```

```
jButtonDesconectar.setFocusable(false);
jButtonDesconectar.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
jButtonDesconectar.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
jButtonDesconectar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButtonDesconectarMouseClicked(evt);
    }
});
jToolBar1.add(jButtonDesconectar);

jButtonDesconectar1.setText("Desconectar Todo");
jButtonDesconectar1.setFocusable(false);
jButtonDesconectar1.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
jButtonDesconectar1.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
jButtonDesconectar1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButtonDesconectar1MouseClicked(evt);
    }
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jButtonDesconectar1MousePressed(evt);
    }
});
jToolBar1.add(jButtonDesconectar1);

jButton1.setText("Ayuda");
jButton1.setFocusable(false);
jButton1.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
jButton1.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButton1MouseClicked(evt);
    }
});
jToolBar1.add(jButton1);
javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(jToolBar1, javax.swing.GroupLayout.DEFAULT_SIZE, 503, Short.MAX_VALUE)
    .addComponent(podRS232Panel1, javax.swing.GroupLayout.DEFAULT_SIZE, 503, Short.MAX_VALUE)
);
```

```
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jToolBar1, javax.swing.GroupLayout.PREFERRED_SIZE, 25, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(podRS232Panel1, javax.swing.GroupLayout.DEFAULT_SIZE, 574, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
        AyudaPod a=new AyudaPod(null,true);
        a.setVisible(true);
    }

     private void jButtonConectarMouseClicked(java.awt.event.MouseEvent evt) {
         this.podRS232Panel1.setModo(PodRS232Panel.CONECTANDO);
     }

     private void jButtonDesconectarMouseClicked(java.awt.event.MouseEvent evt) {
         this.podRS232Panel1.setModo(PodRS232Panel.DESCONECTANDO);
     }

     private void jButtonDesconectar1MousePressed(java.awt.event.MouseEvent evt) {
        podRS232Panel1.desconectarTodo();
     }

     public void setPeriodo(double periodo){
         osciloscopio.setPeriodo(periodo);
     }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButtonConectar;
    private javax.swing.JButton jButtonDesconectar;
    private javax.swing.JButton jButtonDesconectar1;
    private javax.swing.JToolBar jToolBar1;
    private rs232.PodRS232Panel podRS232Panel1;
    // End of variables declaration
}
```

B.8 Ayuda POD.

```
package rs232;

public class AyudaPod extends javax.swing.JDialog {

    /** Creates new form AyudaPod */
    public AyudaPod(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
         this.jTextPane1.setText("Pulse el boton 'Conectar' en la barra de herramientas y despues los pines para conectarlos.\n\n
       "+ "Pulse el boton 'Desconectar' en la barra de herramientas y despues el pin para desconectarlo.");
    }

    @SuppressWarnings("unchecked")

    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jLabel3 = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextPane1 = new javax.swing.JTextPane();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

        jButton1.setText("Aceptar");
        jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton1MouseClicked(evt);
            }
        });

        jLabel3.setFont(new java.awt.Font("Tahoma", 0, 48));
        jLabel3.setText("Ayuda");

        jTextPane1.setBorder(null);
        jTextPane1.setEditable(false);
```

```
    jTextPane1.setOpaque(false);
    jScrollPane1.setViewportView(jTextPane1);

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(23, 23, 23)
                 .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 479,
                javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGroup(layout.createSequentialGroup()
                    .addGap(185, 185, 185)
                    .addComponent(jLabel3))
                .addGroup(layout.createSequentialGroup()
                    .addGap(217, 217, 217)
                    .addComponent(jButton1)))
            .addContainerGap(32, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(7, 7, 7)
            .addComponent(jLabel3)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 125, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(18, 18, 18)
            .addComponent(jButton1)
            .addContainerGap(36, Short.MAX_VALUE))
    );

    pack();
}// </editor-fold>

private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
    this.setVisible(false);
}
```

```
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                AyudaPod dialog = new AyudaPod(new javax.swing.JFrame(), true);
                dialog.addWindowListener(new java.awt.event.WindowAdapter() {

                    public void windowClosing(java.awt.event.WindowEvent e) {
                        System.exit(0);
                    }
                });
                dialog.setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextPane jTextPane1;
    // End of variables declaration
}
```

B.9 Pantalla Osciloscopio.

```
package rs232;


import java.awt.Graphics;
import java.awt.Color;

public class PantallaOsciloscopio extends javax.swing.JPanel {

    private static int LONGITUD_BIT_DEFECTO=10;
    private static double TENSION_POR_DEFECTO=5;

    private int ANCHO;//ancho del panel
    private int ALTO;//alto del panel
    private int DIST_RAYITAS;//distancia entre rayitas por defecto
    private int EJEX;//posicion vertical del eje X
    private boolean senal[]=new boolean[0];//senal recibida mostrandose
    private int velocidad=9600;//velocidad por defecto
    private double tensionEntreRayitas=1;
    private int desplazamientoVertical=0;
    private int desplazamientoHorizontal=0;
    private int pHoriz=50;//poercentaje de desplazamiento horizontal
    private int pVert=0;//poercentaje de desplazamiento vertical
    private double escaladoX=1.0; //lo que se estira la senal en el sentido x


    /** Creates new form PantallaOsciloscopio */
    public PantallaOsciloscopio() {
        initComponents();


        //senal a 1 logico por defecto
        senal=new boolean[24];
        for(int i=0;i<24;i++){
            senal[i]=true;
        }
```

```
}

public void paintComponent(Graphics g){
    this.ANCHO=this.getWidth();
    this.ALTO=this.getHeight();
    this.DIST_RAYITAS=2;
    this.EJEX=ALTO/2;
    desplazamientoHorizontal=(ANCHO*pHoriz)/100;//desplazamiento Horizontal en pixels
    desplazamientoVertical=(ALTO*pVert)/100;//desplazamiento Vertical en pixels
    double factorx=9600.0/((double)velocidad);//factor estiramiento horizontal señal dependiendo de velocidad de transmision

    int i;
    int x,y;

    // Dibujamos los ejes y la graduacion de los mismos
    y=0;
    x=0;
    //Fondo
    Color color=new java.awt.Color(128,128,255);
    g.setColor(color);
    g.fillRect(0, 0,ANCHO , ALTO);
    g.setColor(Color.BLACK);
    //Ejes
    g.drawLine(0, ALTO/2, ANCHO, ALTO/2);
    g.drawLine(ANCHO/2,0,ANCHO/2,ALTO);

    //Rayitas de la graduciaon
    for(i=0;i<ANCHO/2;i=i+DIST_RAYITAS){
        g.drawLine(ANCHO/2+DIST_RAYITAS*i, ALTO/2-3, ANCHO/2+DIST_RAYITAS*i, ALTO/2+3);
        g.drawLine(ANCHO/2-DIST_RAYITAS*i, ALTO/2-3, ANCHO/2-DIST_RAYITAS*i, ALTO/2+3);
        if(i%5==0){
            g.drawLine(ANCHO/2+DIST_RAYITAS*i,0, ANCHO/2+DIST_RAYITAS*i, ALTO);
            g.drawLine(ANCHO/2-DIST_RAYITAS*i,0, ANCHO/2-DIST_RAYITAS*i, ALTO);
        }
    }

    for(i=0;i<ALTO/2;i=i+DIST_RAYITAS){
        g.drawLine(ANCHO/2-3, ALTO/2+DIST_RAYITAS*i, ANCHO/2+3, ALTO/2+DIST_RAYITAS*i);
        g.drawLine(ANCHO/2-3, ALTO/2-DIST_RAYITAS*i, ANCHO/2+3, ALTO/2-DIST_RAYITAS*i);
        if(i%5==0){
            g.drawLine(0, ALTO/2+DIST_RAYITAS*i, ANCHO, ALTO/2+DIST_RAYITAS*i);
```

```
                g.drawLine(0, ALTO/2-DIST_RAYITAS*i, ANCHO, ALTO/2-DIST_RAYITAS*i);
            }
        }

        color = new Color(140,255,255);
        g.setColor(color);

        //Dibujamos la linea que precede a la senal

g.drawLine(0,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas),desplazamientoHorizontal,E
JEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas));
        //Dibujar la linea vertical que une la senal por defecto con el inicio de la senal
        if(!senal[0]){

g.drawLine(desplazamientoHorizontal,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas),des
plazamientoHorizontal,EJEX+desplazamientoVertical-2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas));
        }
        //Dibujar la senal
        for(i=0;i<senal.length;i++){
            x=i*(int)(((double)LONGITUD_BIT_DEFECTO*factorx*escaladoX));
            if(senal[i]){
                y=+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas);
            }
            else{
                y=-2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas);
            }
            g.drawLine(desplazamientoHorizontal+x, desplazamientoVertical+y+EJEX,
desplazamientoHorizontal+x+(int)(((double)LONGITUD_BIT_DEFECTO*factorx*escaladoX)), desplazamientoVertical+y+EJEX);
            if(i>0 && senal[i-1]!=senal[i]){
                g.drawLine(desplazamientoHorizontal+x, desplazamientoVertical+y+EJEX, desplazamientoHorizontal+x,
desplazamientoVertical-y+EJEX);
            }
        }
        //dibujar la linea vertical que una el final de la senal con la senal por defecto
        if(!senal[i-1]){
            x=i*LONGITUD_BIT_DEFECTO*(int)(((double)LONGITUD_BIT_DEFECTO*factorx*escaladoX));
            g.drawLine(desplazamientoHorizontal+x, desplazamientoVertical+y+EJEX, desplazamientoHorizontal+x,
desplazamientoVertical-y+EJEX);
        }
        //Dibujar senal por defecto despues de la senal
```

```
            g.drawLine(desplazamientoHorizontal+x,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEn
            treRayitas),ANCHO,EJEX+desplazamientoVertical+2*DIST_RAYITAS*(int)(TENSION_POR_DEFECTO/tensionEntreRayitas));
        }

    public void setSenal(int velocidad,boolean[] senal){
        this.senal=senal;
        this.velocidad=velocidad;
        this.repaint();
    }



    public void setTensionEntreRayas(double tension){
        tensionEntreRayitas=tension/5;
        repaint();
    }

    public void setDesplazamientoHorizontal(int p){
        pHoriz=p;
        repaint();
    }
    public void setDesplazamientoVertical(int p){
        pVert=p-50;
        repaint();
    }

    public void incrementarEscaladoX(){
        escaladoX=escaladoX*1.2;
        repaint();
    }
    public void decrementarEscaladoX(){
        escaladoX=escaladoX/1.2;
        repaint();
    }

    @SuppressWarnings("unchecked")
```

```
    private void initComponents() {

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 400, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 300, Short.MAX_VALUE)
        );
    }// </editor-fold>
    // Variables declaration - do not modify
    // End of variables declaration
}
```

```
package rs232;

/**
 *
 * @author Jorge
 */
public class Osciloscopio extends javax.swing.JInternalFrame {
    double tension=5;
    /** Creates new form Osciloscopio */
    public Osciloscopio() {
        initComponents();
        jLabelTension.setText(String.valueOf(tension));
    }

    public void aceptarSenal(int velocidad,boolean[] senal){
        pantallaOsciloscopio1.setSenal(velocidad, senal);
    }
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        buttonGroupCanal = new javax.swing.ButtonGroup();
        buttonGroupValores = new javax.swing.ButtonGroup();
        pantallaOsciloscopio1 = new rs232.PantallaOsciloscopio();
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jButtonMas = new javax.swing.JButton();
        jButtonMenos = new javax.swing.JButton();
        jLabelTension = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jLabelTiempo = new javax.swing.JLabel();
        jSlider2 = new javax.swing.JSlider();
        jSlider1 = new javax.swing.JSlider();
```

```
setClosable(true);
setIconifiable(true);
setMaximizable(true);
setResizable(true);

javax.swing.GroupLayout pantallaOsciloscopio1Layout = new javax.swing.GroupLayout(pantallaOsciloscopio1);
pantallaOsciloscopio1.setLayout(pantallaOsciloscopio1Layout);
pantallaOsciloscopio1Layout.setHorizontalGroup(
    pantallaOsciloscopio1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, 484, Short.MAX_VALUE)
);
pantallaOsciloscopio1Layout.setVerticalGroup(
    pantallaOsciloscopio1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, 294, Short.MAX_VALUE)
);

getContentPane().add(pantallaOsciloscopio1, java.awt.BorderLayout.CENTER);

jLabel1.setText("V - DIV");

jButtonMas.setText("+");
jButtonMas.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jButtonMasMousePressed(evt);
    }
});

jButtonMenos.setText("-");
jButtonMenos.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jButtonMenosMousePressed(evt);
    }
});

jLabelTension.setText("jLabel2");

jButton1.setText("+");
jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButton1MouseClicked(evt);
    }
```

```
        });

        jButton2.setText("-");
        jButton2.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton2MouseClicked(evt);
            }
        });

        jLabel2.setText("ms");

        jLabelTiempo.setText("0.1");

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jButtonMas)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(jButtonMenos)
                        .addGap(18, 18, 18)
                        .addComponent(jLabelTension))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(28, 28, 28)
                        .addComponent(jLabel1)))
                .addGap(85, 85, 85)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jButton1)
                        .addGap(18, 18, 18)
                        .addComponent(jButton2))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabelTiempo, javax.swing.GroupLayout.DEFAULT_SIZE, 51, Short.MAX_VALUE)
                        .addGap(19, 19, 19)
                        .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 26,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(163, Short.MAX_VALUE))
```

```
);
jPanel1Layout.setVerticalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(jPanel1Layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
            .addComponent(jLabel2)
            .addComponent(jLabelTiempo))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jButtonMas)
            .addComponent(jButtonMenos)
            .addComponent(jLabelTension)
            .addComponent(jButton1)
            .addComponent(jButton2))
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
);

getContentPane().add(jPanel1, java.awt.BorderLayout.PAGE_END);

jSlider2.setOrientation(javax.swing.JSlider.VERTICAL);
jSlider2.setToolTipText("Posición Vertical");
jSlider2.setCursor(new java.awt.Cursor(java.awt.Cursor.N_RESIZE_CURSOR));
jSlider2.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        jSlider2StateChanged(evt);
    }
});
getContentPane().add(jSlider2, java.awt.BorderLayout.EAST);

jSlider1.setToolTipText("Posición Horizontal");
jSlider1.setCursor(new java.awt.Cursor(java.awt.Cursor.E_RESIZE_CURSOR));
jSlider1.setInheritsPopupMenu(true);
jSlider1.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        jSlider1StateChanged(evt);
    }
});
jSlider1.addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseDragged(java.awt.event.MouseEvent evt) {
```

```
            jSlider1MouseDragged(evt);
        }
        public void mouseMoved(java.awt.event.MouseEvent evt) {
            jSlider1MouseMoved(evt);
        }
    });
    getContentPane().add(jSlider1, java.awt.BorderLayout.PAGE_START);

    pack();
}// </editor-fold>

private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
   pantallaOsciloscopio1.setDesplazamientoHorizontal(jSlider1.getValue());
}

private void jSlider2StateChanged(javax.swing.event.ChangeEvent evt) {
    pantallaOsciloscopio1.setDesplazamientoVertical(100-jSlider2.getValue());
}

private void jButtonMasMousePressed(java.awt.event.MouseEvent evt) {
    if(tension<10){
        tension=tension+1;
    }
    pantallaOsciloscopio1.setTensionEntreRayas(tension);
    jLabelTension.setText(String.valueOf(tension));
}

private void jButtonMenosMousePressed(java.awt.event.MouseEvent evt) {
    if(tension>1){
        tension=tension-1;
    }
    pantallaOsciloscopio1.setTensionEntreRayas(tension);
    jLabelTension.setText(String.valueOf(tension));
}

public void setPeriodo(double periodo){
    jLabelTiempo.setText("0."+String.valueOf(Math.round(10000*periodo)));
}
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
    pantallaOsciloscopio1.incrementarEscaladoX();
}
```

```
    private void jButton2MouseClicked(java.awt.event.MouseEvent evt) {
        pantallaOsciloscopio1.decrementarEscaladoX();
    }

    private void jSlider1MouseMoved(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        System.out.print("Posición Horizontal");
    }

    private void jSlider1MouseDragged(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
    }

    // Variables declaration - do not modify
    private javax.swing.ButtonGroup buttonGroupCanal;
    private javax.swing.ButtonGroup buttonGroupValores;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButtonMas;
    private javax.swing.JButton jButtonMenos;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabelTension;
    private javax.swing.JLabel jLabelTiempo;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JSlider jSlider1;
    private javax.swing.JSlider jSlider2;
    private rs232.PantallaOsciloscopio pantallaOsciloscopio1;
    // End of variables declaration
}
```

B.10 Osciloscopio.

```
package rs232;

public class Osciloscopio extends javax.swing.JInternalFrame {
    double tension=5;
    /** Creates new form Osciloscopio */
    public Osciloscopio() {
        initComponents();
        jLabelTension.setText(String.valueOf(tension));
    }

    public void aceptarSenal(int velocidad,boolean[] senal){
        pantallaOsciloscopio1.setSenal(velocidad, senal);
    }

    @SuppressWarnings("unchecked")

    private void initComponents() {

        buttonGroupCanal = new javax.swing.ButtonGroup();
        buttonGroupValores = new javax.swing.ButtonGroup();
        pantallaOsciloscopio1 = new rs232.PantallaOsciloscopio();
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jButtonMas = new javax.swing.JButton();
        jButtonMenos = new javax.swing.JButton();
        jLabelTension = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jLabelTiempo = new javax.swing.JLabel();
        jSlider2 = new javax.swing.JSlider();
        jSlider1 = new javax.swing.JSlider();
```

```
setClosable(true);
setIconifiable(true);
setMaximizable(true);
setResizable(true);

javax.swing.GroupLayout pantallaOsciloscopio1Layout = new javax.swing.GroupLayout(pantallaOsciloscopio1);
pantallaOsciloscopio1.setLayout(pantallaOsciloscopio1Layout);
pantallaOsciloscopio1Layout.setHorizontalGroup(
    pantallaOsciloscopio1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, 484, Short.MAX_VALUE)
);
pantallaOsciloscopio1Layout.setVerticalGroup(
    pantallaOsciloscopio1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, 294, Short.MAX_VALUE)
);

getContentPane().add(pantallaOsciloscopio1, java.awt.BorderLayout.CENTER);

jLabel1.setText("V - DIV");

jButtonMas.setText("+");
jButtonMas.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jButtonMasMousePressed(evt);
    }
});

jButtonMenos.setText("-");
jButtonMenos.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        jButtonMenosMousePressed(evt);
    }
});

jLabelTension.setText("jLabel2");



jButton1.setText("+");
jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
```

```
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButton1MouseClicked(evt);
    }
});

jButton2.setText("-");
jButton2.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButton2MouseClicked(evt);
    }
});

jLabel2.setText("ms");

jLabelTiempo.setText("0.1");

javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(jPanel1Layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(jButtonMas)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jButtonMenos)
                .addGap(18, 18, 18)
                .addComponent(jLabelTension))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(28, 28, 28)
                .addComponent(jLabel1)))
        .addGap(85, 85, 85)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(jButton1)
                .addGap(18, 18, 18)
                .addComponent(jButton2))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
                .addComponent(jLabelTiempo, javax.swing.GroupLayout.DEFAULT_SIZE, 51, Short.MAX_VALUE)
                .addGap(19, 19, 19)
```

```
                .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 26,
            javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addContainerGap(163, Short.MAX_VALUE))
);
jPanel1Layout.setVerticalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(jPanel1Layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
            .addComponent(jLabel2)
            .addComponent(jLabelTiempo))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jButtonMas)
            .addComponent(jButtonMenos)
            .addComponent(jLabelTension)
            .addComponent(jButton1)
            .addComponent(jButton2))
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
);

getContentPane().add(jPanel1, java.awt.BorderLayout.PAGE_END);

jSlider2.setOrientation(javax.swing.JSlider.VERTICAL);
jSlider2.setToolTipText("Posición Vertical");
jSlider2.setCursor(new java.awt.Cursor(java.awt.Cursor.N_RESIZE_CURSOR));
jSlider2.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        jSlider2StateChanged(evt);
    }
});
getContentPane().add(jSlider2, java.awt.BorderLayout.EAST);




jSlider1.setToolTipText("Posición Horizontal");
jSlider1.setCursor(new java.awt.Cursor(java.awt.Cursor.E_RESIZE_CURSOR));
```

```
    jSlider1.setInheritsPopupMenu(true);
    jSlider1.addChangeListener(new javax.swing.event.ChangeListener() {
        public void stateChanged(javax.swing.event.ChangeEvent evt) {
            jSlider1StateChanged(evt);
        }
    });
    jSlider1.addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
        public void mouseDragged(java.awt.event.MouseEvent evt) {
            jSlider1MouseDragged(evt);
        }
        public void mouseMoved(java.awt.event.MouseEvent evt) {
            jSlider1MouseMoved(evt);
        }
    });
    getContentPane().add(jSlider1, java.awt.BorderLayout.PAGE_START);

    pack();
}// </editor-fold>

private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
   pantallaOsciloscopio1.setDesplazamientoHorizontal(jSlider1.getValue());
}

private void jSlider2StateChanged(javax.swing.event.ChangeEvent evt) {
    pantallaOsciloscopio1.setDesplazamientoVertical(100-jSlider2.getValue());
}

private void jButtonMasMousePressed(java.awt.event.MouseEvent evt) {
    if(tension<10){
        tension=tension+1;
    }
    pantallaOsciloscopio1.setTensionEntreRayas(tension);
    jLabelTension.setText(String.valueOf(tension));
}




private void jButtonMenosMousePressed(java.awt.event.MouseEvent evt) {
    if(tension>1){
```

```
            tension=tension-1;
        }
        pantallaOsciloscopio1.setTensionEntreRayas(tension);
        jLabelTension.setText(String.valueOf(tension));
    }

    public void setPeriodo(double periodo){
        jLabelTiempo.setText("0."+String.valueOf(Math.round(10000*periodo)));
    }
    private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
        pantallaOsciloscopio1.incrementarEscaladoX();
    }

    private void jButton2MouseClicked(java.awt.event.MouseEvent evt) {
        pantallaOsciloscopio1.decrementarEscaladoX();
    }

    // Variables declaration - do not modify
    private javax.swing.ButtonGroup buttonGroupCanal;
    private javax.swing.ButtonGroup buttonGroupValores;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButtonMas;
    private javax.swing.JButton jButtonMenos;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabelTension;
    private javax.swing.JLabel jLabelTiempo;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JSlider jSlider1;
    private javax.swing.JSlider jSlider2;
    private rs232.PantallaOsciloscopio pantallaOsciloscopio1;
    // End of variables declaration
}
```

Anexo C.
Trabajo a realizar por el alumno.

H. Cuestionario 1:

a. (7.2.2) Dibujar la forma de onda para cada una de las configuraciones marcando los niveles de tensión y tiempos de bit. Determinar para cada una de las configuraciones que bits se han transmitido.

b. (7.2.3) Cuando se transmite el carácter 'a' ¿coinciden los bits que se ven en la pantalla con el código ASCII del carácter 'a'?.

c. (7.2.3) ¿Dónde esta el bit de paridad?.

d. (7.2.3) Observar que ocurre (en ambos sentidos de la comunicación) cuando las velocidades son distintas. Dar una explicación.

e. (7.3.1) ¿Cuáles son los códigos ASCII de los caracteres Xon y Xoff?.

f. (7.3.1) ¿Qué efectos tienen los caracteres Xon yXoff ?.

g. (7.3.2) ¿Qué efectos tienen sobre la comunicación entre los dos PC's desconectar la línea CTS de un PC?.

h. (7.3.2) ¿Qué efectos tienen sobre la comunicación entre los dos PC's desconectar la línea RTS de un PC?.

i. (7.4.1) ¿Cómo se comporta el osciloscopio, como ETD o como ETCD?.

j. (7.4.2) ¿Qué control de flujo hay que usar en el terminal y como son las conexiones del cable?.

k. (7.4.2) ¿Cómo se debe configurar al equipo terminal?.

l. (7.4.3) Determinar los comandos que hay que enviar para realizar las siguientes funciones sobre el osciloscopio:

- Activar el canal 1.
- Poner la base de tiempos a 1ms.
- Poner el canal 1 a 5 voltios por división.
- El canal 1 en acoplo en continua.

I. Cuestionario 2:

a. (7.5.1) Determinar:

- El valor de SOH.
- Cuantos bytes componen la trama.
- Cuantos bytes componen el campo de control de errores y su valor.
- Cuantas tramas se necesitan para transmitir el fichero.

b. (7.5.1) Realizar un datagrama de la transición en donde se muestren las fases de la comunicación y la duración de cada una de ellas.

c. (7.5.1) ¿Cómo funciona el control de flujo?.

d. (7.5.1) ¿Cómo se detectan errores?.

e. (7.5.1) ¿Qué sucede cuando hay errores en la recepción?.

f. (7.5.1) Calcular:

- Tiempo de transición del fichero sin protocolo X-Módem.
- Tiempo de transición del fichero con protocolo X-Módem.
- El rendimiento teórico al usar X-Módem.