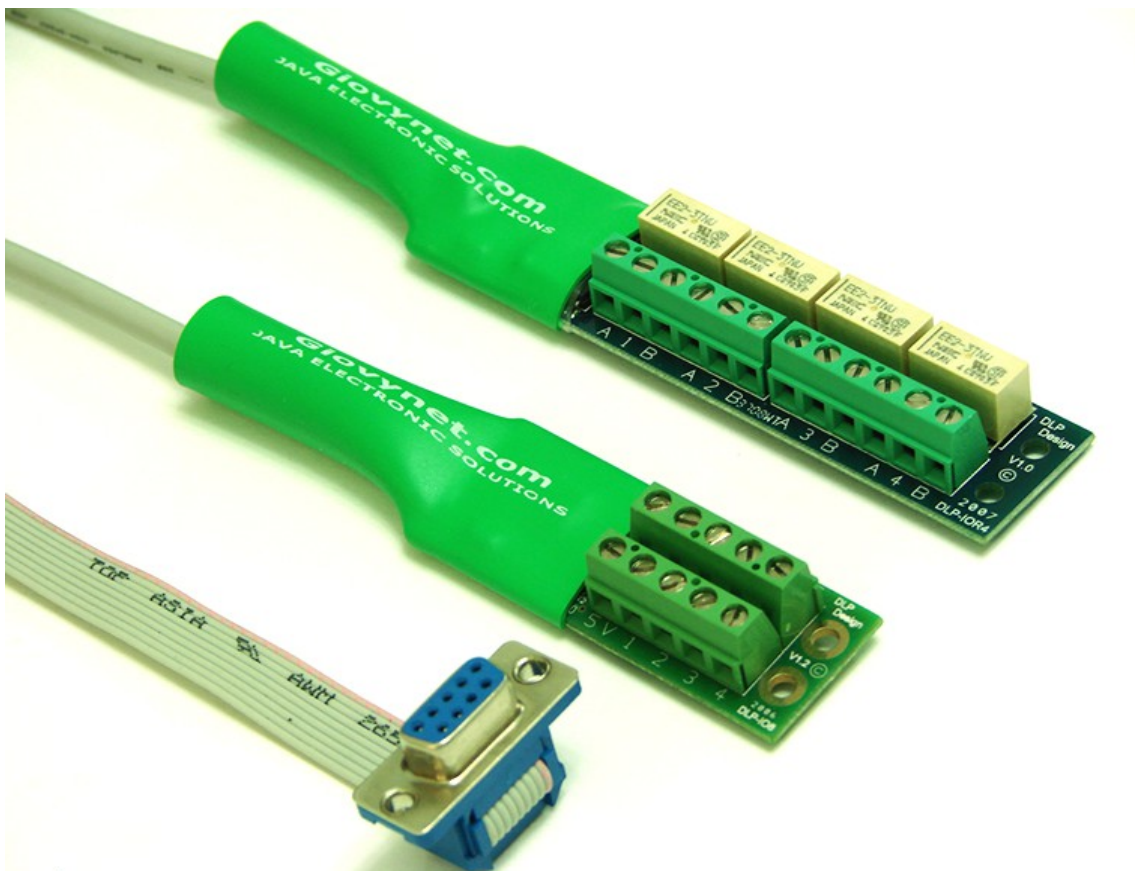


Manual de Giovynet Driver Versión 2.0

Giovanny Rey Cediel





© 2012 Giovynet

2ª edición



Reconocimiento de marcas

Java, Java Micro Edition (JME), Java Standard Edition (JSE), Java Enterprise Edition (JEE) y Java Runtime Environment (JRE), son marcas registradas de Oracle Corporation. DLP-IO8 y DLP-IOR4 son marcas registradas de DLP Design, Inc.

Nota aclaratoria

La información aquí contenida está sujeta a cambios sin previo aviso. Este documento es sólo para fines informativos. Giovynet.com y su personal no ofrecen garantías de ningún tipo por la exactitud, integridad, la interpretación o el uso de la información contenida en este documento. Es su responsabilidad cumplir con todas las leyes de derechos de autor. Giovynet.com puede tener patentes, solicitudes de patentes, marcas registradas, derechos de autor u otros derechos de propiedad intelectual sobre los contenidos en este documento. Salvo lo dispuesto expresamente en cualquier acuerdo por escrito con Giovynet.com, la entrega de este documento no otorga ninguna licencia sobre estas patentes, marcas, derechos de autor u otra propiedad intelectual.

Tabla de Contenido

Manual de Giovynet Driver Versión 2.0.....	1
Capítulo 1. Introducción a Giovynet Driver.....	6
Distribuciones y tipos de licencia.....	7
Componentes.....	8
Arquitectura.....	9
Capítulo 2. Instalación y Configuración de Herramientas para Desarrollo.....	10
Instalación y configuración de Sun JDK para Windows.....	10
Instalación y configuración de Sun JDK para Ubuntu Linux.....	12
Instalación de Eclipse IDE Para Windows.....	14
Instalación de Eclipse IDE Para Ubuntu Linux.....	16
Capítulo 3. Creación de un proyecto Java con Giovynet Driver en Eclipse IDE.....	20
Capítulo 4. Puerto GIV8DAQ.....	32
Dimensiones.....	33
Especificaciones.....	33
Máximos rangos.....	33
Consideraciones importantes para Windows.....	34
Programando el puerto GIV8DAQ con Eclipse IDE y Java.....	35
¿Cómo saber cuantos dispositivos se pueden Instanciar?.....	35
¿Cómo saber si hay dispositivos GIV8DAQ conectados?.....	35
¿Cómo obtener una instancia de los dispositivos GIV8DAQ conectados?.....	35
¿Cómo activar un puerto GIV8DAQ?.....	36
¿Cómo saber cual es el puerto de comunicaciones asociado?.....	36
¿Cómo establecer un canal en alto (5VDC)?.....	37
¿Cómo establecer un canal en “bajo” o “falso” (0VDC)?.....	38
¿Cómo obtener el valor digital (alto o bajo) de un canal?.....	38
¿Cómo obtener lectura de voltaje de un canal?.....	39
¿Cómo conectar el sensor de temperatura GIV18B20 en un canal?.....	40
¿Cómo obtener lectura de temperatura?.....	41
¿Cómo establecer el modo de lectura de temperatura en escala Fahrenheit o Celsius?.....	42
Consideraciones importantes en lecturas de temperatura.....	43
¿Cómo saber la cantidad de puertos GIV8DAQ instanciados?.....	44
¿Cómo finalizar una instancia de un puerto GIV8DAQ?.....	44
Capítulo 5. Puerto GIV4R.....	45
Dimensiones.....	46
Especificaciones.....	46
Consideraciones importantes para Windows.....	46
Programando el puerto GIV4R con Eclipse IDE y Java.....	47
¿Cómo saber cuantos dispositivos se pueden instanciar?.....	48
¿Cómo saber si hay dispositivos GIV4R conectados?.....	48
¿Cómo obtener una instancia de los dispositivos GIV4R conectados?.....	48
¿Cómo activar un puerto GIV4R?.....	49

¿Cómo saber cual es el puerto de comunicaciones asociado?.....	49
¿Cómo conectar el terminal común de un relevo con el borne A?.....	50
¿Cómo conectar el terminal común de un relevo con el borne B?.....	51
¿Cómo saber la cantidad de puertos GIV4R instanciados?.....	51
¿Cómo finalizar una instancia de un puerto GIV4R?.....	51
Capítulo 6. Envío y Recepción de Caracteres ASCII A Través del Puerto Serie RS-232.....	60
¿Cómo saber que puertos serie hay libres?.....	53
¿Cómo configurar el puerto serie?.....	53
¿Cómo enviar datos?.....	54
¿Cómo recibir datos?.....	55
¿Cómo enviar caracteres de control?.....	57
¿Cómo recibir caracteres de control?.....	58
¿Cómo implementar hilo para recibir datos de forma independiente?.....	58
Capítulo 7. Distribución de Aplicaciones.....	61
¿Que es y cómo se instala el JRE?.....	61
¿Cómo crear un JAR ejecutable con Eclipse IDE?.....	61
¿Cómo crear un folder de distribución?.....	63
¿Cómo iniciar una aplicación desde el folder de distribución?.....	64
Capítulo 8. Excepciones Frecuentes.....	66

Capítulo 1. Introducción a Giovynet Driver

En ocasiones los fabricantes de dispositivos electrónicos se enfrentan con la problemática de comunicar o controlar sus creaciones con un computador, muchos intentan trabajar con Java, sin embargo declinan su intención por los complicados procesos que deben seguir para manipular hardware externo. Es aquí donde se presenta Giovynet Driver como una opción de Java para manejar circuitos externos.

De manera formal, Giovynet Driver es un marco de trabajo o “framework” que posibilita el uso de lenguaje Java para crear aplicaciones que se comuniquen con circuitos externos al PC.

Giovynet Driver faculta a Java, para interfazar circuitos electrónicos y/o circuitos electromecánicos desde un ordenador. En consecuencia Java se convierte en una opción para el fabricante de hardware que desea comunicar sus creaciones con un PC. De la misma manera Giovynet Driver abre el camino para que aplicaciones previamente construidas en Java sean capaces de manipular mecanismos o máquinas.

Giovynet Driver versión 2.0 usa los siguientes puertos de comunicaciones como medio de enlace entre Java y los dispositivos o circuitos externos:

- **Puerto USB GIV8DAQ:** consiste en un módulo cuya conexión al ordenador es USB, no necesita fuente de energía externa para funcionar (la toma del PC), presenta tamaño y diseño adecuados para acoplarse fácilmente a circuitos electrónicos. Presenta ocho canales independientes en funcionamiento y configuración. Cada canal permite a Java monitorear o sensar voltajes, controlar procesos, adquirir datos de temperatura, disparar relevos, establecer voltajes lógicos (0-5VDC), etc. Para ver datos técnicos acerca de este módulo diríjase al capítulo 4.
- **Puerto USB GIV4R:** Consiste en un módulo cuya conexión al ordenador es USB, no necesita fuente de energía externa para funcionar (la toma del PC), presenta tamaño y diseño adecuados para acoplarse fácilmente a circuitos electrónicos. Consiste en cuatro relevos independientes, estos guardan su estado en caso de desconexión o apagado del PC. Este dispositivo es pensado para usuarios que desean activar o desactivar programáticamente circuitos de mayor potencia que la del propio PC. En sus bornes puede resistir voltajes de 110VDC a 220 VDC, con una corriente máxima de hasta 2 Amperios. Para ver más detalles acerca de este módulo diríjase al capítulo 5.

- **Puerto Serie:** Giovynet Driver soporta el **envío y recepción de caracteres ASCII a través del puerto serie RS - 232**, en conexión “null modem” (no handshaking) es decir, solo se usan las líneas: transmisión, recepción y común o “ground”. Esto indica que el control de flujo no se realiza por “hardware”, esta tarea queda a cargo del desarrollador. En el capítulo 6, se hablará en detalle de las instrucciones Java necesarias para manipular este dispositivo.

Distribuciones y tipos de licencia

Giovynet Driver versión 2.0 trabaja bajo los sistemas operativos Windows y Linux, y se distribuye en tres “sabores” según el uso, cantidad de dispositivos (número de instancias), y arquitectura (x86 o x64). A continuación se describe cada una de las diferentes distribuciones de Giovynet Driver:

Giovynet Driver For Personal (x86)

Esta distribución es gratis, se puede descargar desde el sitio oficial de Giovynet.com, es compilada para arquitecturas x86 (32- bit) únicamente, es pensada para desarrolladores que desean realizar una aplicación con fines de prueba y/o aprendizaje. Se permite el uso de un solo dispositivo en una aplicación, esto quiere decir que se puede usar, ya sea un puerto serie, o un puerto GIV8DAQ, o un puerto GIV4R. Si se intenta usar dos o más dispositivos la aplicación lanzará una excepción. La licencia para esta distribución, **prohíbe el uso de la misma en trabajos con fines de lucro.**

Giovynet Driver For Bussines Four Devices (x86/x64)

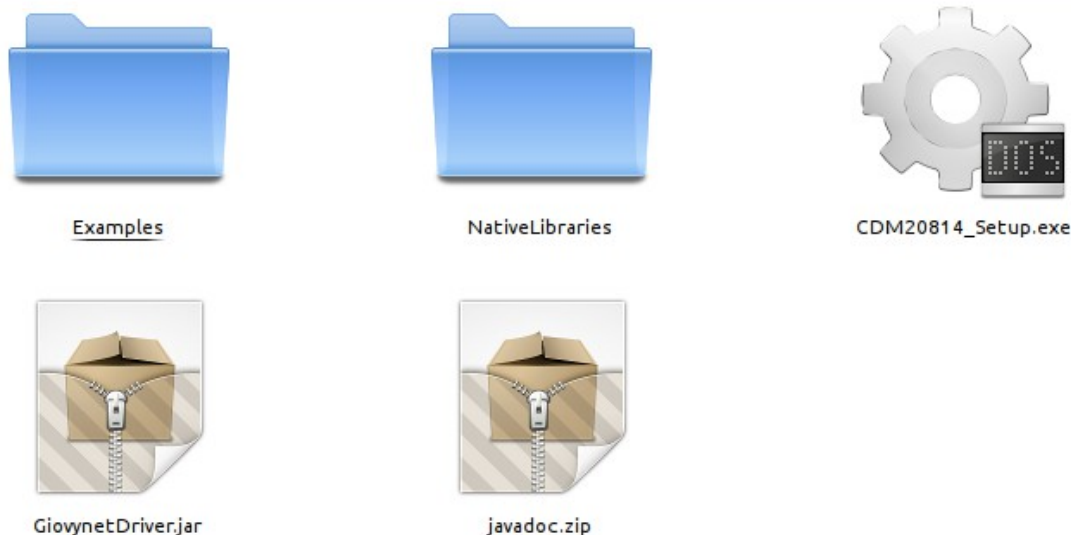
Esta distribución se puede obtener por un valor comercial (ver detalles en Giovynet.com), se puede comprar para arquitectura x86 (32- bit), o para arquitectura x64 (64- bit). Es pensada para desarrolladores que desean usar hasta cuatro dispositivos en una aplicación, se puede usar combinación de puertos serie, puertos GIV8DAQ, o puertos GIV4R. Si se intenta usar más de cuatro dispositivos se producirá una excepción que detendrá la aplicación. La licencia para esta distribución **permite realizar trabajos con fines lucrativos.**

Giovynet Driver For Bussines Sixteen Devices (x86/x64)

Esta distribución tiene un valor comercial (ver detalles en Giovynet.com), se puede comprar para arquitectura x86 (32-bit), o para arquitectura x64 (64-bit). Es pensada para desarrolladores que desean usar hasta dieciséis dispositivos en una aplicación, se puede usar o instanciar una combinación de puertos serie, puertos GIV8DAQ, o puertos GIV4R. Si se intenta instanciar más de dieciséis dispositivos se producirá una excepción que detendrá la aplicación. La licencia para esta distribución **permite realizar trabajos con fines lucrativos.**

Componentes

Giovynet Driver viene empaquetado en un archivo “zip”. Al descomprimir este archivo se generan dos folders y tres archivos:



El folder **Examples** contiene un proyecto con código fuente comentariado, que conforma aplicaciones sencillas cuyo propósito es demostrar el manejo de los puertos de comunicaciones GIV8DAQ, GIV4R, y serie (RS-232).

El folder **NativeLibraries** contiene dos importantes archivos, ellos son: *libSerialPort.dll* y *libSOSerialPort.so*. Estos archivos deben estar en la carpeta principal del proyecto Java cuando se esta trabajando en una aplicación, y cuando se realiza el “despliegue” o “deploy” del proyecto estos archivos deben estar en el mismo folder que el archivo “JAR” resultado del despliegue. De lo contrario la aplicación lanzará una excepción de tipo **UnsatisfiedLinkError**.

El archivo **javadoc.zip** es la documentación estándar Java de las clases y métodos de Giovynet Driver.

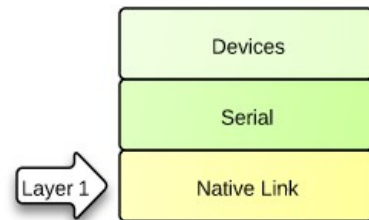
El archivo Java **GiovynetDriver.jar**, contiene paquetes y clases necesarios para controlar las interfaces físicas (puertos GIV8DAQ, GIV4R y RS-232) entre el circuito externo y una aplicación Java. Este archivo debe ser referenciado en la ruta o “path” del proyecto, de lo contrario el entorno de desarrollo marcará errores de sintaxis.

El archivo **CDM208014_Setup.exe**, consiste en una aplicación segura y verificada para Windows, es distribuida por la empresa de tecnología: Future Technology Devices International Ltda. Esta aplicación instala en Windows los “drivers” para que este sistema operativo reconozca los puertos GIV¹, ya que estos puertos son construidos internamente con un chip de Future Technology Devices International. Sucede que algunas versiones de Windows no tienen incluido este “driver”, por tanto se debe instalar manualmente dando dobleclick sobre este archivo. Esta situación no se presenta en Linux porque el “driver” está incluido en el kernel, de tal manera que Linux reconoce estos dispositivos automáticamente.

Arquitectura

La arquitectura interna de Giovynet Driver se puede describir fácilmente en tres capas, donde las capas inferiores soportan las capas superiores.

La primera capa, **Native Link**, tiene el propósito de establecer comunicación entre una aplicación Java y el sistema operativo. Por ejemplo, si una aplicación necesita saber que puertos están conectados al PC; Java lanzará una consulta a **Native Link**. Cuando **Native Link** recibe la consulta, la traduce a lenguaje propio del sistema operativo y posteriormente la ejecuta. Esta ejecución produce una respuesta que es traducida a lenguaje Java y enviada de vuelta a la aplicación, en este caso se traduce las respuestas del sistema operativo a código Java.



La segunda capa, **Serial**, constituye todas las sentencias Java necesarias para controlar los puertos serie conectados al PC.

Finalmente la tercera capa, **Devices**, conforma todas las sentencias Java necesarias para controlar los puertos GIV que constituyen la interfaz física entre la aplicación Java y el circuito externo.

1 Puertos GIV hace referencia a los puertos GIV8DAQ y GIV4R.

Capítulo 2. Instalación y configuración de herramientas para desarrollo

En este capítulo se explicará la instalación y configuración de Sun JDK y de Eclipse IDE para los sistemas operativos Windows y Linux. Si el lector conoce estos temas puede sentirse en libertad de saltar este capítulo. Aunque los temas aquí tratados son ampliamente difundidos en la comunidad de programadores Java, se incluyen como una referencia complementaria a usuarios conocedores o neófitos.

Como se mencionó en el primer capítulo, Giovynet Driver es un marco de trabajo o “framework” de desarrollo para lenguaje Java. Por lo tanto, se requiere de un conjunto de programas y librerías que permiten compilar, ejecutar y depurar el código java. Este conjunto de programas y librerías se conoce como Sun JDK (iniciales de Java Development Kit), el cual se distribuye de forma gratuita por Oracle en su sitio web oficial. Existen varias versiones del Sun JDK para varios sistemas operativos entre los cuales se encuentra, Windows y Linux.

Otra herramienta no menos importante es el entorno de desarrollo integrado o IDE (iniciales de Integrated development Environment). El IDE es una herramienta amigable que permite hacer uso eficiente del Sun JDK. Con un IDE se construyen aplicaciones de manera fácil y rápida.

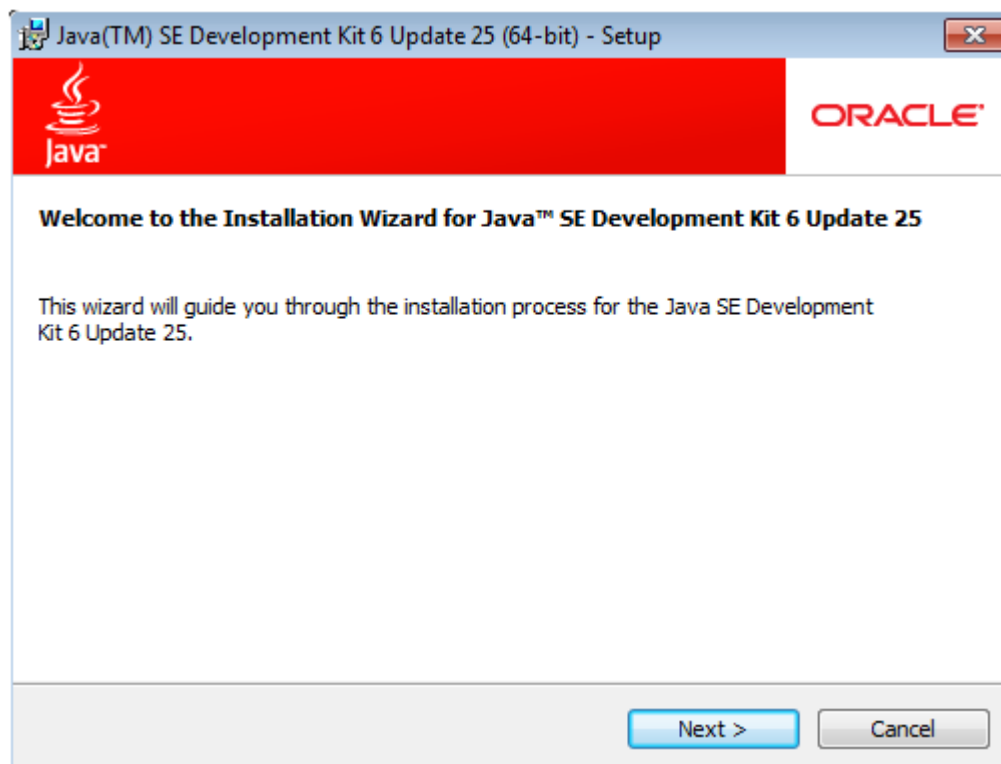
En la actualidad existen varios IDE para trabajar en Java, los más conocidos son Eclipse y NetBeans, en este libro se usará Eclipse IDE.

Instalación y configuración de Sun JDK para Windows

Los ejemplos aquí citados se realizaron bajo Windows 7, El procedimiento varia muy poco en versiones como XP o Vista. Instalar el Sun JDK es fácil, primero diríjase al sitio oficial de descargas de Oracle: <http://www.oracle.com/technetwork/java/javase/downloads/index.html> seleccione plataforma y arquitectura (Windows x32 o Windows x64) y descargue el archivo.

El archivo descargado tendrá un nombre similar a *jdk-6u25-windows-x...exe*, luego de dar doble click sobre este, se lanzará un asistente que le guiará paso por paso en la instalación del Sun JDK. Al finalizar, el sistema lanzará un formulario para registrar el producto si lo desea, este proceso es gratis y opcional.

La siguiente imagen muestra el asistente de instalación del Sun JDK.

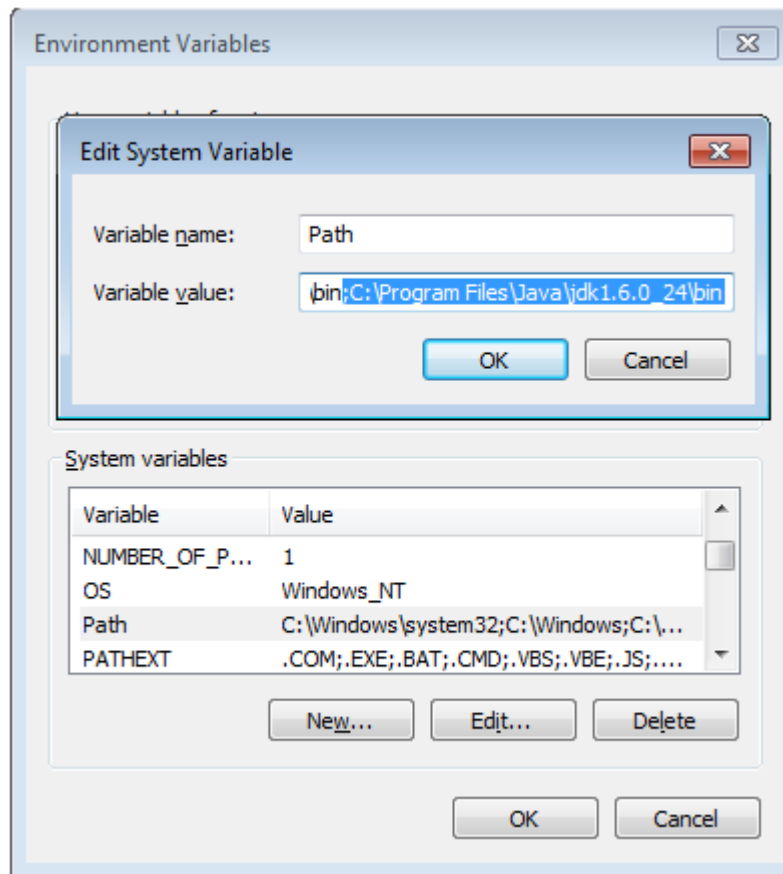


Como resultado de la instalación, se crearán una serie de “folders” o carpetas con nombre distintivo para cada versión de Sun JDK, en este caso se creó la ruta C:\Program Files\Java\jdk1.6.0_25\bin\, debido a que la versión de JDK instalada fué 1.6.025. En esta ubicación están todas las herramientas necesarias para compilar, ejecutar y depurar el código fuente java.

El siguiente paso es configurar la ruta de instalación en la variable de entorno “path. Esto permitirá referenciar el folder jdk1.6.0_25\bin\, desde cualquier ubicación en el sistema operativo y se hace con el fin de invocar el compilador sin necesidad de ir a la ruta de instalación del JDK.

Para realizar esta tarea, diríjase al botón “Inicio”, de click derecho en “Equipo”, luego de click en “Propiedades”, seleccione el enlace “Configuración Avanzada del Sistema”, luego de click en la ficha “Configuración Avanzada”, seguidamente de click en el botón “Variables de entorno”. A continuación aparecerá una ventana que muestra las variables del sistema, búsquela variable “path” en el área “Variables del Sistema” luego de click en el botón “Editar”, en seguida aparecerá un campo editable con una cadena de texto que representa los valores de la variable separados por punto y coma (;), agregue al final de esta cadena la expresión **;%C:\Program Files\Java\jdk1.6.0_25\bin**, (esta debe ser la ruta de los archivos binarios desempaquetados por JDK), luego acepte y finalice.

La siguiente imagen muestra el proceso de edición de la variable “path”.



Hasta aquí la instalación y configuración del JDK en Windows.

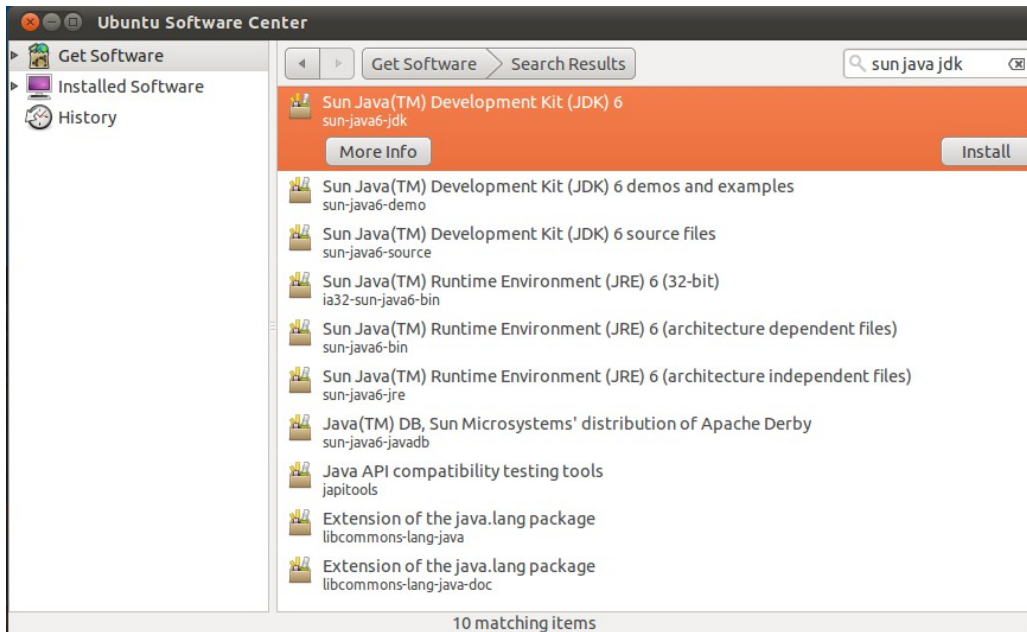
Instalación y configuración de Sun JDK para Ubuntu Linux

Los ejemplos aquí citados se realizaron bajo la versión 11.04 de Ubuntu Linux. Ubuntu y otras versiones de Linux incluyen por defecto, el OpenJDK.

OpenJDK es una implementación cien por ciento fuente abierta de lenguaje Java. Sun JDK es una implementación de lenguaje Java que en gran parte es de fuente abierta, pero sin embargo contiene algunos binarios que Oracle no ha “soltado” bajo licencia open source. A continuación, se explicará como instalar Sun JDK.

Abra la aplicación “Ubuntu Software Center”, en el área de búsqueda escriba “sun java jdk”, seguidamente se mostrará un grupo de resultados; seleccione la versión jdk que desea instalar. En el tiempo de escritura de este texto se encontraba disponible la versión 6.

La siguiente imagen muestra la interfaz gráfica de Ubuntu Software Center.



Seguidamente presione el botón “Install” para que se realice la descarga y se produzca la instalación del Sun JDK.

Como resultado, se crearán una serie de “folders” o “carpetas” con nombre distintivo para cada versión de Sun JDK, en este caso la versión exacta es la 1.6.0.24, la ruta de instalación es /usr/lib/jvm/java-6-sun-1.6.0.24/, esta ruta es importante ya que allí se encuentran las librerías que permiten compilar, ejecutar y depurar el código fuente Java.

Para terminar esta tarea solo resta configurar las variables de entorno para que las librerías de Sun JDK puedan ser accedidas desde cualquier ubicación en el sistema operativo, esto se hace agregando las siguientes líneas de texto al inicio del archivo /etc/profile (realice esta operación como usuario root):

```
##### Configuración de la variables de entorno JAVA #####
```

```
JAVA_HOME="/usr/lib/jvm/java-6-sun-1.6.0.24/"
```

```
JRE_HOME="/usr/lib/jvm/java-6-sun-1.6.0.24/jre/"
```

```
CLASSPATH="."
```

```
PATH=$PATH:/usr/lib/jvm/java-6-sun-1.6.0.24/
```

```
export JAVA_HOME
```

```
export CLASSPATH
```

```
export PATH
```

Luego de esto, lance una terminal o consola y digite la sentencia:

```
sudo update-alternatives --config java
```

La ejecución de la anterior sentencia, solicitará al usuario (root) seleccionar la versión de JDK que desea establecer en el sistema, como paso final seleccione la versión Sun JDK instalada.

Para probar que efectivamente la configuración se realizó con éxito, en la terminal, digite la siguiente sentencia y presione la tecla “enter”:

```
java -version
```

Si todo estuvo bien, el resultado será semejante a lo siguiente:

```
java version "1.6.0_24"  
Java(TM) SE Runtime Environment (build 1.6.0_24-b07)  
Java HotSpot(TM) 64-Bit Server VM (build 19.1-b02, mixed mode)
```

Hasta aquí la instalación y configuración del JDK en Ubuntu Linux.

Instalación de Eclipse IDE para Windows

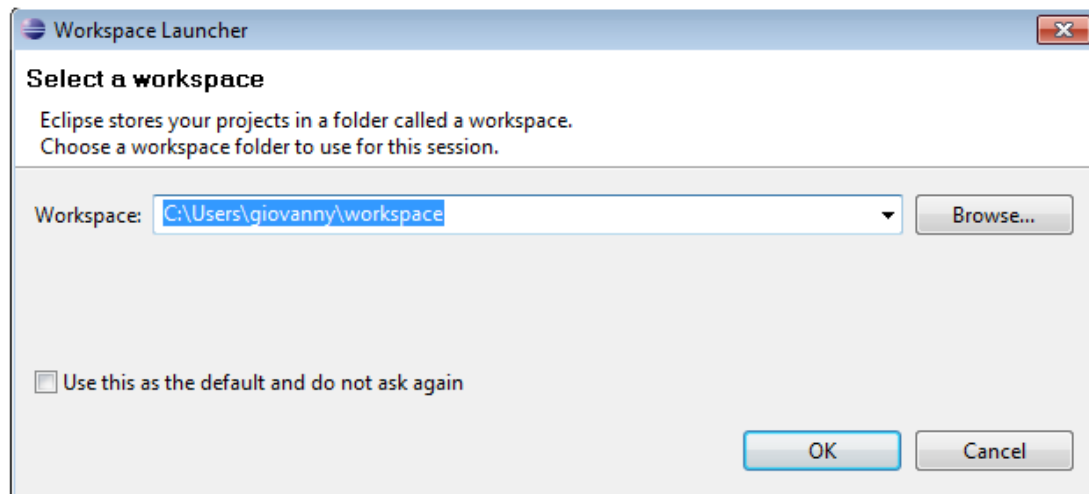
Antes de instalar Eclipse, es necesario primero instalar el Sun JDK, si aún no lo ha hecho, por favor realícelo siguiendo los pasos de la sección “Instalación y configuración de Sun JDK para Windows” en este mismo capítulo.

Después de instalar y configurar el Sun JDK, diríjase a la sección de descargas del sitio web oficial de Eclipse (<http://www.eclipse.org/downloads/>), busque la distribución “Eclipse IDE for Java Developers”, seleccione plataforma y arquitectura (Windows x86 o Windows x64) y proceda a descargarla.

Eclipse IDE viene empaquetado en un archivo “zip”, al descomprimir este archivo se producirá el folder “eclipse”, dentro de este folder, busque el archivo *eclipse.exe* y de doble click sobre este para iniciar Eclipse. Realice esta operación cada vez que desee iniciar Eclipse.

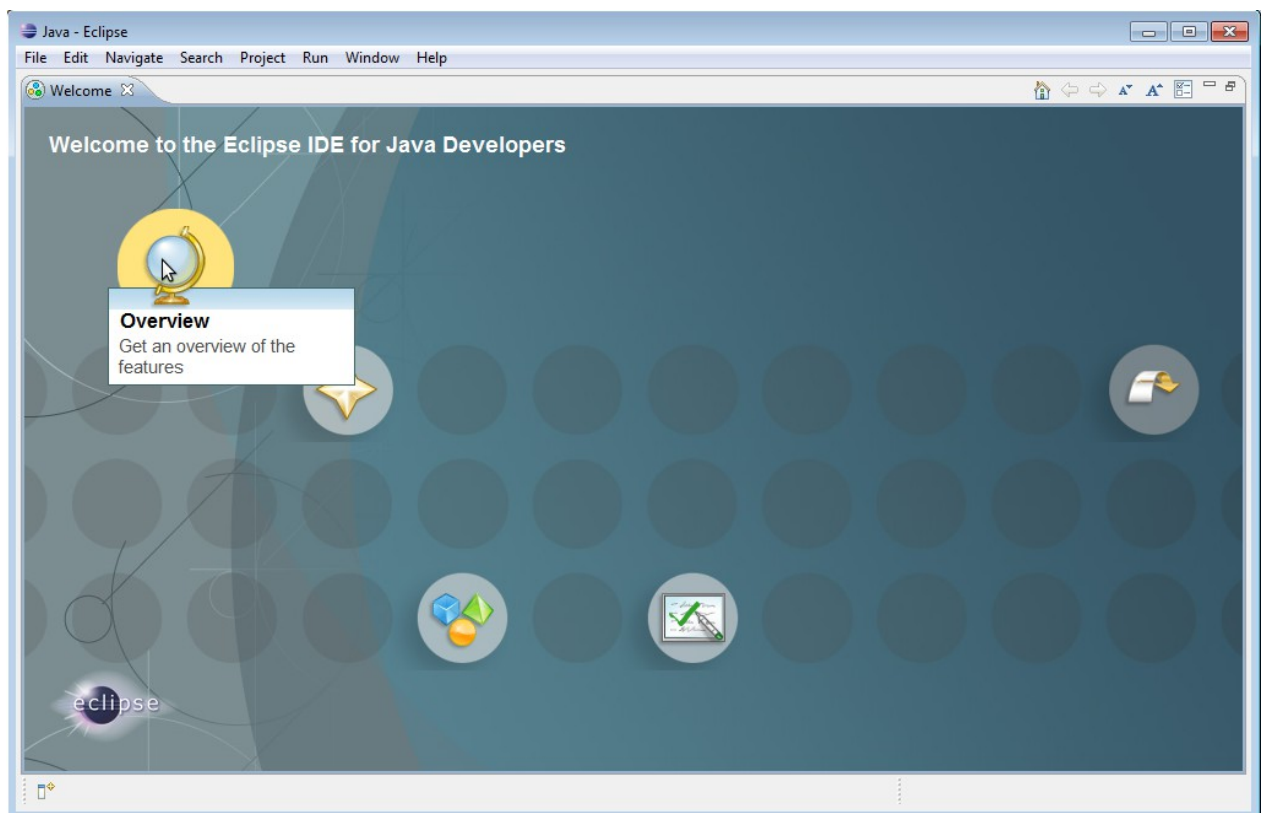
Lo primero que aparece al inicio de Eclipse, es un asistente que permite seleccionar la ubicación donde se almacenarán los proyectos. Seleccione una ubicación (con permisos para crear, leer y escribir archivos). Esta ubicación es conocida por Eclipse como “workspace”.

La siguiente imagen, muestra el asistente para definir el “workspace” o f6lder que almacenar6 los proyectos.



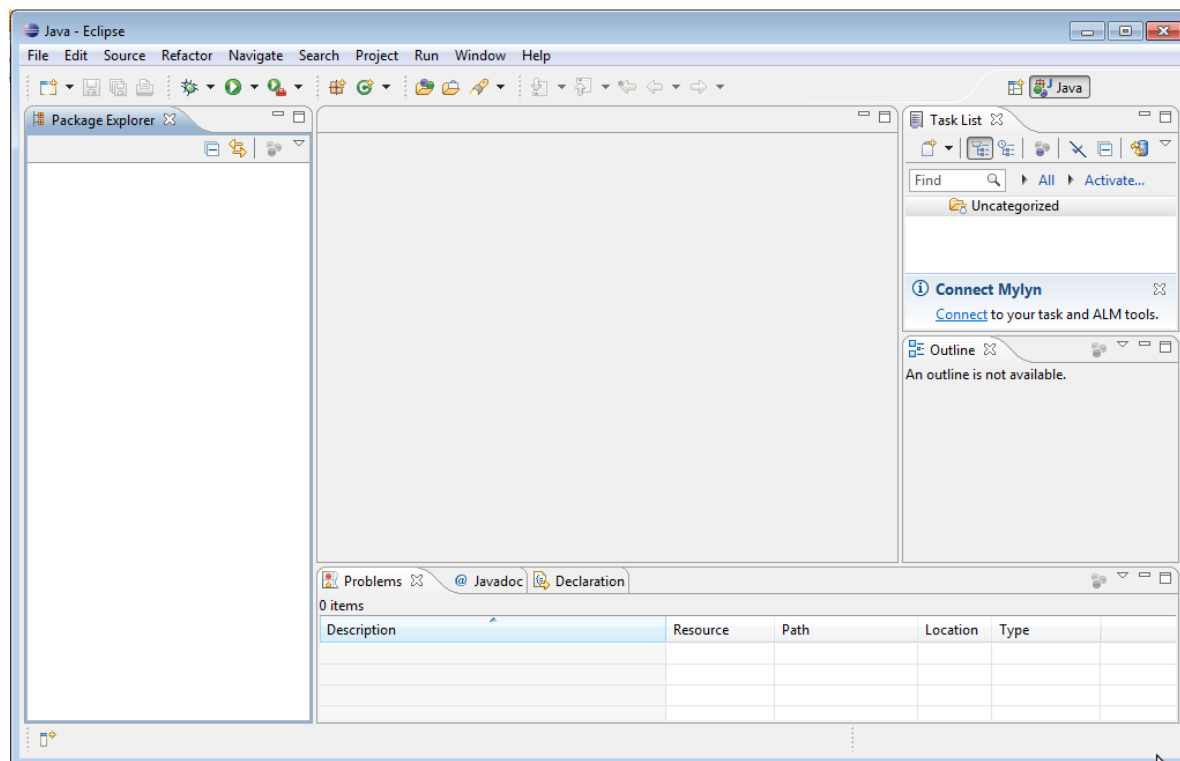
Cuando se realiza esta acci3n por primera vez, se mostrar6 una ficha con un conjunto de iconos de enlace a tutoriales que conforman un asistente para el aprendizaje de Eclipse. Si6ntase libre de visitar estos enlaces.

La siguiente imagen muestra la ficha de bienvenida de Eclipse.



Finalmente diríjase directamente al panel de control. Para realizar esto, de click en la “X” de la ficha de bienvenida.

La siguiente figura muestra el panel de control de Eclipse IDE que aparece luego de cerrar la ficha de bienvenida.



Hasta aquí la instalación de Eclipse IDE para Windows.

Instalación de Eclipse IDE para Ubuntu Linux

Antes de instalar Eclipse, es necesario instalar el Sun JDK, si aún no lo ha hecho, por favor realícelo siguiendo los pasos descritos en la sección “Instalación y configuración de Sun JDK para Ubuntu Linux” en este mismo capítulo.

Luego de instalar y configurar el Sun JDK, diríjase a la sección de descargas del sitio web oficial de Eclipse (<http://www.eclipse.org/downloads/>), busque la distribución “Eclipse IDE for Java Developers”, seleccione plataforma y arquitectura (Linux x86 o Linux x64) y realice la descarga.

Eclipse IDE viene empaquetado en un archivo “tar.gz”, para descomprimirlo, abra una terminal y ubíquese en la ruta donde se encuentra el archivo (por ejemplo `cd /home/FolderUsuario/Downloads`), estando allí ejecute la sentencia :

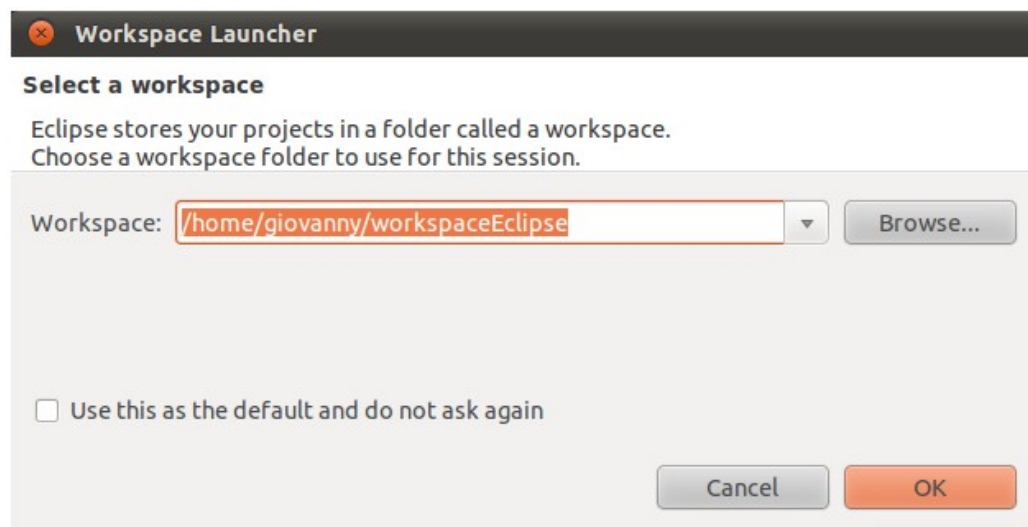
```
sudo tar -xvf nombreArchivoDescargado.tar.gz
```

La ejecución de esta sentencia producirá un folder de nombre “eclipse”, para arrancar Eclipse abra una “terminal”, ubíquese dentro del folder, luego ejecute la siguiente sentencia para iniciar Eclipse :

```
sudo eclipse
```

Lo primero que aparecerá al iniciar Eclipse es un asistente para seleccionar la ubicación donde se almacenarán los proyectos, asegúrese de seleccionar una ubicación con permisos para crear, leer y escribir archivos. Esta ubicación es conocida por Eclipse como “workspace”.

La siguiente captura de pantalla, muestra una imagen del asistente para definir el “workspace” o folder que almacenará los proyectos.



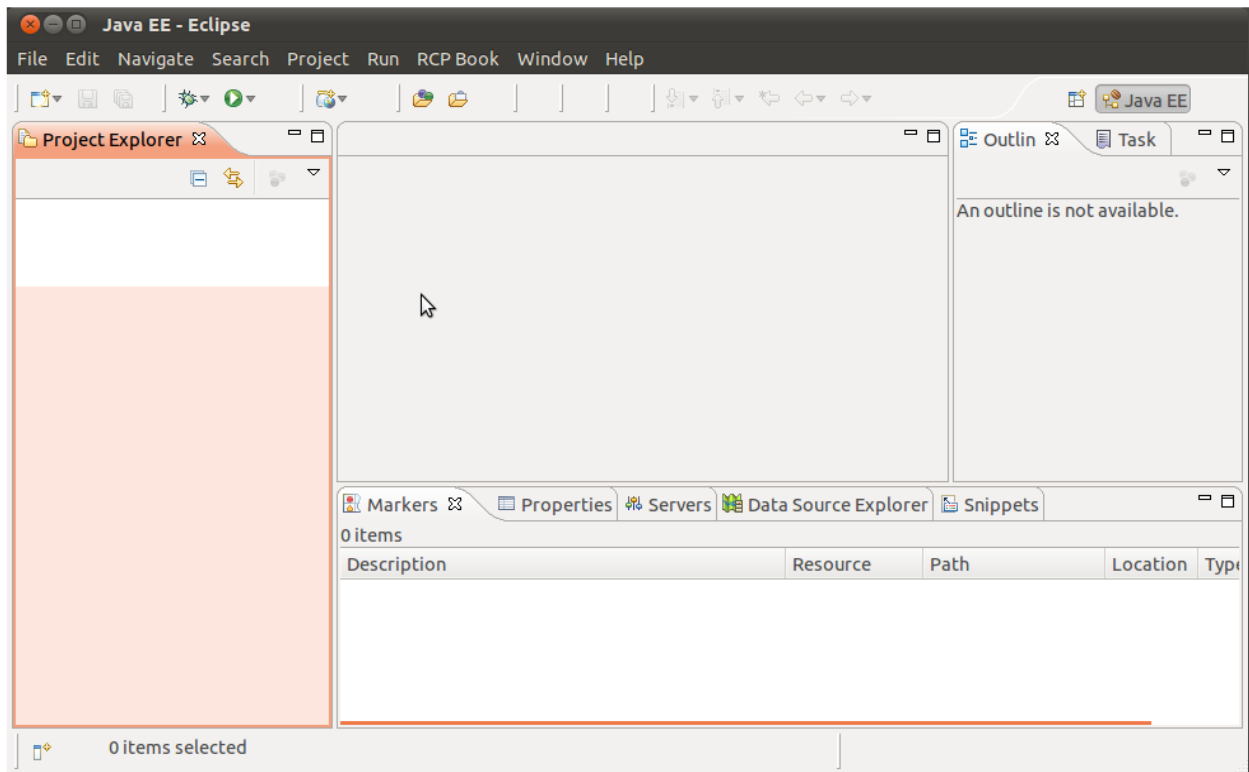
Cuando se realiza esta acción por primera vez, se mostrará una ficha con un conjunto de íconos de enlace que conforman un asistente de aprendizaje para Eclipse. Si lo desea puede visitar estos enlaces.

La siguiente captura de pantalla muestra la ficha de bienvenida de Eclipse.



Finalmente diríjase directamente al panel de control de Eclipse, para realizar esto de click en la "X" de la ficha de bienvenida.

La siguiente figura muestra el panel de control de Eclipse IDE que aparece luego de cerrar la ficha.

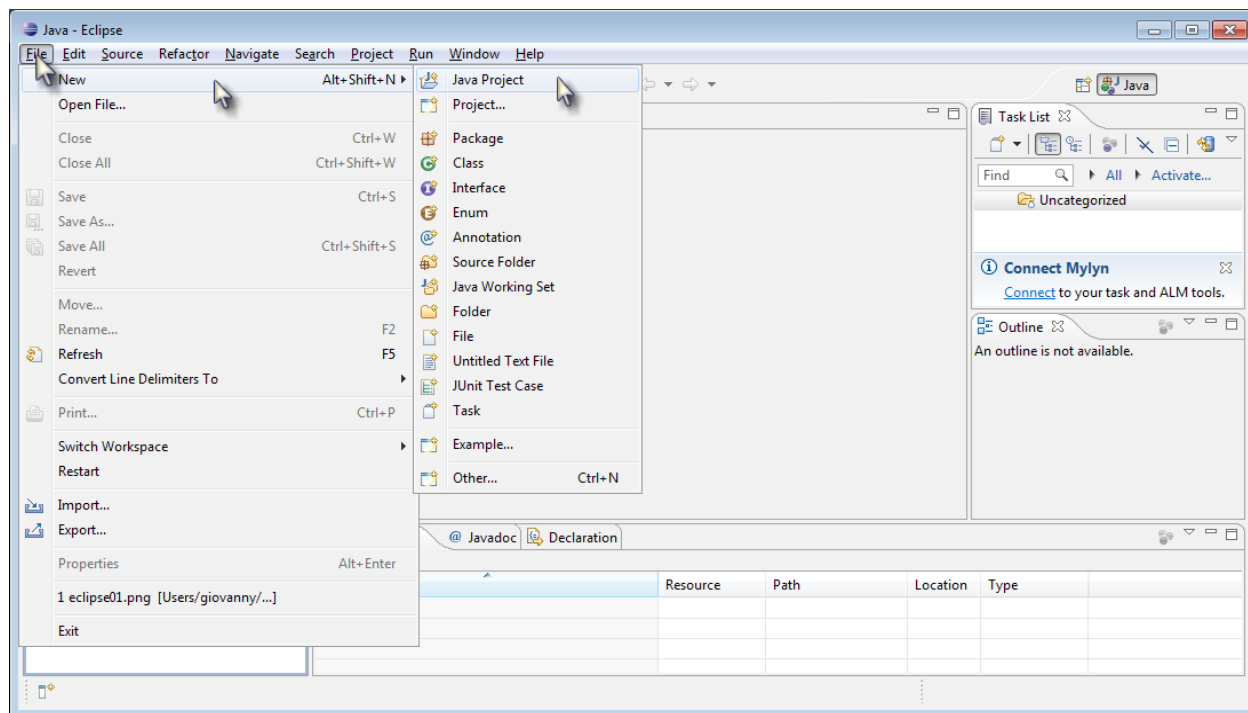


Hasta aquí la instalación de Eclipse IDE para Ubuntu Linux.

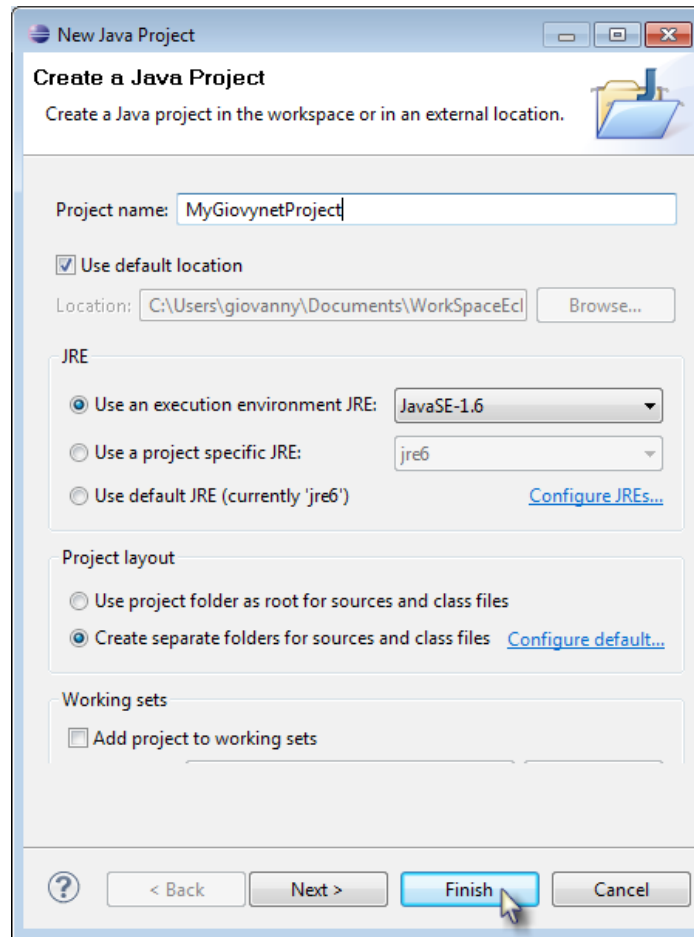
Capítulo 3. Creación de un proyecto Java con Giovynet Driver en Eclipse IDE

En el primer capítulo se describieron cinco componentes que conforman Giovynet Driver, de estos solo son necesarios tres para conformar un proyecto Java con Giovynet Driver, a continuación se describe como realizar esta tarea.

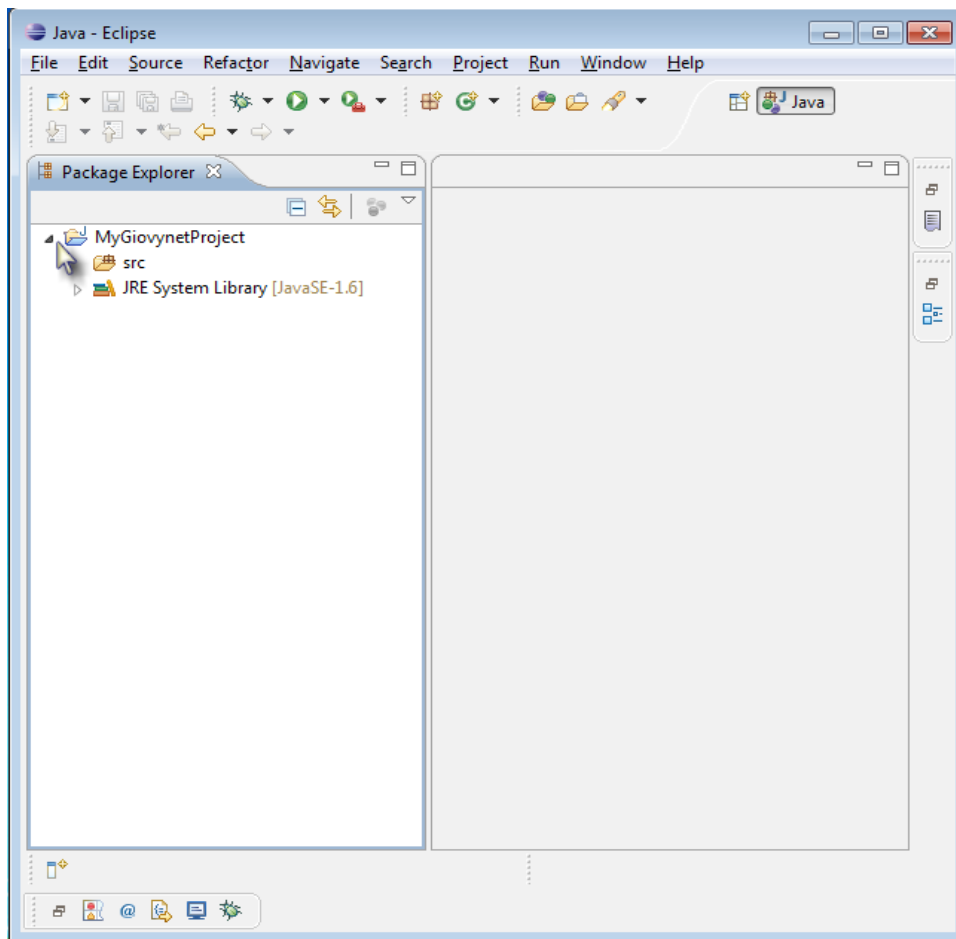
Luego de iniciar Eclipse, dirijase a la la parte superior izquierda de la barra de herramientas y de click en *File | New | Java Project*, así como se observa en la siguiente imagen:



En seguida aparecerá un asistente que creará el proyecto. Ingrese un nombre distintivo en el campo “Project Name” y de click en el botón “Finish”, como se observa a continuación:



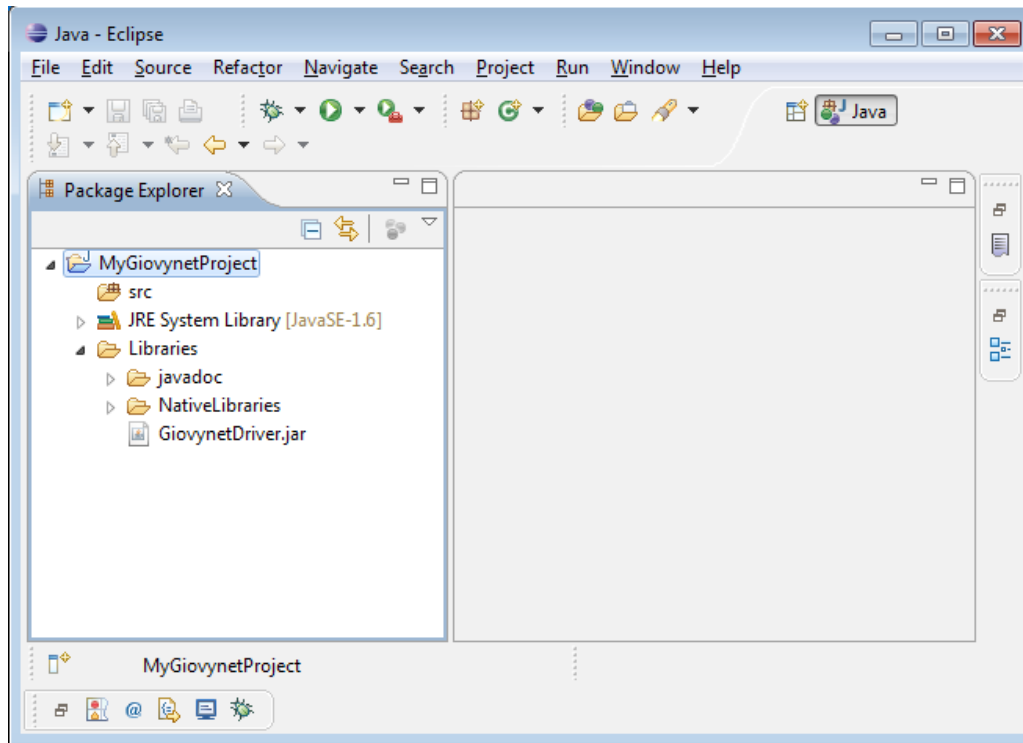
Luego de finalizar el asistente, aparecerá un de fólder con el nombre del proyecto en el área "Package Explorer". El proyecto se verá como sigue:



El siguiente paso consiste en crear dentro del proyecto un “folder” o “carpeta” de nombre *libraries* donde se agregarán los tres componentes de Giovynet Driver. Para hacer esto desde Eclipse, de click derecho en el fólder principal del proyecto y seleccione *New | Folder*, en seguida aparecerá un campo de texto donde debe digitar el nombre del fólder, digite la palabra *libraries* y finalice, después de esto se creará el fólder, dentro de este, copie y pegue los siguientes componentes :

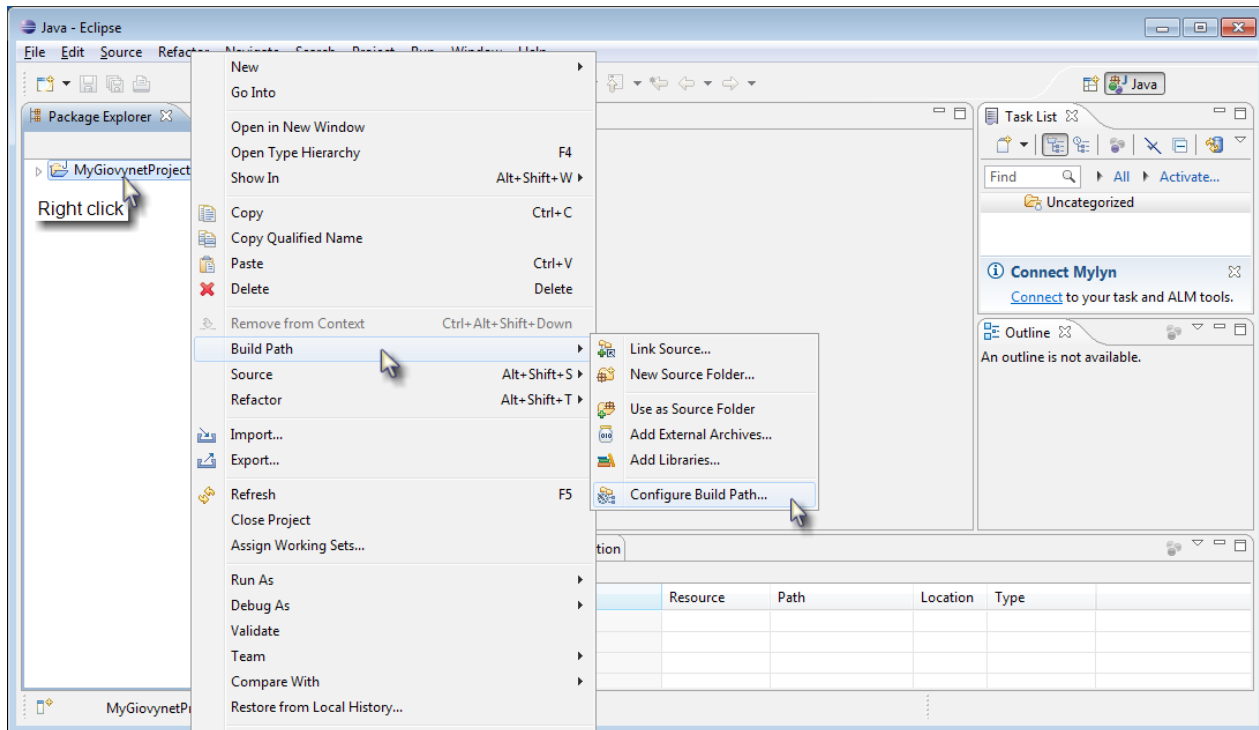
- GiovynetDriver.jar
- NativeLibraries
- javadoc

después de esto el proyecto se verá como muestra la siguiente imagen:

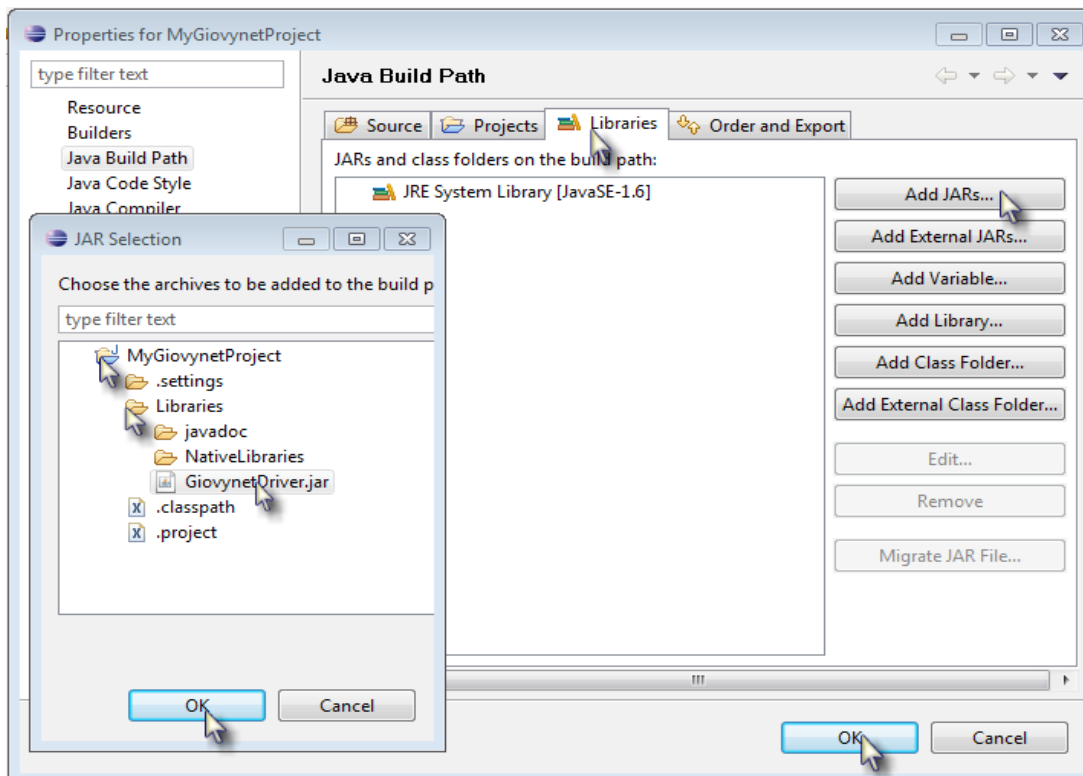


El siguiente paso consiste en vincular en el proyecto el archivo Java “GiovynetDriver.jar” con el fin de poder hacer uso de las rutinas de Giovynet Driver.

De click derecho sobre el nombre del proyecto (en el área “Package Explorer”), luego seleccione *Build Path | Configure Build Path*, así como sigue:

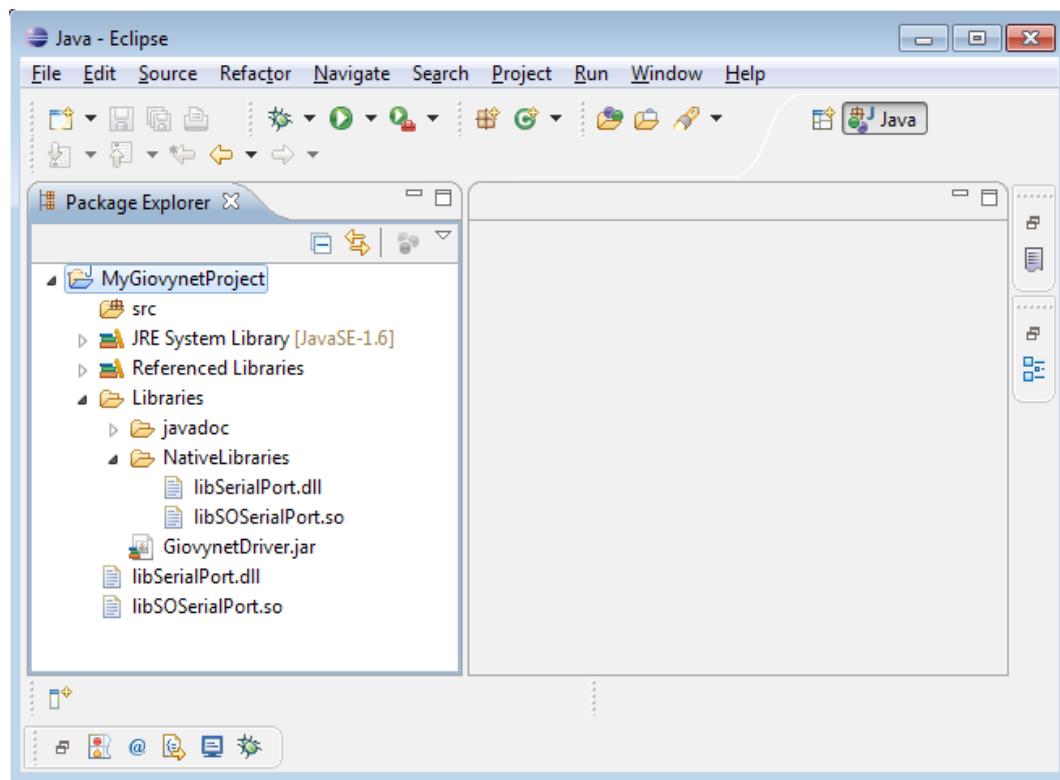


En seguida aparecerá la ventana de propiedades del proyecto, busque la sección “Libraries”, de click en el botón “Add JARs...” y seleccione el archivo “GiovynetDriver.jar”. Como ejemplo observe la siguiente imagen:



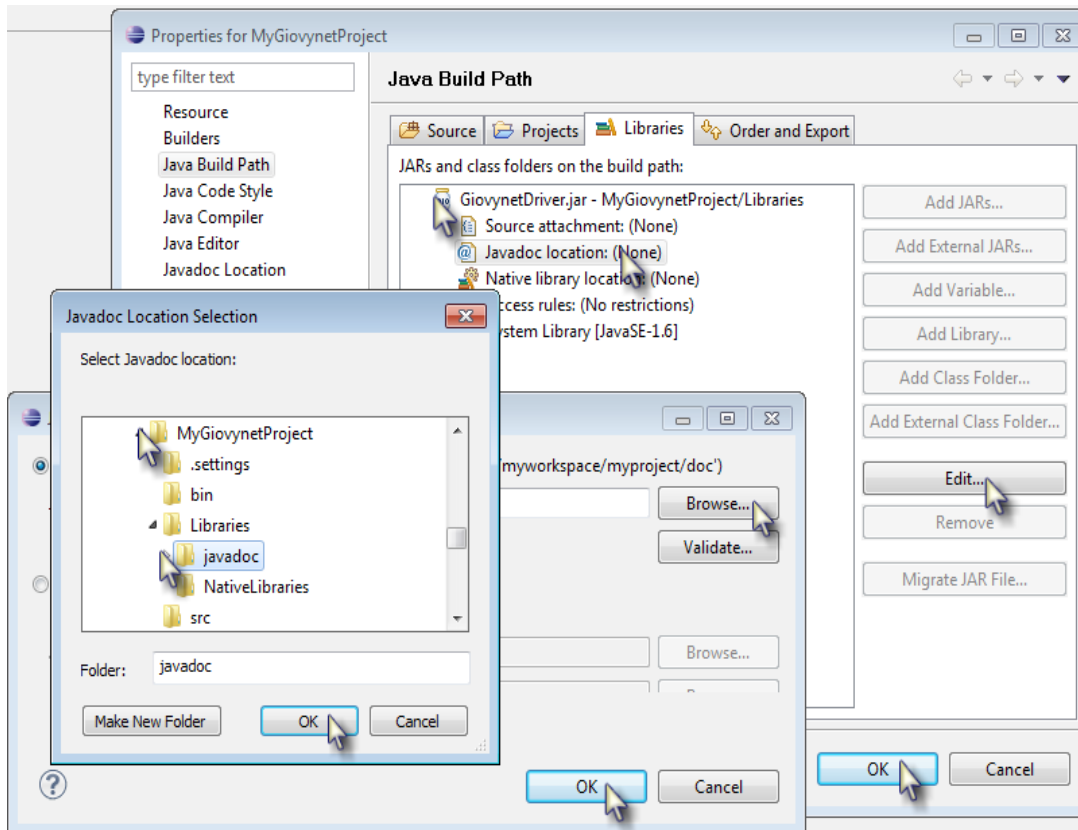
El paso final consiste en copiar y pegar dentro del f6lder principal del proyecto los archivos: *libSerialPort.dll* y *libSOSerialPort*. Estos se encuentran dentro del f6lder NativeLibraries. El prop6sito de estas librer6as es establecer comunicaci6n entre Java y el sistema operativo, para ver m6s informaci6n sobre esto dir6jase al tema “Arquitectura” en el primer cap6tulo de este libro.

Luego de realizar este paso ya se puede comenzar a construir una aplicaci6n Java con Giovynet Driver, la siguiente imagen muestra como se ver6 el proyecto:

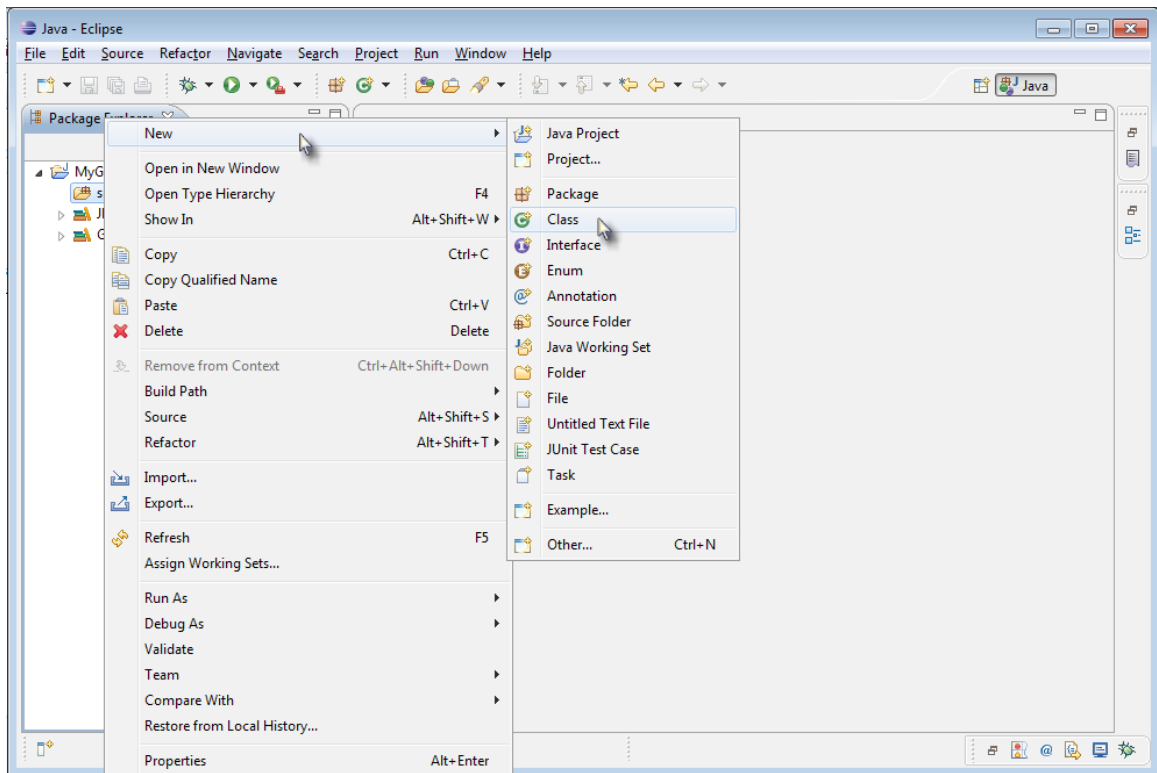


Existe un paso opcional pero recomendado, y consiste en referenciar en el proyecto la documentaci6n est6ndar de clases y m6todos que conforman Giovynet Driver, al hacer esto Eclipse podr6 mostrar informaci6n correspondiente a clases, m6todos y par6metros propios de Giovynet Driver en el momento de digitar c6digo fuente y por presionar al tiempo las teclas Ctrl y barra espaciadora. Esta caracter6stica puede ser de mucha ayuda para el programador.

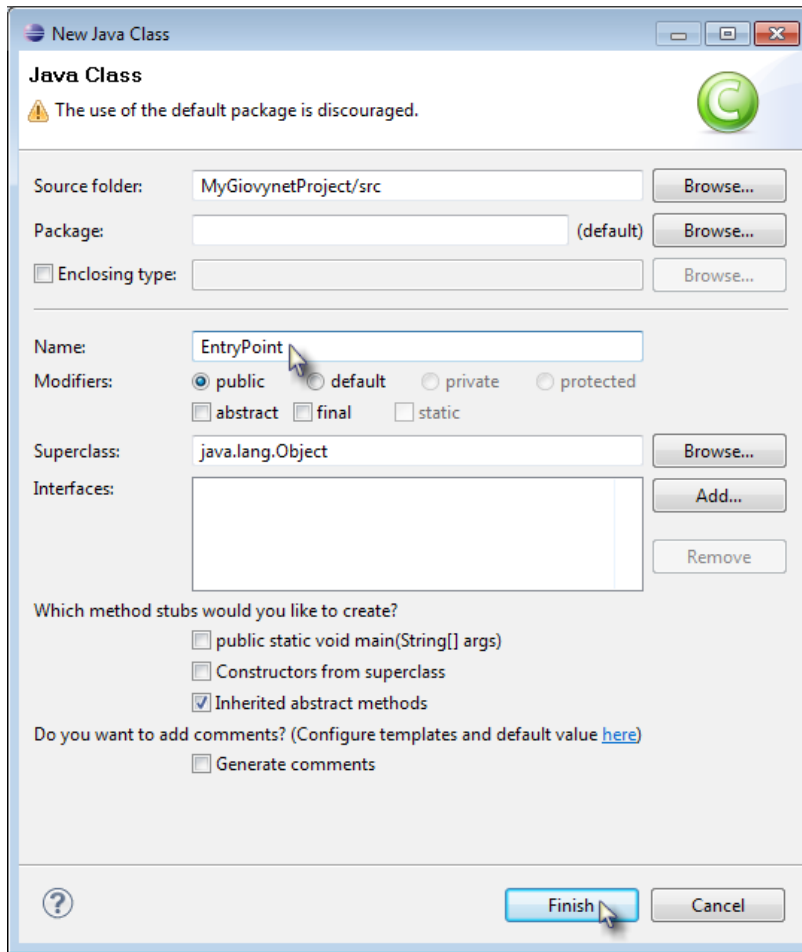
Para referenciar la documentaci6n de Giovynet Driver desde Eclipse, de click derecho en el f6lder principal del proyecto y seleccione *Build Path | Configure Build Path*, luego dir6jase a la secci6n “Libraries” y de click en el 6cono de GiovynetDriver.jar para desplegar el 6rbol de propiedades, en seguida de click en “Javadoc Location”, luego dir6jase al bot6n “Edit” y de click sobre este para lanzar el asistente que le permitir6 seleccionar el Javadoc de Giovynet Driver, as6 como lo muestra la siguiente imagen:



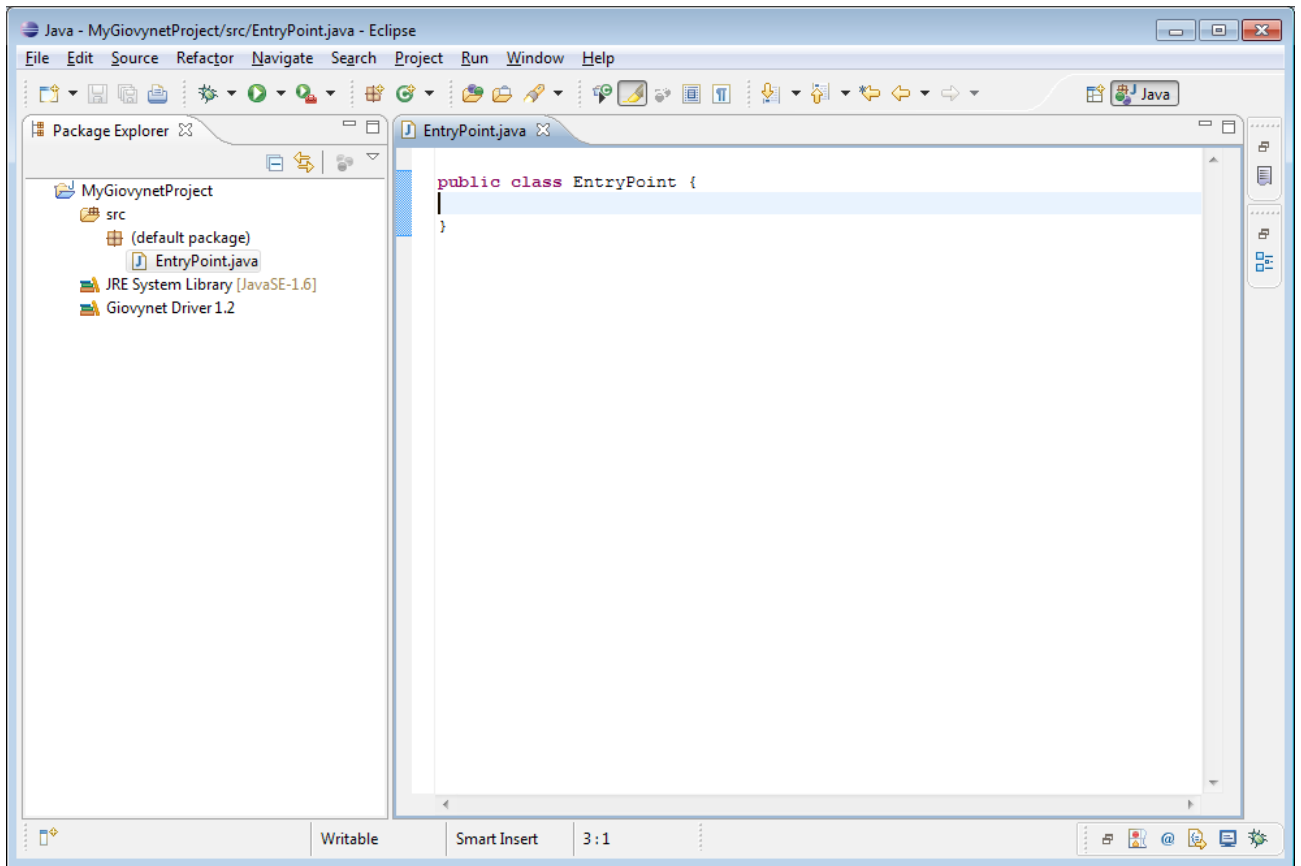
Ahora para probar el proyecto, se realizará una simple aplicación que mostrará los puertos series disponibles en el PC. Diríjase a la carpeta principal del proyecto, de click en ella para desplegar el árbol de subcarpetas, en seguida de click derecho sobre la carpeta "src", seleccione "New" | "Class", como se observa en la siguiente imagen:



Después de dar click en “Class” aparecerá un asistente que permite crear una clase. En el campo “Name” digite “EntryPoint”, luego de click en “Finish”. Observe la siguiente imagen para aclarar la idea:



Luego de dar click en el botón “Finish”, se creará la clase, y se mostrará el editor para comenzar a agregar código fuente Java, así como se muestra a continuación:

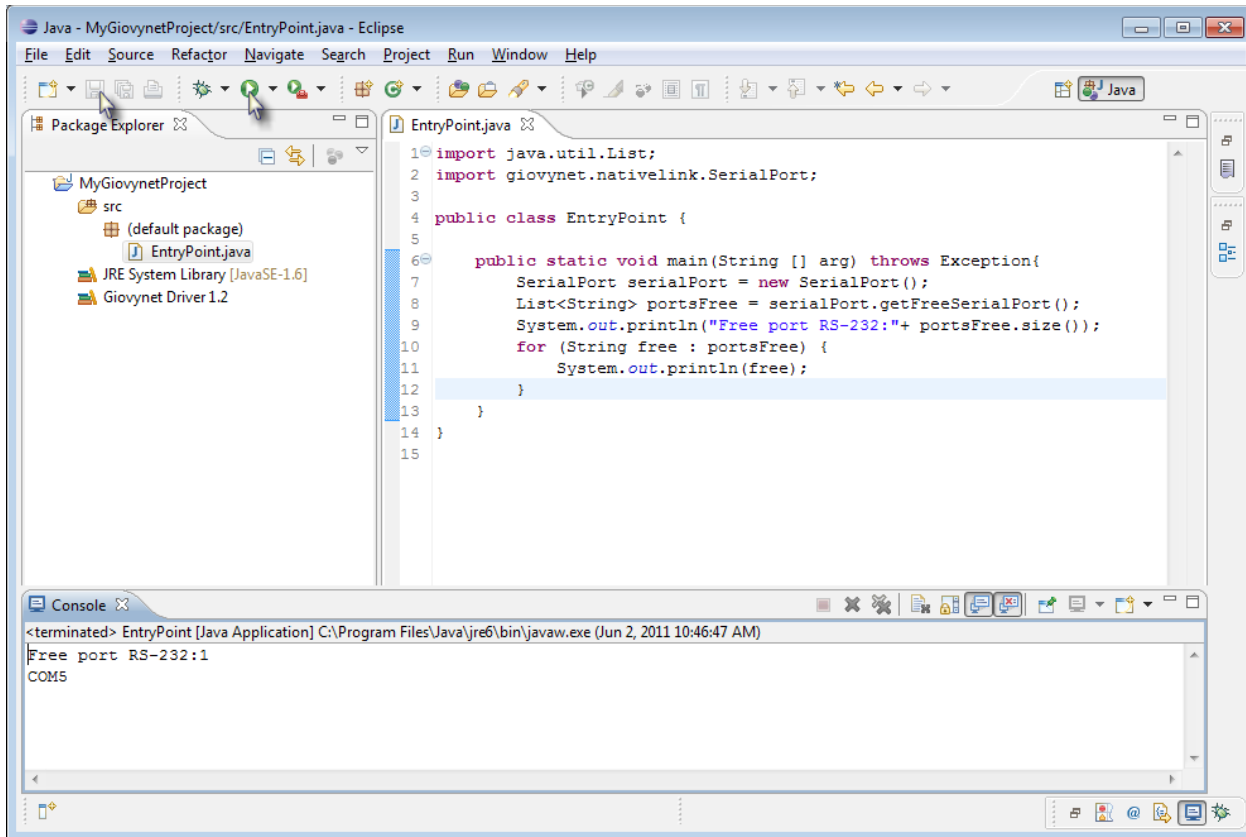


Escriba en el editor el siguiente código:

```
import java.util.List;  
import giovynt.nativelink.SerialPort;  
public class EntryPoint {  
    public static void main(String [] arg) throws Exception{  
        SerialPort serialPort = new SerialPort();  
        List<String> portsFree = serialPort.getFreeSerialPort();  
        System.out.println("Free RS-232 ports:");  
        for (String free : portsFree) {  
            System.out.println(free);  
        }  
    }  
}
```

Luego de esto se debe salvar o guardar la clase, lo puede hacer presionando las teclas “Ctrl” y “S”, o presione el botón en forma de “disquete” ubicado en la barra de herramientas.

Finalmente ejecute el código presionando al tiempo las teclas “Ctr” y “F6”, o si prefiere, diríjase a la barra de herramientas y presione el botón en forma de círculo verde con un triángulo en medio. Observe la siguiente imagen para reforzar el texto.



La ejecución de la anterior clase mostrará un mensaje en consola informando los puertos RS-232 disponibles para usar, en este caso el COM5 es el único puerto libre.

Sugerencia 1:

Si al ejecutar la aplicación se lanza el siguiente error:

java.lang.UnsatisfiedLinkError: Can't load library.

Esto se debe a que no se puede referenciar nativos (**libSerialPort.dll** y **libSOSerialPort.so**), porque no se encuentran en el folder del proyecto.

Para solucionar este error, copie los archivos **libSerialPort.dll** y **libSOSerialPort.so**, que se encuentran en el folder “NativeLibraries” y péguelos en el folder del proyecto.

Sugerencia 2:

Si al ejecutar la aplicación se lanza el error:

java.lang.UnsatisfiedLinkError: (Possible cause: architecture word width mismatch).

Se debe a que no se puede referenciar los archivos nativos (**libSerialPort.dll** y **libSOSerialPort.so**), porque estos están compilados en una arquitectura diferente a la arquitectura del PC. Para ilustrar una posible situación, suponga que se quiere realizar un desarrollo para una arquitectura x64 (64-bit), pero se usan archivos nativos compilados para arquitectura x86 (32-bit), debido a que las arquitecturas no son compatibles, entonces al intentar compilar se producirá este error.

La solución es reemplazar los archivos nativos, por archivos nativos de arquitectura correcta.

Capítulo 4. Puerto GIV8DAQ.

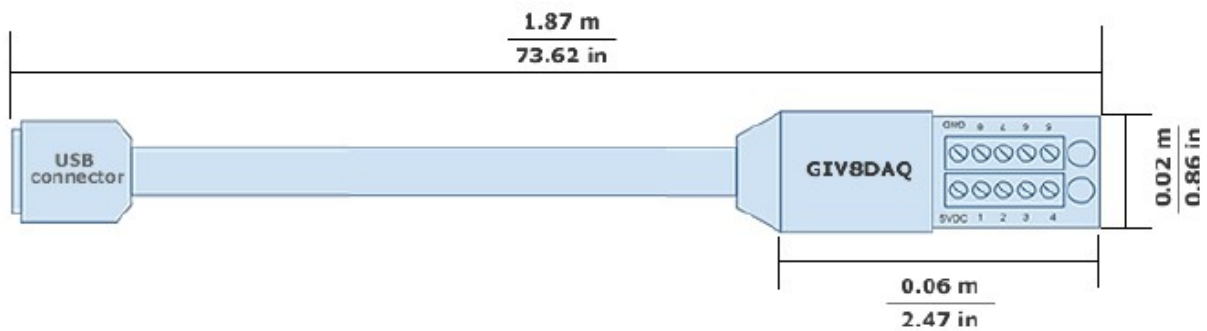
El puerto GIV8DAQ es una interfaz que se conecta al computador por USB. No necesita fuentes externas para funcionar debido a que toma la potencia del propio PC, por su tamaño y diseño se adapta fácilmente a circuitos externos.



Mediante el puerto GIV8DAQ, una aplicación Java puede realizar lo siguiente:

- Acoplar circuitos externos por medio de canales definidos programáticamente para funcionar como entradas o salidas digitales en lógica TTL (0-5VDC).
- Obtener medidas de voltaje análogo con resolución de 10 bits por canal, lo cual indica que para voltajes entre 0 y 5 V podría sensarse cambios en saltos de 50mV.
- Obtener medidas de temperatura por canal mediante previa conexión del sensor GIV12B20 (del cual se hablará más adelante).

Dimensiones



Especificaciones

GIV8DAQ es un sistema de 8 canales independientes, deriva su potencia del puerto USB del PC. Cada canal presenta las siguientes especificaciones:

- *Salida digital*: estado alto (5VDC) o estado bajo (0VDC), corriente hasta 25 mA.
- *Entrada digital*: "true" (5VDC) o "false", (0VDC).
- *Entrada análoga*: 10-bit A/D 0-5VDC.
- *Temperatura*: lectura usando el sensor GIV18B20 (comprado separadamente), en el rango de 67 a 257°F (-55 a 125°C).

Máximos rangos

Señales aplicadas por encima de los siguientes rangos pueden causar daños permanentes en el puerto:

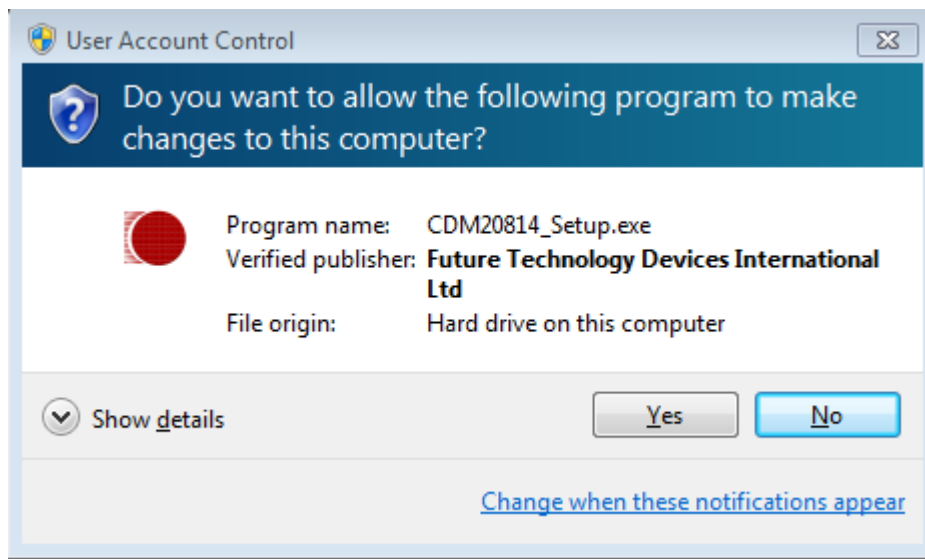
- Temperatura de funcionamiento: 0-70 °C.
- Tensión en E / S con respecto a tierra o "ground": -0,3 V a 5,3 V.
- Corriente para cualquier E / S: 25 mA.

Consideraciones importantes para Windows

Puede suceder que en el momento de conectar el puerto, Windows no lo reconozca. Esto se debe a que Windows no tiene instalado el "driver" para los puertos GIV. Por tanto se debe instalar manualmente dando dobleclick sobre el archivo **CDM208014_Setup.exe** que viene incluido en todas las distribuciones de "Giovynet_Driver versión 2.0". Este archivo es una aplicación segura y

verificada para Windows, es distribuida por la reconocida empresa de tecnología: *Future Technology Devices International Ltda.* Esta aplicación instala en Windows los “drivers” para que este sistema reconozca los puertos GIV. Esta situación no se presenta en Linux por que el “driver” para los puertos GIV viene incluido en el kernel, de tal manera que Linux reconoce estos dispositivos automaticamente.

La siguiente imagen muestra la ventana de información lanzada por Windows luego de dar dobleclick sobre el archivo **CDM208014_Setup.exe**.



Para comenzar la instalación presione “Yes” en seguida, se lanzará una consola informando el proceso de instalación, cuando finaliza se cierra automaticamente, esta acción es muy rapida y para algunos computadores imperceptible. Luego de la instalación Windows reconocerá sin problemas el puerto.

Programando el puerto GIV8DAQ con Eclipse IDE y Java

Antes de desarrollar aplicaciones con Java y Giovynet Driver son necesarias tres cosas: instalar el JDK , instalar Eclipse IDE y saber conformar un proyecto Java con Giovynet Driver. Si aún no ha instalado estas herramientas, dirijase al primer capítulo de este libro, y realice los pasos allí citados. Si desea saber como conformar un proyecto Java con Giovynet Driver, dirijase al capítulo 3.

El objetivo aquí es mostrar de manera fácil como manipular el GIV4R desde Java. De aquí en adelante y hasta que finalice el capítulo el lector se encontrará con una serie de secciones tituladas “¿Como saber?”, estas secciones explican de forma detallada cada una de las instrucciones del

GIV4R, también se muestra en cada sección sencillos trozos de código, listos para ser ejecutados desde una clase en un proyecto Java con Giovynet Driver.

¿Cómo saber cuantos dispositivos se pueden Instanciar?

Así como se señaló en el primer capítulo, cada distribución de Giovynet Driver permite manipular o instanciar una cantidad determinada de dispositivos entre puertos RS-232 y puertos GIV. Para saber programáticamente cuantos puertos puede instanciar una distribución, utilice el método **getNumDevicesAllowed()**, perteneciente a la clase estática **Info**. Este método devuelve un número entero que representa la cantidad de dispositivos permitidos. El siguiente ejemplo muestra en consola la cantidad de dispositivos permitidos:

```
public static void main (String [] arg) throws Exception {  
    System.out.println("Cantidad de dispositivos permitidos: "+ Info.getNumDevicesAllowed());  
}
```

¿Cómo saber si hay dispositivos GIV8DAQ conectados?

Lo primero es instanciar la clase **ScanDevices**, luego use el método **findGIV8DAQ()**, este método retorna una lista de objetos GIV8DAQ conectados al ordenador. Si el tamaño de la lista es cero, significa que no hay dispositivos conectados. El siguiente trozo de código ilustra esta idea:

```
public static void main (String [] arg) throws Exception{  
    ScanDevices sd = new ScanDevices();  
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();  
    if(listDevGIV8DAQ.size()==0){  
        System.out.println("No hay dispositivos GIV8DAQ conectados");  
    }else{  
        System.out.println("Hay: "+ listDevGIV8DAQ.size()+  
            "dispositivos GIV8DAQ conectados");  
    }  
}
```

¿Cómo obtener una instancia de los dispositivos GIV8DAQ conectados?

Para realizar esta tarea use el método **findGIV8DAQ()**, de la clase ScanDevices. Este método devuelve una lista de instancias u objetos de tipo GIV8DAQ, cada objeto GIV8DAQ representa un dispositivo GIV8DAQ, por lo tanto para referenciar una instancia de un dispositivo se utiliza el método **get(int index)** de la lista retornada por el método **findGIV8DAQ()**. El siguiente trozo de código muestra un ejemplo de esto:

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ oneDev = listDevGIV8DAQ.get(0);
        System.out.println("Se obtuvo la primera instancia de la lista");
    }
}
```

¿Cómo activar un puerto GIV8DAQ?

Cada puerto viene con una clave de activación, esta clave debe ser asignada programáticamente después de instanciar el puerto. Esto se hace con el comando **activate(String activationKey)** de la clase GIV8DAQ:

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    //Trae las instancias de los GIV8DAQ conectados
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        giv8daq.activate("#####-##");//Activación del puerto
        System.out.println("Se activo el puerto GIV8DAQ");
    }
}
```

Si se intenta ejecutar un comando sin activar el puerto se lanzará la siguiente excepción:

giovinet.permissions.PermissionsException: The device is inactive. To activate it, first you must use the method: activate (String activation_Key);

¿Cómo saber cual es el puerto de comunicaciones asociado?

Cada puerto GIV tiene asociado un único puerto de comunicaciones **Com**, para saber cual es el puerto asociado a un GIV8DAQ, se utiliza el método **getCom()** de la clase **GIV8DAQ**, este método retorna un objeto de la clase **Com**. La clase **Com** presenta el método **getPort()**, el cual retorna el nombre del puerto. El siguiente trozo de código muestra un ejemplo de esto:

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    //Trae las instancias de los GIV8DAQ conectados
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        //Activación del puerto
        giv8daq.activate("18820-01");
        //Obtención del nombre del COM
        String strCom = giv8daq.getCom().getPort();
        System.out.println("El puerto asociado con el GIV8DAQ es: "
            +strCom);
    }
}
```

¿Cómo establecer un canal en alto (5VDC)?

El dispositivo GIV8DAQ presenta ocho canales configurables de forma independiente para varios usos. Para configurar un canal como salida digital en “alto” (5VDC), se utiliza el método **command(Channel ch, DigitalOut instruction)**. Este método presenta dos parámetros , el primer parámetro es un atributo estático tipo “enum” llamado **Channel** de la clase **GIV8DAQ**. **Channel** presenta ocho variables estáticas que inician con prefijo “ch[1-8]” y representan cada uno de los ocho canales del puerto. El segundo parámetro es un atributo estático (enum) de la clase **GIV8DAQ** llamado **DigitalOut**. **DigitalOut** presenta la variable estática **set_High**, la cual establece el canal seleccionado en “alto” (5VDC). El siguiente trozo de código es utilizado para establecer en “alto” (5VDC), el canal cuatro.

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
```

```

    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        giv8daq.activate("18820-01");//Activacion del puerto
        //Comando para establecer el canal 4 en alto (5VDC)
        giv8daq.command(GIV8DAQ.Channel.ch4,GIV8DAQ.DigitalOutput.set_High);
        System.out.println("Ahora el canal 4 mide 5VDC.");
    }
}

```

¿Cómo establecer un canal en “bajo” o “falso” (0VDC)?

El puerto GIV8DAQ presenta ocho canales configurables para varios propósitos. Para configurar un canal como salida digital en “bajo” (0VDC), se utiliza el método **command(Channel ch, DigitalOut instruction)**. Este método presenta dos parámetros, el primer parámetro es un atributo estático (enum) de la clase **GIV8DAQ** llamado **Channel**. **Channel** presenta ocho variables estáticas que inician con prefijo “ch[1-8]” y representan cada uno de los ocho canales del puerto.

El segundo parámetro es un atributo estático (enum) de la clase **GIV8DAQ** llamado **DigitalOutput**. **DigitalOutput** presenta la variable estática **set_Low**, la cual establece el canal seleccionado (primer parámetro) en “bajo” (0VDC). El siguiente trozo de código es utilizado para establecer en bajo (0VDC), el canal cuatro.

```

public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        giv8daq.activate("18820-01");//Activacion del puerto
        //Comando para establecer el canal 4 en bajo (0VDC)
        giv8daq.command(GIV8DAQ.Channel.ch4,GIV8DAQ.DigitalOutput.set_Low);
        System.out.println("Ahora el canal 4 mide 5VDC.");
    }
}

```

¿Cómo obtener el valor digital (alto o bajo) de un canal canal?

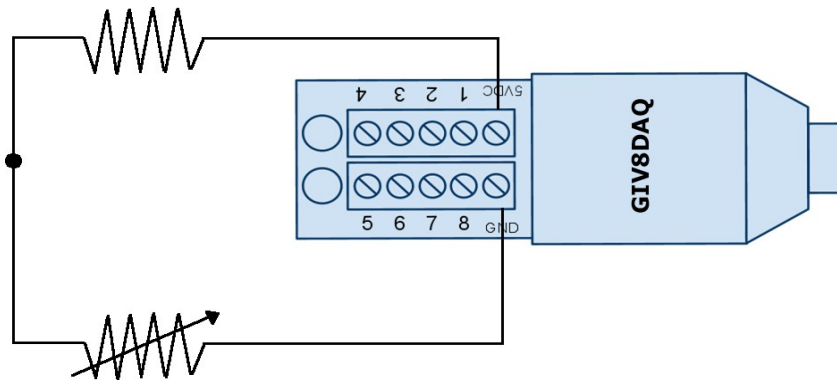
Para obtener el estado lógico de un canal (“alto” o “bajo”) sometido a un potencial TTL (0 - 5VDC), se utiliza el método **command(Channel ch, DigitalInput instruction)**. Este método presenta dos parámetros, el primer parámetro es un atributo estático (enum) de la clase **GIV8DAQ** llamado **Channel**. **Channel** presenta ocho variables estáticas que inician con prefijo “ch[1-8]”, estas representan cada uno de los ocho canales del puerto. El segundo parámetro es un atributo (enum) de la clase **GIV8DAQ** llamado **DigitalInput**, este presenta la variable estática **get_Digital_Input**. Esta variable indica que se obtendrá el valor lógico del canal. El siguiente trozo de código es utilizado para obtener el estado lógico del canal uno.

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        //Activacion del puerto
        giv8daq.activate("18820-01");
        //Comando que obtiene el estado lógico del canal 1
        Boolean statusCH1 =
        giv8daq.command(GIV8DAQ.Channel.ch1,GIV8DAQ.DigitalInput.get_Digital_Input);
        System.out.println("El estado logico del canal 1 es : "+statusCH1);
    }
}
```

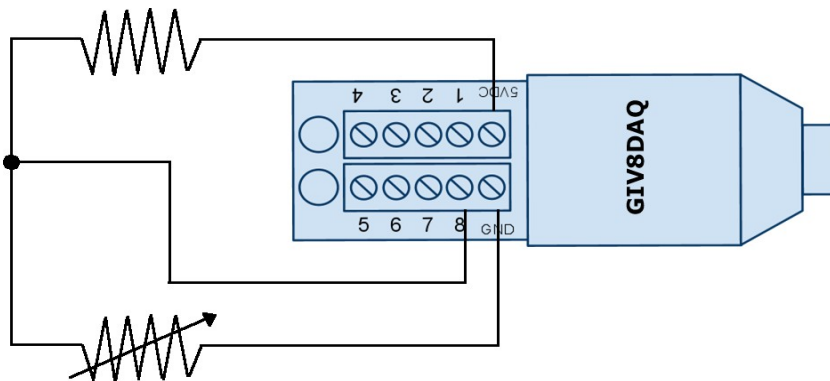
Si los bornes del canal 1 están sometidos a un potencial de 5VDC, el estado sera “true”, si los bornes del canal 1 están sometidos a un potencial de 0VDC el estado será “false”.

¿Cómo obtener lectura de voltaje de un canal?

Cada canal de GIV8DAQ puede leer el voltajes. Para ilustrar este hecho con un ejemplo, suponga que se tiene un circuito serie compuesto por una resistencia fija y una resistencia variable, un extremo del circuito se conecta al borne GND y el otro extremo se conecta al borne 5V, como se ilustra en la siguiente figura.



Suponga que se requiere leer el voltaje de la resistencia variable. Para hacer esto se conecta el punto de unión entre las resistencias a un canal del GIV8DAQ, luego programáticamente se obtiene el valor de voltaje en dicho canal. La siguiente figura ilustra este ejemplo utilizando el canal 8.



Para obtener el valor de voltaje (en voltios) en un canal, se utiliza el método **command(Channel ch, Analog instruction)**, que retorna el valor en formato "String". Este método presenta dos parámetros, el primer parámetro es un atributo estático (enum) de la clase **GIV8DAQ** llamado **Channel**. **Channel** presenta ocho variables estáticas que inician con prefijo "ch[1-8]", estas representan cada uno de los ocho canales del dispositivo.

El segundo parámetro es otro atributo tipo "enum" de la clase **GIV8DAQ** llamado **Analog**. **Analog** presenta la variable estática **get_Voltaje_Measurement**. Esta variable indica que se obtendrá el valor de voltaje presente en el canal seleccionado (primer parámetro). El siguiente trozo de código se usa para obtener el valor de voltaje presente en el canal ocho.

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        //Activacion del puerto
        giv8daq.activate("18820-01");
    }
}
```



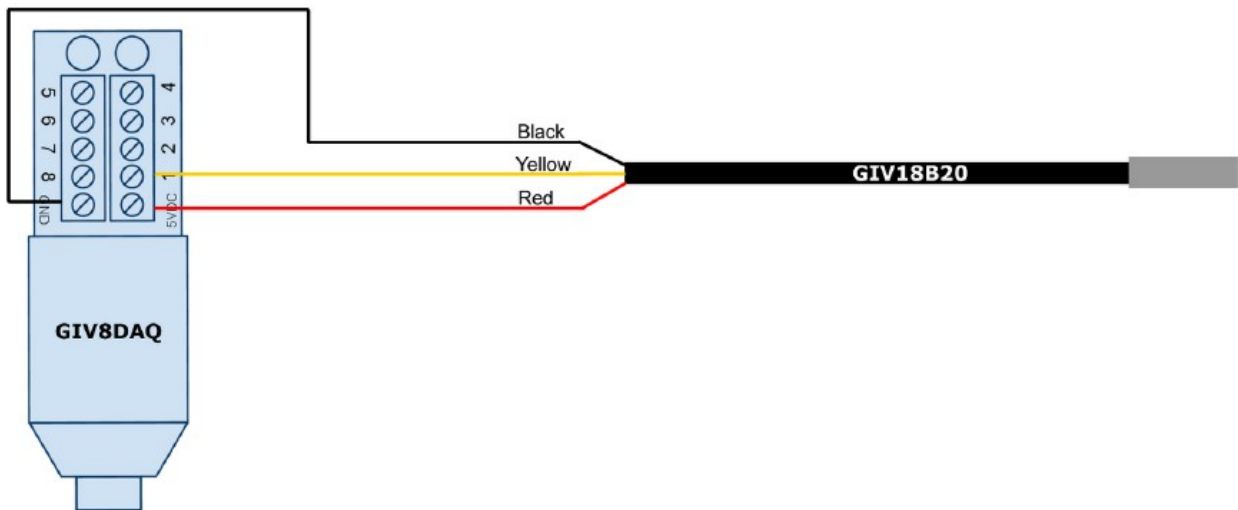
```

//Comando que lee el voltage en el canal 8
String voltageCH8 =
giv8daq.command(GIV8DAQ.Channel.ch8,GIV8DAQ.Analog.get_Voltage_Measurement);
System.out.println("El voltaje en el canal 8 es: "+ voltageCH8);
}
}

```

¿Cómo conectar el sensor de temperatura GIV18B20 en un canal?

A cada canal del GIV8DAQ se puede conectar un sensor de temperatura GIV18B20, para hacer esto realice los siguientes pasos: conecte el cable rojo al borne 5V, conecte el cable negro al borne GND, y finalmente conecte el cable amarillo al canal deseado. Para reforzar la idea, a continuación se muestra una imagen que ilustra la conexión de un sensor GIV18B20 al canal uno de un puerto GIV8DAQ.



El sensor GIV18B20 posee una sonda resistente al agua y de acero inoxidable, presenta una longitud de 118 pulgadas (3 mts), es capaz de suministrar medidas en el rango de 67 a 257°F (-55 a 125°C). Para ver más datos técnicos acerca de este módulo en dirijase a Giovynet.com.

¿Cómo obtener lectura de temperatura?

Para obtener la lectura de temperatura, de un sensor GIV18B20, conectado a un canal, se utiliza la sentencia, **command(Channel ch, Analog instruction)** que

retorna el valor en formato “String”. Este método presenta dos parámetros, el primer parámetro es un atributo estático (enum) de la clase **GIV8DAQ** llamado **Channel**.

Channel presenta ocho variables estáticas que inician con prefijo “ch[1-8]”, estas representan cada uno de los ocho canales del dispositivo.

El segundo parámetro es un atributo estático (enum) de la clase **GIV8DAQ** llamado **Analog**.

Analog presenta la variable estática **get_Temperature_Measurement**. Esta variable indica que se obtendrá el valor de temperatura para el canal seleccionado (primer parámetro).

Se debe tener en cuenta que el primer dato leído será de 999.99°, las siguientes lecturas devolverán datos validos.

El siguiente trozo de código muestra cuatro lecturas de temperatura que retorna un sensor GIV18B20 conectado al canal tres en un puerto GIV8DAQ.

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        //Activacion del puerto
        giv8daq.activate("18820-01");
        String valueTemp ="";
        //Comando que obtiene la temperatura de un sensor GIV12B20
        //conectado al canal 3.
        //La primera lectura se descarta
        giv8daq.command(GIV8DAQ.Channel.ch3,GIV8DAQ.Analog.get_Temperature_Measurement);
        for (int i = 0; i < 4; i++){
            Thread.sleep(1000);//Tiempo de espera de 1 segundo
            valueTemp =
            giv8daq.command(GIV8DAQ.Channel.ch3,GIV8DAQ.Analog.get_Temperature_Measurement);
            System.out.println(i+" Dato de Temperatura :"+ valueTemp);
        }
        System.out.println("La temperatura registrada en el canal uno es: "+ valueTemp);
        System.out.println("Tiempo establecido para lectura de respuesta:"+ giv8daq.getTimeInquiryCH3());
    }
}
```

¿Cómo establecer el modo de lectura de temperatura en escala Fahrenheit o Celsius?

El formato de lectura por defecto para todos los canales es Fahrenheit, sin embargo es posible cambiar el formato de lectura a Celsius. Para establecer el modo de lectura se utiliza el método: **command(TemperaturaReadingMode instruction)**, el cual dispone del parámetro

TemperaturaReadingMode. Este parámetro es un atributo estático (enum) de la clase **GIV8DAQ**, que presenta las variables estáticas **set_Celsius** y **set_Fahrenheit**, estas se utilizan para establecer el modo de lectura en Celsius o Fahrenheit . Por ejemplo el siguiente trozo de código muestra como establecer el modo de lectura en Celsius:

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        //Activacion del puerto
        giv8daq.activate("18820-01");
        GIV8DAQ giv8dq = listDevGIV8DAQ.get(0);
        //Establece el formato de lectura en
        //Celsius para todos los canales.
        giv8dq.command(GIV8DAQ.TemperaturaReadingMode.set_Celsius);
        String valueTemp ="";
        //Comando que obtiene la temperatura de un sensor GIV12B20
        //conectado al canal 3.
        valueTemp =
        giv8dq.command(GIV8DAQ.Channel.ch3,GIV8DAQ.Analog.get_Temperature_Measurement);
        System.out.println("La temperatura registrada en el canal uno es: "+ valueTemp);
    }
}
```

Consideraciones importantes en lecturas de temperatura

Cuando se ejecuta un comando para lectura de temperatura o voltaje, Giovynet Driver internamente envía una consulta hacia el puerto GIV8DAQ, este recibe y procesa la consulta, y guarda el resultado en un area de memoria, luego de un tiempo de 100 milisegundos (establecido por defecto), Giovynet Driver lee el area de memoria y retorna la respuesta a Java. No es usual pero puede suceder que el tiempo de procesamiento y recepción supere el tiempo de espera de lectura establecido. Esto pasa sobre todo para lecturas de temperatura, debido a que se suma el tiempo de procesamiento del GIV8DAQ con el tiempo de procesamiento del sensor GIV18B20. Si esto sucede el dato la lectura obtenido será errado, fuera de rango tal como 999.99°, por tanto se recomienda que programáticamente se descarten los datos leídos fuera de los rangos 67 a 257°F (-55 a 125°C), que son los rangos de lectura posibles para el GIV18B20. Por otro lado también existe la posibilidad

programáticamente de ampliar los tiempos de procesamiento y recepción para cada canal, usando el método:

setTimeInquiryCH[1-8](int tiempoMilisegundos); de la clase GIV8DAQ.

Por ejemplo el siguiente trozo de código establece a 500 milisegundos el tiempo de lectura de respuesta para el canal tres.

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        //Activacion del puerto
        giv8daq.activate("18820-01");
        //Comando para establecer el formato de lectura en Celcius
        //para todos los canales.
        giv8daq.command(GIV8DAQ.TemperaturaReadingMode.set_Celsius);
        //Comando para establecer 500mS como el tiempo
        //de lectura de respuesta en el canal 3.
        giv8daq.setTimeInquiryCH3(500);
        String valueTemp ="";
        //Comando que obtiene la temperatura de un sensor GIV12B20
        //conectado al canal 3.
        valueTemp = giv8daq.command(GIV8DAQ.Channel.ch3,GIV8DAQ.Analog.get_Temperature_Measurement);
        System.out.println("La temperatura registrada en el canal uno es: "+ valueTemp);
        System.out.println("Tiempo establecido para lectura de respuesta: "+
            giv8daq.getTimeInquiryCH3());
    }
}
```

¿Cómo saber la cantidad de puertos GIV8DAQ instanciados?

Ya que no todas las distribuciones de Giovynet Driver pueden instanciar la misma cantidad de dispositivos², puede ser necesario programáticamente saber la cantidad actual de instancias de GIV8DAQ. Para esto se utiliza el método **getNumberOfInstancesGIV8DAQ()**, de la clase estática **Info**.

```
public static void main (String [] arg) throws Exception {
```

² Refiérase al capítulo 1, Distribuciones y Licencias.

```

        System.out.println("Cantidad de GIV8DAQ actualmente instanciados: "+
Info.getNumberOfInstancesGIV8DAQ();
    }

```

¿Cómo finalizar una instancia de un puerto GIV8DAQ?

Para finalizar una instancia de GIV8DAQ, use el método **close()** de la clase GIV8DAQ. En el siguiente ejemplo finaliza la primera instancia, devuelta por el método findGIV8DAQ().

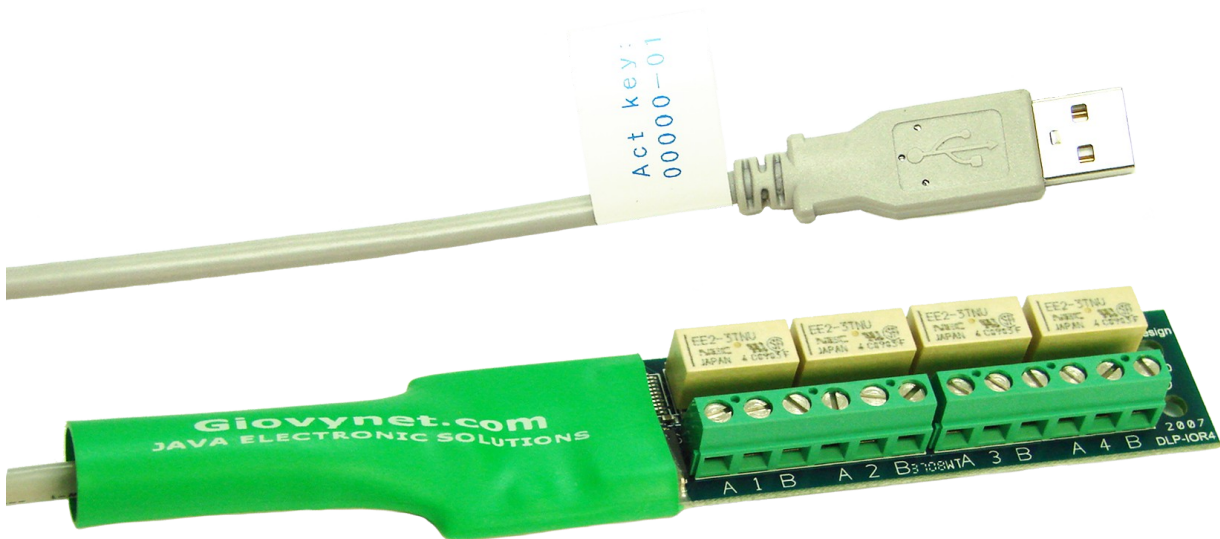
```

public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
    if(listDevGIV8DAQ.size()==0){
        System.out.println("No hay dispositivos GIV8DAQ conectados");
    }else{
        GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
        System.out.println("GIV8DAQ instanciados: "+
            Info.getNumberOfInstancesGIV8DAQ());
        //Finaliza la instancia giv8daq
        giv8daq.close();
        System.out.println("GIV8DAQ instanciados (después de cerrar): "+
            Info.getNumberOfInstancesGIV8DAQ());
    }
}

```

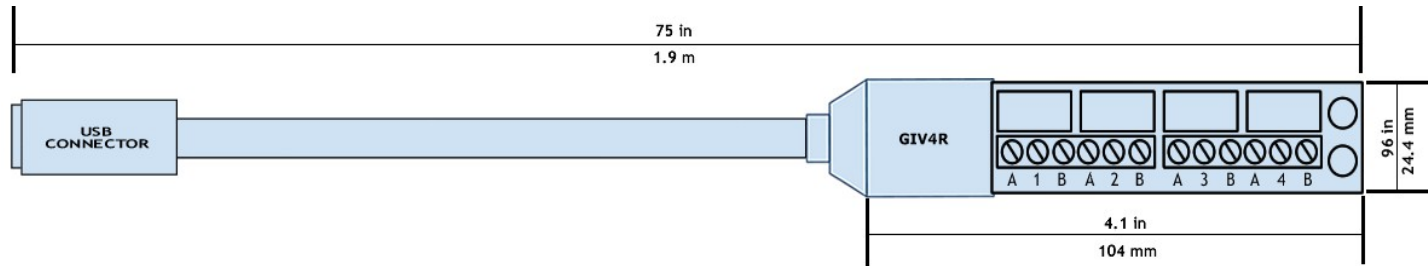
Capítulo 5. Puerto GIV4R

El puerto GIV4R es una interfaz que se conecta al computador por USB. No necesita fuentes externas para funcionar debido a que toma la potencia del propio PC, Se adapta fácilmente a circuitos externos por su tamaño y diseño. Consta de cuatro relevos, cada relevo presenta tres bornes o terminales de conexión, el borne del centro representa la terminal común del relevo y esta marcado con un número de 1 a 4. Cada terminal común puede ser conectado programáticamente con el borne de la izquierda marcado como “A” o con el borne de la derecha marcado como “B”. La longitud del cable de conexión es de 73 pulgadas (1.8 mts)



Cada relé es independiente en su funcionamiento y mantiene su estado en caso de desconexión o apagado del PC. EL puerto GIV4R es pensado para usuarios que desean activar o desactivar circuitos de mayor potencia que la del propio PC.

Dimensiones



Especificaciones

El puerto USB GIV4R es un dispositivo de 4 relevos independientes, deriva su potencia del del PC. Cada relevo presenta las siguientes especificaciones:

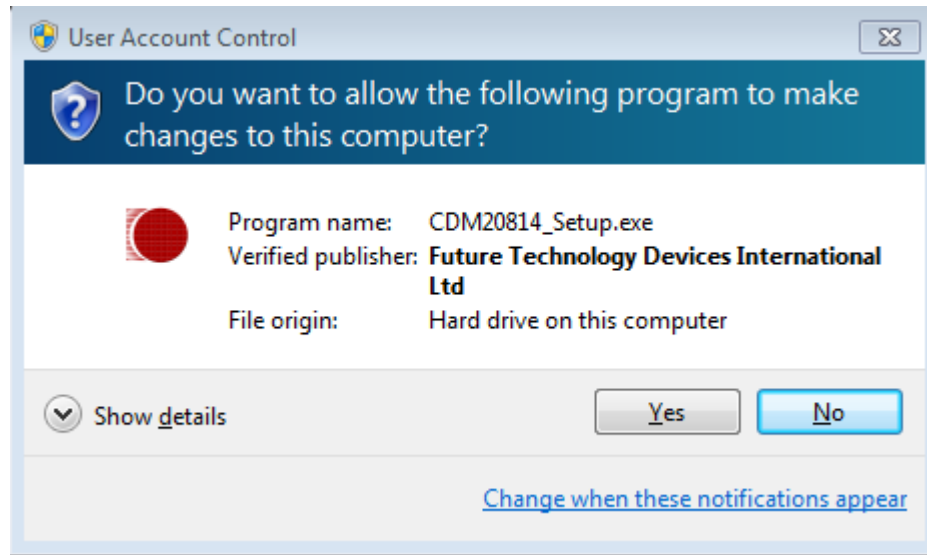
- Valores de contacto: 60W, 125VA
- Máxima conmutación voltaje: 220 VDC, 250 VCA
- Máxima corriente de conmutación: 2A
- Máxima corriente de carga de: 2A
- Temperatura de funcionamiento: 0-70 ° C

Someter el puerto a valores mayores o fuera de rango de los señalados arriba puede causar daños permanentes en este.

Consideraciones importantes para Windows

Puede suceder que en el momento de conectar el puerto, Windows no lo reconozca. Esto se debe a que Windows no tiene instalado el “driver” para los puertos GIV. Por tanto se debe instalar manualmente dando dobleclick sobre el archivo **CDM208014_Setup.exe** que viene incluido en todas las distribuciones de “Giovynet_Driver versión 2.0”. Este archivo es una aplicación segura y verificada para Windows, es distribuida por la reconocida empresa de tecnología: *Future Technology Devices International Ltda.* Esta aplicación instala en Windows los “drivers” para que este sistema reconozca los puertos GIV. Esta situación no se presenta en Linux por que el “driver” para los puertos GIV viene incluido en el kernel, de tal manera que Linux reconoce estos dispositivos automáticamente.

La siguiente imagen muestra la ventana de información lanzada por Windows luego de dar dobleclick sobre el archivo **CDM208014_Setup.exe**.



Para comenzar la instalación presione “Yes”, en seguida se lanzará una consola informando el proceso de instalación, cuando finaliza se cierra automáticamente, esta acción es muy rápida y para algunos computadores imperceptible. Luego de la instalación Windows reconocerá sin problemas el puerto.

Programando el puerto GIV4R con Eclipse IDE y Java

Antes de desarrollar aplicaciones con Java y Giovynet Driver son necesarias tres cosas: instalar el JDK , instalar Eclipse IDE y saber conformar un proyecto Java con Giovynet Driver. Si aún no ha instalado estas herramientas, diríjirse al primer capítulo de este libro, y realice los pasos allí citados. Si desea saber como conformar un proyecto Java con Giovynet Driver, diríjase al capítulo 3.

El objetivo aquí es mostrar de manera fácil como manipular el GIV8DAQ desde Java. De aquí en adelante y hasta que finalice este capítulo el lector se encontrará con una serie de secciones tituladas “¿Como saber? “, estas secciones explican de forma detallada cada una de las instrucciones del GIV4R, también se muestran para cada sección sencillos trozos de código, listos para se ejecutados desde una clase en un proyecto Java con Giovynet Driver.

¿Cómo saber cuantos dispositivos se pueden instanciar?

Así como se señaló en el primer capítulo, cada distribución de Giovynet Driver permite manipular o instanciar una cantidad determinada de dispositivos entre puertos RS-232 y puertos GIV. Para saber programáticamente cuantos puertos puede instanciar una distribución, utilice el método

getNumDevicesAllowed(), perteneciente a la clase estática **Info**. Este método devuelve un número entero que representa la cantidad de dispositivos permitidos. El siguiente ejemplo muestra en consola la cantidad de dispositivos permitidos:

```
public static void main (String [] arg) throws Exception {  
    System.out.println("Cantidad de dispositivos permitidos: "+ Info.getNumDevicesAllowed());  
}
```

¿Cómo saber si hay dispositivos GIV4R conectados?

Lo primero es instanciar la clase **ScanDevices**, luego use el método **findGIV4R()**, este método retorna una lista de objetos GIV4R conectados al ordenador. Si el tamaño de la lista es cero, significa que no hay dispositivos conectados. El siguiente trozo de código es una demostración de lo anterior:

```
public static void main (String [] arg) throws Exception {  
    ScanDevices sd = new ScanDevices();  
    List<GIV4R> listDevGIV4R = sd.findGIV4R();  
    if(listDevGIV4R.size()==0){  
        System.out.println("No hay dispositivos GIV4R conectados");  
    }else{  
        System.out.println("Hay: "+ listDevGIV4R.size()+  
            " dispositivos GIV4R conectados");  
    }  
}
```

¿Cómo obtener una instancia de los dispositivos GIV4R conectados?

Para realizar esta tarea use el método **findGIV4R()**, de la clase **ScanDevices**. Este método devuelve una lista de instancias u objetos de tipo GIV4R, cada objeto GIV4R representa un puerto GIV4R, por lo tanto para referenciar una instancia de un dispositivo se utiliza el método **get(int**

index) de la lista retornada por el método **findGIV4R()**. El siguiente trozo de código muestra un ejemplo de esto:

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV4R> listDevGIV4R =sd.findGIV4R();
    if(listDevGIV4R.size()==0){
        System.out.println("No hay dispositivos GIV4R conectados");
    }else{
        GIV4R oneDev = listDevGIV4R.get(0);
        System.out.println("Se obtuvo la primera instancia de la lista");
    }
}
```

¿Cómo activar un puerto GIV4R?

Cada puerto viene con una clave de activación, esta clave debe ser asignada programáticamente después de instanciar el puerto. Esto se hace con el comando **activate(String activationKey)** de la clase GIV4R:

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV4R> listDevGIV4R =sd.findGIV4R();
    if(listDevGIV4R.size()==0){
        System.out.println("No hay dispositivos GIV4R conectados");
    }else{
        GIV4R giv4r = listDevGIV4R.get(0);
        //Activación del puerto
        giv4r.activate("#####-##");
        System.out.println("Se activó el primer puerto GIV4R.");
    }
}
```

Si se intenta ejecutar un comando sin activar el puerto se lanzará la siguiente excepción:

giovinet.permissions.PermissionsException: The device is inactive. To activate it, first you must use the method: activate (String activation_Key);

¿Cómo saber cual es el puerto de comunicaciones asociado?

Cada puerto GIV tiene asociado un único puerto de comunicaciones Com, para saber cual es el puerto asociado a un GIV4R, se utiliza el método **getCom()** de la clase **GIV4R**, este método retorna un objeto de la clase **Com**, este representa el puerto de comunicaciones. La clase **Com** presenta el método **getPort()**, el cual retorna el nombre del puerto. El siguiente trozo de código muestra un ejemplo de esto:

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV4R> listDevGIV4R =sd.findGIV4R();
    if(listDevGIV4R.size()==0){
        System.out.println("No hay dispositivos GIV4R conectados");
    }else{
        GIV4R giv4r = listDevGIV4R.get(0);
        //Activación del puerto
        giv4r.activate("#####-##");
        System.out.println("Se activó el primer puerto GIV4R.");
        //Obtención del nombre del COM
        String strCom = giv4r.getCom().getPort();
        System.out.println("El puerto asociado con el GIV4R es: "
            +strCom);
    }
}
```

¿Cómo conectar el terminal común de un relévo con el borne A?

Cada relevo presenta tres bornes, el borne del centro representa la conexión común que puede ser conectada programáticamente con el borne “A” (izquierda) o con el borne “B” (derecha).

Para conectar el terminal común de un relevo con el borne “A” (izquierda), se usa el método **command(Relay relay, Connec.to_A)** de la clase **GIV4R**. El primer parámetro representa el relevo y su terminal común, el segundo parámetro representa a donde se conectará esta terminal (borne “A” o borne “B”). El siguiente trozo de código demuestra como conectar el terminal común del relevo “1” con el borne “A”.

```
public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV4R> listDevGIV4R =sd.findGIV4R();
    if(listDevGIV4R.size()==0){
        System.out.println("No hay dispositivos GIV4R conectados");
    }else{
        GIV4R giv4r = listDevGIV4R.get(0);
        //Activación del puerto
```

```

        giv4r.activate("18820-01");
        System.out.println("Se activó el primer puerto GIV4R.");
        giv4r.command(GIV4R.Relay_1,GIV4R.Connect.to_A);
        System.out.println("Se conectó el relévo 1 con el borne A.");
    }
}

```

¿Cómo conectar el terminal común de un relévo con el borne B?

Para conectar el terminal común de un relevo con el borne “B” (izquierda), se usa el método **command(Relay relay, Connec.to_B)** de la clase GIV4R. El primer parámetro representa el relevo y su terminal común, el segundo parámetro representa donde se conectará (borne A o borne B). El siguiente trozo de código demuestra como conectar el terminal común del relévo “4” con el borne “B”.

```

public static void main (String [] arg) throws Exception {
    ScanDevices sd = new ScanDevices();
    List<GIV4R> listDevGIV4R =sd.findGIV4R();
    if(listDevGIV4R.size()==0){
        System.out.println("No hay dispositivos GIV4R conectados");
    }else{
        GIV4R giv4r = listDevGIV4R.get(0);
        //Activación del puerto
        giv4r.activate("18820-01");
        System.out.println("Se activó el primer puerto GIV4R.");
        giv4r.command(GIV4R.Relay_4,GIV4R.Connect.to_B);
        System.out.println("Se conectó el relévo 4 con el borne B.");
    }
}

```

¿Cómo saber la cantidad de puertos GIV4R instanciados?

Ya que no todas las distribuciones de Giovynet Driver pueden instanciar la misma cantidad de dispositivos³, puede ser necesario programáticamente obtener la cantidad actual de instancias de

³ Refiérase al capítulo uno sección Distribuciones y Licencias.

GIV8DAQ. Para esto se utiliza el método **getNumberOfInstancesGIV8DAQ()**, de la clase estática **Info**.

```
public static void main (String [] arg) throws Exception {  
    System.out.println("Cantidad de GIV8DAQ actualmente instanciados: "+  
        Info.getNumberOfInstancesGIV8DAQ());  
}
```

¿Cómo finalizar una instancia de un puerto GIV4R?

Para finalizar una instancia de GIV4R, use el método **close()** de la clase GIV4R. En el siguiente ejemplo se finaliza la primera instancia, retornada por el método findGIV4R().

```
public static void main (String [] arg) throws Exception {  
    ScanDevices sd = new ScanDevices();  
    List<GIV4R> listDevGIV4R =sd.findGIV4R();  
    if(listDevGIV4R.size()==0){  
        System.out.println("No hay dispositivos GIV4R conectados");  
    }else{  
        GIV4R giv4r = listDevGIV4R.get(0);  
        System.out.println("GIV4R instanciados: "+  
            Info.getNumberOfInstancesGIV4R());  
        //Finaliza la instancia giv4r  
        giv4r.close();  
        System.out.println("GIV4R instanciados (después de cerrar): "+  
            Info.getNumberOfInstancesGIV4R());  
    }  
}
```

Capítulo 6. Envío y recepción de caracteres ASCII a través del puerto serie RS-232

Este capítulo consta de una serie de secciones tituladas “¿Cómo saber?” , estas secciones explican de forma detallada cada una de las instrucciones usadas para enviar y recibir caracteres ASCII a través del puerto serie RS-232, para esto se utilizan sencillos trozos de código, listos para ejecutarse en una clase Java.

¿Cómo saber que puertos serie hay libres?

Primero se instancia un objeto de tipo **giovynet.nativelink.SerialPort**, seguidamente se utiliza el método **getFreeSerialPort()** para obtener una lista String de puertos libres:

```
public static void main(String[] args)throws Exception{  
  
    SerialPort serialPort = new SerialPort();  
    List<String> portsFree = serialPort.getFreeSerialPort();  
    for (String free : portsFree) {  
        System.out.println(free);  
    }  
}
```

¿Cómo configurar el puerto serie?

Primero se crea un objeto de tipo **giovynet.serial.Parameter**, el cual presenta por "default" la siguiente configuración:

- port = COM1
- baudRate = 9600
- byteSize = 8
- stopBits = 1
- parity = N (no)

Para trabajar con la anterior configuración, solo resta instanciar un objeto de tipo **giovynt.serial.Com**, pasandole como constructor el objeto "parameter". de esta manera se abre el puerto serie "COM1" y queda listo para trabajar.

```
Parameters parameter = new Parameters();
```

```
Com com = new Com(parameter);
```

Para cambiar los parámetros por "default", se utilizan los métodos **set** del objeto **parameter**, antes de instanciar la clase **giovynt.serial.Com**. Por ejemplo; para configurar el puerto COM2 a una velocidad de 460800 bps, se utilizan las siguientes instrucciones:

```
Parameters param = new Parameters();
```

```
param.setPort("COM2");
```

```
param.setBaudRate(Baud._460800);
```

```
Com com = new Com(param);
```

¿Cómo enviar datos?

Para realizar este trabajo Giovynet Driver utiliza la clase **giovynt.serial.Com**, que dispone de tres métodos que se presentan a continuación:

Método sendArrayChar(char[] data) : void

Este método es usado para enviar elementos de un " array de tipo char". El intervalo de tiempo con que se envía cada elemento está determinado por el método **setMinDelayWrite(int milisegundos)** de la clase **giovynt.serial.Parameters**, por "default" el tiempo establecido para Windows es 0 milisegundos, y para Linux es de 10 milisegundos. Por ejemplo, el siguiente código, se utiliza para enviar los elementos *char* almacenados en un array, cada 100 milisegundos:

```
public static void main(String[] args)throws Exception{
    Parameters param = new Parameters();
    param.setPort("COM1");
    param.setBaudRate(Baud._9600);
    param.setMinDelayWrite(100);
    Com com1 = new Com(param);
    char[] data = {'1','A','2','B'};
    com1.sendArrayChar(data);
    com1.close();
}
```

La ejecución de este código, luego de 100 milisegundos enviará el *char* '1', luego de otros 100 milisegundos se enviará el *char* 'A', luego de otros 100 milisegundos se enviará el *char* '2', así sucesivamente hasta que se envíe el último *char* almacenado en el *array data*.

Método `sendSingleData(overloaded)` : void

Este método es usado para enviar un elemento de tipo *char*, o de tipo *String* o de tipo *Hex*. El elemento se enviará transcurrido el tiempo determinado en el método **`setMinDelayWrite(int milisegundos)`** de la clase **`giovynet.serial.Parameters`**, por "default" el tiempo establecido para Windows es 0 milisegundos, y para Linux es de 10 milisegundos. Por ejemplo, el siguiente código muestra como enviar los datos 'a', 41, "S", en intervalos de 100 milisegundos:

```
public static void main(String[] args)throws Exception{
    Parameters param = new Parameters();
    param.setPort("COM1");
    param.setBaudRate(Baud._9600);
    param.setMinDelayWrite(100);
    Com com1 = new Com(param);
    com1.sendSingleData('a');
    com1.sendSingleData(41);
    com1.sendSingleData("S");
    com1.sendSingleData(0xff);
    com1.close();
}
```

La ejecución del anterior código, enviará luego de transcurridos 50 milisegundos, el elemento "h", seguidos de otros 50 milisegundos, se enviará el elemento "e", luego de otros 50 milisegundos, se enviará el elemento "l", y así sucesivamente hasta completar la palabra "hello".

¿Cómo recibir datos?

Para realizar esta tarea Giovynet Driver utiliza la clase **`giovynet.serial.Com`**, que dispone de cinco métodos que se presentan a continuación:

Método `receiveSingleChar()` : char

Este método es usado para recibir un dato ASCII, luego del tiempo establecido por el método **`setMinDelayWrite(int milisegundos)`** de la clase **`giovynet.serial.Parameters`**, por "default" el tiempo establecido para Windows es 0 milisegundos, y para Linux es de 10 milisegundos. No se recomienda para recibir caracteres de control, tales como <ACK>, <NACK>, <LF>, ...etc. Por

ejemplo el siguiente código se utiliza para recibir 10 elementos tipo *char*, cada 50 milisegundos, e imprimirlos por consola:

```
public static void main(String[] args){

    Parameters param = new Parameters();
    param.setPort("COM1");
    param.setBaudRate(Baud._9600);
    param.setMinDelayWrite(50);
    Com com1 = new Com(param);

    char data;
    for(int x=0; x<10; x++){
        data=com1.receiveSingleChar();
        System.out.println(data);
    }

    com1.close();
}
```

Método receiveSingleCharAsInteger() : int

Este método es usado para recibir un dato ASCII en representación entera, luego del tiempo establecido por el método **setMinDelayWrite(int milisegundos)**, de la clase **giovynet.serial.Parameters**, por "default" el tiempo establecido para Windows es 0 milisegundos, y para Linux es de 10 milisegundos.

Este método es recomendado para recibir caracteres de control tales como, <ACK>, <NACK>, <LF>, ...etc. Por ejemplo, el siguiente código recibe 10 datos tipo *Int*, cada 50 milisegundos, y los imprime por consola:

```
public static void main(String[] args)throws Exception{
    Parameters param = new Parameters();
    param.setPort("COM1");
    param.setBaudRate(Baud._9600);
    param.setMinDelayWrite(50);
    Com com1 = new Com(param);
    int data;
    for(int x=0; x<10; x++){
        data=com1.receiveSingleCharAsInteger();
        System.out.println(data);
    }
    com1.close();
}
```

Método **receiveToString(int amount) : String**

Este método se utiliza para recibir varios elementos ASCII y retornarlos en una cadena tipo *String*.

La recepción de cada elemento se realiza luego del tiempo establecido por el método

setMinDelayWrite(int milisegundos), de la clase **giovynet.serial.Parameters**, por "default" el tiempo establecido para Windows es 0 milisegundos, y para Linux es de 10 milisegundos. Por ejemplo, el siguiente código se utiliza para recibir 10 elementos tipo *String* cada 50 milisegundos, almacenarlos en una variable y luego de esto, imprimir la variable por consola.

```
public static void main(String[] args)throws Exception{
    Parameters param = new Parameters();
    param.setPort("COM1");
    param.setBaudRate(Baud._9600);
    param.setMinDelayWrite(50);
    Com com1 = new Com(param);

    String data;
    data=com1.receiveToString(10);
    System.out.println(data);
    com1.close();
}
```

Método **receiveToStringBuilder(int untilAmount, StringBuilder stringBuilder) : void**

Este método se utiliza para recibir varios elementos de tipo "String" y almacenarlos en un objeto de tipo **StringBuilder**. La cantidad de objetos a recibir y el "StringBuilder", se pasan como parámetros.

La recepción de cada elemento se realiza luego del tiempo establecido por el método

setMinDelayWrite(int milisegundos), de la clase **giovynet.serial.Parameters**, por "default" el tiempo establecido para Windows es 0 milisegundos, y para Linux es de 10 milisegundos. Por ejemplo, el siguiente código lee 20 datos *String* cada 30 milisegundos y los almacena en un objeto de la clase *StringBuilder*.

```
public static void main(String[] args)throws Exception{
    Parameters param = new Parameters();
    param.setPort("COM1");
    param.setBaudRate(Baud._9600);
    param.setMinDelayWrite(30);
    Com com1 = new Com(param);

    StringBuilder stringBuilder = new StringBuilder;
```

```

        com1.receiveToStringBuilder(20,stringBuilder);
        com1.close();
    }

```

¿Cómo enviar caracteres de control?

Para enviar caracteres de control se recomienda utilizar su representación decimal o hexadecimal con el método **sendSingleData()**, de la clase **giovynet.serial.Com**.

Por ejemplo el siguiente código envía el símbolo ACK (acknowledge):

```

public static void main (String [] args) throws Exception {
    Parameters param = new Parameters ();
    param.setPort ("COM1");
    param.setBaudRate (Baud._9600)
    Com1 = new Com Com (param);

    int ack = 6 //Integer representation of ACK in ASCII code is 6
    data = com1.sendSingleData (ack);
    com1.close ();
}

```

¿Cómo recibir caracteres de control?

Para recibir caracteres de control se recomienda obtenerlos en representación decimal o hexadecimal, usando el método **receiveSingleCharAsInteger()**, de la clase **giovynet.serial.Com**. Por ejemplo, el siguiente código imprime en pantalla "OK" cuando recibe el símbolo ACK.

```

public static void main(String[] args)throws Exception{
    Parameters param = new Parameters();
    param.setPort("COM1");
    param.setBaudRate(Baud._9600);
    Com com1 = new Com(param);
    int ack=06;//Integer representation of ACK in ASCII code is 06
    int data=0;
    while(true){
        data=com1.receiveSingleCharAsInteger();
        if (data==ack){

```

```

        System.out.println("OK");
        break;
    }
}
com1.close();
}

```

¿Cómo implementar hilo para recibir datos de forma independiente?

Suponga que se desea construir una aplicación que cada vez que lea o reciba cierto caracter (o caracteres) por puerto serie, desempeñe una tarea como por ejemplo, mostrar un mensaje por pantalla, enviar un correo, realizar una consulta en base de datos, o quizá enviar un mensaje por el mismo puerto serie. Esto se puede hacer implementando un hilo o "Thread".

En Java hay dos maneras de implementar un "thread", una manera es heredar la clase **Thread** y la otra manera es implementar la interfaz **Runnable**, en ambos casos se implementa el método **run()** con las tareas que se quieren ejecutar independientes a la tarea principal o "hilo main".

Como ejemplo, a continuación se mostrará el código de dos clases que trabajan de forma separada pero que hacen parte de la misma aplicación.

La clase **EntryPoint** lanza la segunda clase (**ReadiangThread**) y envía datos por el puerto serie "COMUSB3", y la clase **ReadiangThread**, recibe o lee los datos del mismo puerto serie de forma independiente, y los imprime en consola.

```

import giovynet.serial.Com;
import giovynet.serial.Parameters;
public class EntryPoint {

    private static Com com;

    public static void main(String[] args) throws Exception {
        System.out.println("INICIÓ APLICACIÓN");
        Parameters param = new Parameters();
        param.setPort("COMUSB3"); //Selecciona un puerto
        com = new Com(param); //Instancia el COM
        //Instacia y ejecuta el hilo de lectura (Ver clase HiloLectura)
        Thread hiloLectura = new Thread(new HiloLectura(com));
        //Inicia el hilo de lectura
        hiloLectura.start();

        //Envia 10 veces cada 0,5 segundos el entero 2
        for (int x = 0; x < 10; x++) {

```

```

        com.sendSingleData('A');
        System.out.println("Envía el valor A por el puerto COMUSB3.");
        Thread.sleep(270);
    }

    //Detiene el hilo
    hiloLectura.interrupt();
    //Asigna null a la referencia para que sea candidata a destruir por el garbage collector
    hiloLectura = null;
    //Llama al garbage collector para liberar memoria.
    System.gc();
    System.out.println("FINALIZÓ LA APLICACIÓN");
}
}

```

```

import giovynet.serial.Com;
public class HiloLectura implements Runnable {

    private Com com;

    public HiloLectura(Com com) {
        this.com = com;
    }

    @Override
    //Ejecuta este método cuando se llama a HiloLectura.start()
    public void run() {
        System.out.println(" -->Inicia ejecución de hilo de lectura.");
        try{
            char data = 0;
            while (true) { //Realiza un ciclo infinito
                data= com.receiveSingleChar();// lee el puerto
                if (data !=0){ //Si el dato leído es diferente de cero
                    //Imprime el dato, espera 50mS y vuelve a leer hasta que el dato sea igual a cero.
                    do {
                        System.out.println(" Lectura desde hilo (Thread) el valor: "+
                            data);
                        Thread.sleep(250);
                        data = com.receiveSingleChar();
                    }while (data != 0);
                }
            }
        }catch (Exception e) {
            System.out.println(" <--Finaliza ejecución de hilo de lectura.");
        }
    }
}

```


Capítulo 7. Distribución de aplicaciones

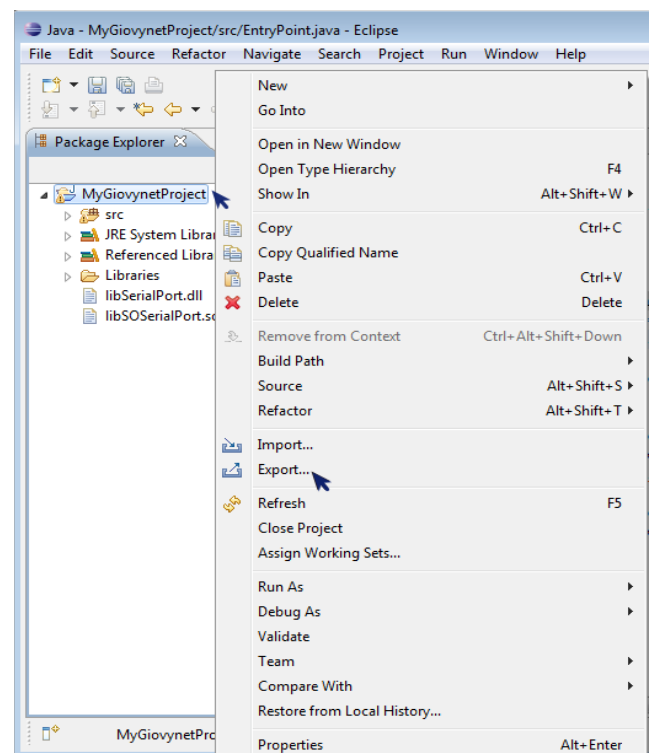
Eclipse IDE permite crear fácilmente un archivo ejecutable, sin embargo no solo este archivo es necesario para ejecutar aplicaciones Java con Giovynet Driver, junto con este archivo se deben adjuntar los archivos nativos de Giovynet Driver, y previo a la ejecución de la aplicación se debe instalar el JRE (Java Runtime Environment). En este capítulo se describe cómo realizar cada uno de estos pasos.

¿Que es y cómo se instala el JRE?

Una aplicación Java con Giovynet Driver, podría ser ejecutada en Linux y Windows indistintamente, para que esto sea posible, tanto Linux como Windows deben tener instalada una aplicación conocida como entorno de ejecución Java o JRE . En sistemas operativos como Ubuntu y Debian el JRE ya viene instalado por defecto, pero esto no sucede en Windows, para este sistema operativo se debe instalar el JRE manualmente. Para realizar esto simplemente diríjase al sitio web: <http://www.java.com/es/download/>, descargue y ejecute el JRE. La siguiente imagen muestra el sitio web que permite descargar la aplicación.

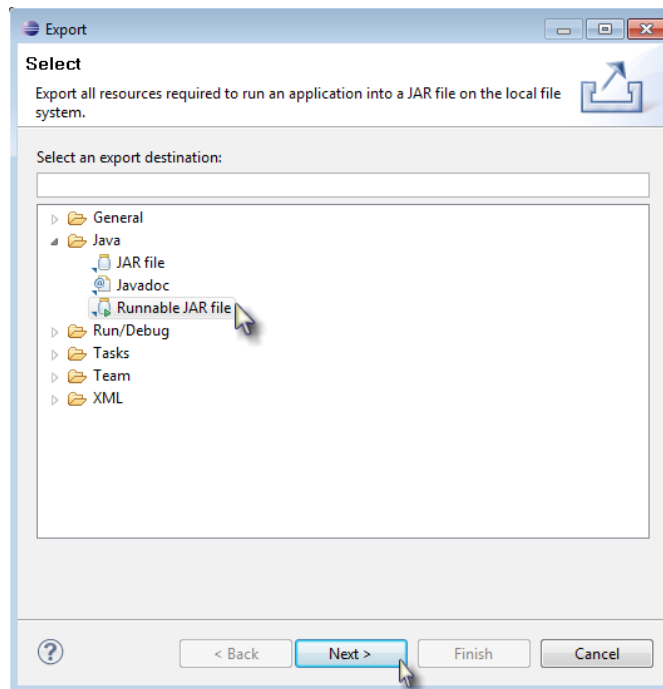
¿Cómo crear un JAR ejecutable con Eclipse IDE?

Para crear un archivo ejecutable Java, la aplicación debe tener al menos una clase con el método **public static void main(String[] arg)**, este método conforma el punto de entrada de la aplicación, esto quiere decir que la ejecución del archivo Java, equivale a ejecutar el método “main” de una clase previamente seleccionada, por tanto, en dicho método se deben instanciar las clases que conforman los componentes principales del programa.



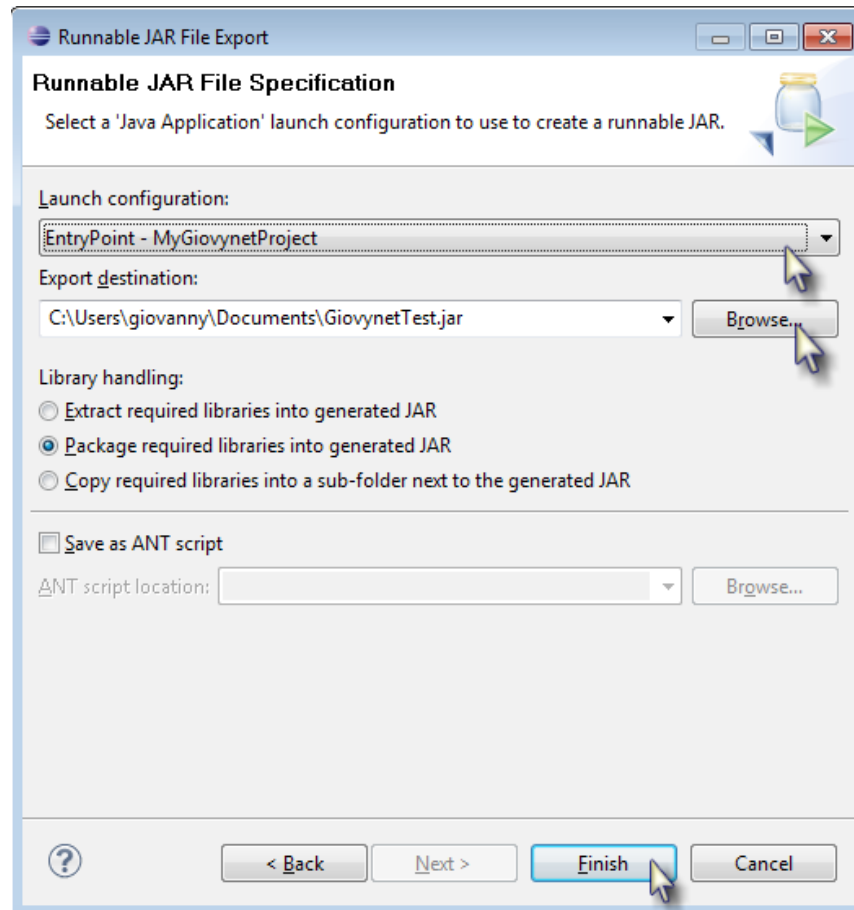
Para crear un archivo Java ejecutable en Eclipse IDE, de click derecho sobre la carpeta del proyecto, luego seleccione la opción “export”, así como muestra la imagen derecha.

Luego de esto aparecerá una ventana que muestra opciones de exportación, seleccione la opción **Runnable JAR File** del folder **Java** y de click en el botón “Next”, así como se muestra a continuación.



En seguida se mostrará una ventana con varias opciones de exportación, la opción “Launch configuration” permite seleccionar la “Clase” con método “main” que será llamado cuando se ejecute el archivo Java. La opción “Export destination” permite seleccionar el folder de destino y el nombre del archivo Java. La opción “Library Handling” debe seleccionarse como: “Package required libraries into generated JAR”, esto permite que las clases y métodos de Giovynet Driver usados en la construcción de la aplicación, se empaqueten dentro del archivo Java ejecutable. Como última opción se encuentra “Save as ANT script”, esta opción permite crear un archivo xml, interpretado por la herramienta de compilación y construcción ANT.

Después de seleccionar las opciones de exportación y dar click en el botón “Finish”, se crea el archivo Java ejecutable. La siguiente figura muestra las opciones seleccionadas para la creación de un archivo Java ejecutable llamado GiovynetTest.jar, cuyo método “main” está en la clase EntryPoint.



¿Cómo crear un folder de distribución?

El archivo Java ejecutable por si solo no podrá ejecutar la aplicación, ya que este archivo intentará enlazar los archivos nativos "libSerialPort.dll" y "libSOSerialPort.so", si no encuentran estos archivos la aplicación lanzará la excepción **UnsatisfiedLinkError: Can't load library...**

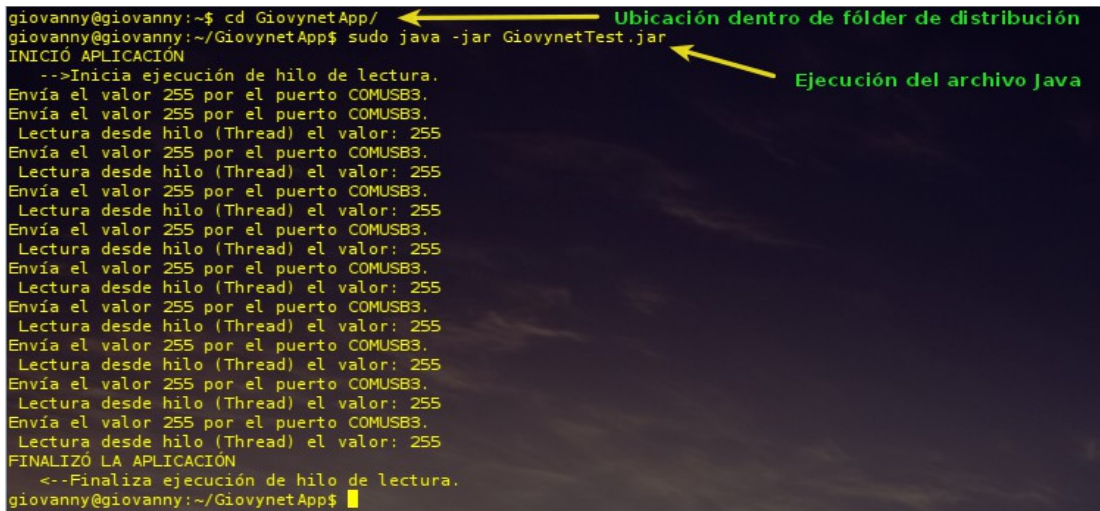
Por tanto la manera correcta de distribuir una aplicación es conformar un folder que dentro tenga el archivo java ejecutable y los archivos nativos. De esta manera cuando se ejecute el archivo java se podrá referenciar dichos archivos debido a que se encuentran en la misma ubicación. La siguiente gráfica ilustra esta idea.

¿Cómo iniciar una aplicación desde el folder de distribución?

Si se esta en Windows, para iniciar una aplicación Java con Giovynet Driver, ubíquese dentro del folder de distribución y ejecute el archivo java dando doble click sobre este. Indistintamente si se está en Windows o Linux ejecute desde consola la sentencia **java -jar** seguida del nombre del archivo Java ejecutable.

Si la aplicación no presenta interfaz gráfica luego de dar doble click en el archivo Java ejecutable no aparecerá nada. Por tanto para ejecutar aplicaciones que no presenten interfaz gráfica se recomienda lanzar la aplicación mediante la sentencia **java -jar**.

A continuación se muestra la ejecución de una aplicación Java con Giovynet Driver desde una consola usando el comando **java -jar** en Ubuntu Linux.

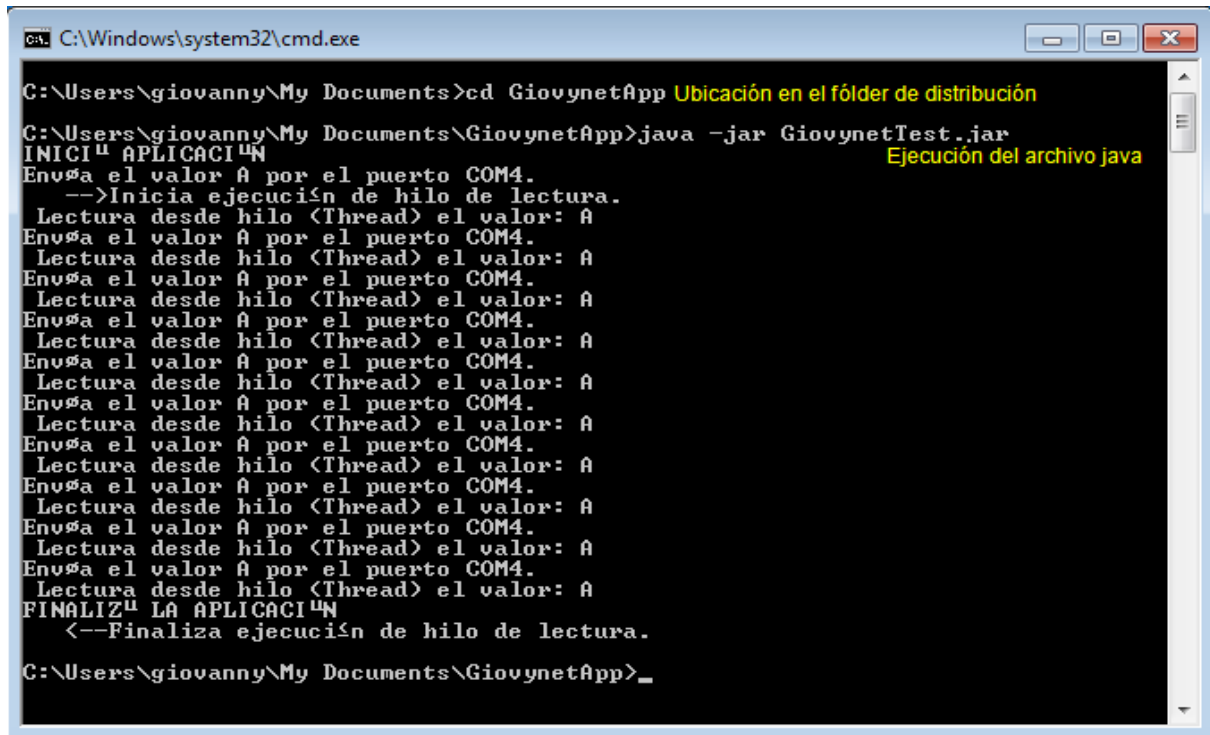


```
giovanny@giovanny:~$ cd GiovynetApp/
giovanny@giovanny:~/GiovynetApp$ sudo java -jar GiovynetTest.jar
INICIÓ APLICACIÓN
-->Inicia ejecución de hilo de lectura.
Envía el valor 255 por el puerto COMUSB3.
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
Envía el valor 255 por el puerto COMUSB3.
Lectura desde hilo (Thread) el valor: 255
FINALIZÓ LA APLICACIÓN
<--Finaliza ejecución de hilo de lectura.
giovanny@giovanny:~/GiovynetApp$
```

Ubicación dentro de folder de distribución

Ejecución del archivo Java

Finalmente, se muestra la ejecución de una aplicación Java con Giovynet Driver desde una consola usando el comando **java -jar** desde Windows.



```
C:\Windows\system32\cmd.exe

C:\Users\giovanny\My Documents>cd GiovynetApp Ubicación en el folder de distribución
C:\Users\giovanny\My Documents\GiovynetApp>java -jar GiovynetTest.jar Ejecución del archivo java
INICIADA APLICACIÓN
Envíase el valor A por el puerto COM4.
-->Inicia ejecución de hilo de lectura.
Lectura desde hilo <Thread> el valor: A
Envíase el valor A por el puerto COM4.
Lectura desde hilo <Thread> el valor: A
Envíase el valor A por el puerto COM4.
Lectura desde hilo <Thread> el valor: A
Envíase el valor A por el puerto COM4.
Lectura desde hilo <Thread> el valor: A
Envíase el valor A por el puerto COM4.
Lectura desde hilo <Thread> el valor: A
Envíase el valor A por el puerto COM4.
Lectura desde hilo <Thread> el valor: A
Envíase el valor A por el puerto COM4.
Lectura desde hilo <Thread> el valor: A
Envíase el valor A por el puerto COM4.
Lectura desde hilo <Thread> el valor: A
Envíase el valor A por el puerto COM4.
Lectura desde hilo <Thread> el valor: A
FINALIZADA LA APLICACIÓN
--Finaliza ejecución de hilo de lectura.
C:\Users\giovanny\My Documents\GiovynetApp>
```

Hasta aquí lo concerniente a la distribución de aplicaciones.

Capítulo 8. Excepciones frecuentes

A continuación se describen algunas excepciones que pueden ser lanzadas por una aplicación Java con Giovynet Driver en tiempo de ejecución o compilación. Se describe también los motivos por los cuales se provocan, y se sugiere como solucionar estas.

java.lang.UnsatisfiedLinkError: Can't load library.

Se produce debido a que no se puede referenciar los archivos nativos (**libSerialPort.dll** y **libSOSerialPort.so**), porque no se encuentran en el folder del proyecto.

Para solucionar este error, copie los archivos **libSerialPort.dll** y **libSOSerialPort.so**, que se encuentran en el folder “NativeLibraries”; y pégelos en el fólder del proyecto.

java.lang.UnsatisfiedLinkError: (Possible cause: architecture word width mismatch).

Se produce debido a que no se puede referenciar los archivos nativos (**libSerialPort.dll** y **libSOSerialPort.so**), porque estos están compilados en una arquitectura diferente a la arquitectura del PC. Para ilustrar una posible situación, suponga que se quiere realizar un desarrollo para una arquitectura x64 (64-bit), pero en este desarrollo se usan archivos nativos compilados para arquitectura x86 (32-bit), debido a que las arquitecturas no son compatibles al intentar compilar se producirá este error. La solución es remplazar los archivos nativos por archivos nativos de arquitectura correcta.

giovynet.permissions.PermissionsException: You have exceeded the number allowed of devices instantiated for this license.

Esta excepción se produce debido a que se intenta instanciar una cantidad de dispositivos mayor a los permitidos por la licencia de GiovynetDriver. Como ejemplo se tiene el siguiente caso: suponga que se adquirió Giovynet Driver con licencia para un dispositivo, y se utiliza para hacer un desarrollo donde se instancia al tiempo un puerto serie y un puerto GIV8DAQ. Al intentar ejecutar esta aplicación se lanzará esta excepción, debido a que la cantidad de instancias es mayor a la permitida por la licencia adquirida. La solución es controlar programáticamente la aplicación para que no supere el número de instancias permitidas.

giovynet.devices.DeviceException: The device was closed, you should re-instantiate this object.

Esta excepción se produce debido a que se intenta hacer uso de un dispositivo que fué cerrado previamente. La solución es re-instanciar el dispositivo.

giovynet.permissions.PermissionsException: The device is inactive. To activate it, first you must use the method: `activate (String activation_Key);`

Esta excepción se produce debido a que se intenta ejecutar un comando para un puerto GIV que no ha sido previamente activado. La solución consiste en activar el puerto GIV antes de usar los comandos de este. Para activar un dispositivo GIV use el método **activate (String activation_Key);** . El parámetro **activation_key** corresponde a la clave de activación que viene atada al conector USB del puerto GIV.

Giovynet.com © 2012. todos los derechos reservados .

S.D.G.