

Communication par liaison série RS232 en Java

Sébastien Jean

IUT de Valence
Département Informatique

v2.2, 7 décembre 2011

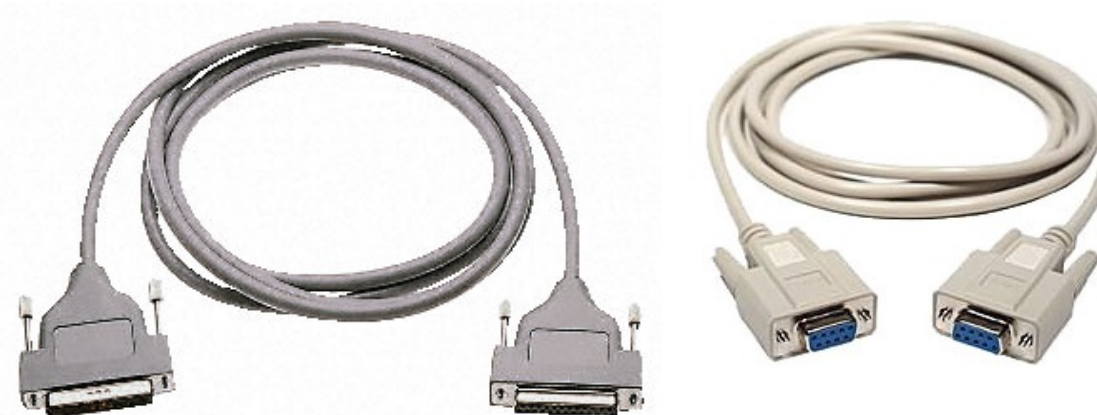


Rappel sur la liaison série RS232

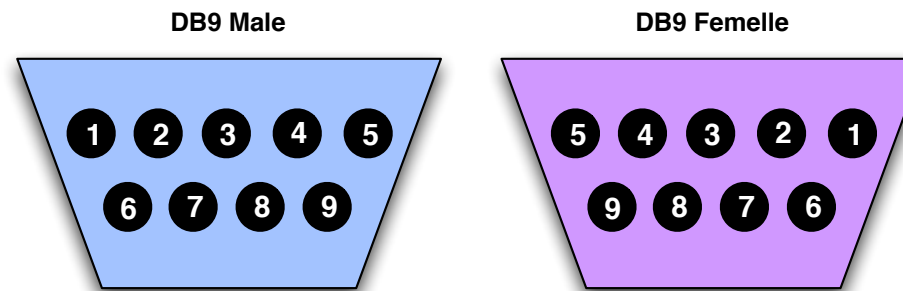
- **Transmission asynchrone de « caractères »**
 - Suites binaires de longueur fixe comprise entre 5 et 8 bits
- Débit typique jusqu'à **un peu plus de 200 kbit/s**
- Les caractères de longueur n bits sont transmis sous la forme d'une **trame** de m bits ($m > n$) comprenant
 - 1 bit de **start** (signalant le début de la trame)
 - n **bits de données**
 - 0 ou 1 **bit de parité** (paire ou impaire)
 - s **bits de stop** (signalant la fin de la trame)
- **Contrôle de flux optionnel**
 - Matériel (*RTS/CTS*)
 - Logiciel (*XON/XOFF*)

Rappel sur la liaison série RS232 (suite)

- Utilisée pour relier des terminaux (**DTE**, *Data Terminal Equipment*, typiquement des PCs) à des équipements communicants (**DCE**, *Data Communication Equipment*, typiquement des modems)
- Deux types de connecteurs
 - 25 broches (**DB-25**)
 - 9 broches (**DB-9**)
- Le câblage **DTE-DCE** est **droit**, le câblage **DTE-DTE** est **croisé**



Rappel sur la liaison série RS232 (fin)



- ① **CD** (*Carrier Detect*) : Détection de porteuse
- ② **RXD** (*Receive Data*) : Réception de données
- ③ **TXD** (*Transmit Data*) : Transmission de données
- ④ **DTR** (*Data Terminal Ready*) : DTE prêt
- ⑤ **GND** (*Signal Ground*) : Masse logique
- ⑥ **DSR** (*Data Set Ready*) : DCE prêt
- ⑦ **RTS** (*Request To Send*) : Demande d'émission
- ⑧ **CTS** (*Clear To Send*) : Autorisation d'émission
- ⑨ **RI** (*Ring Indicator*) : Indicateur de sonnerie

Java Communications API

- **Java Communications API** (a.k.a. **JavaComm**) est une API pour la gestion de **communications** (série ou parallèle) avec un équipement externe depuis une application Java
 - <http://www.oracle.com/technetwork/java/index-jsp-141752.html>
 - Définie en 1997, version actuelle : 3.0
- JavaComm propose deux modèles gestion de la communication
 - **Flux**, où la liaison est vue « **comme un socket** » fournissant des flux de lecture/écriture binaire d'octets
 - **Événementiel**, où l'application utilisant la liaison peut être **notifiée d'événements** particuliers (données disponibles, ...)

Implémentations de *JavaComm*

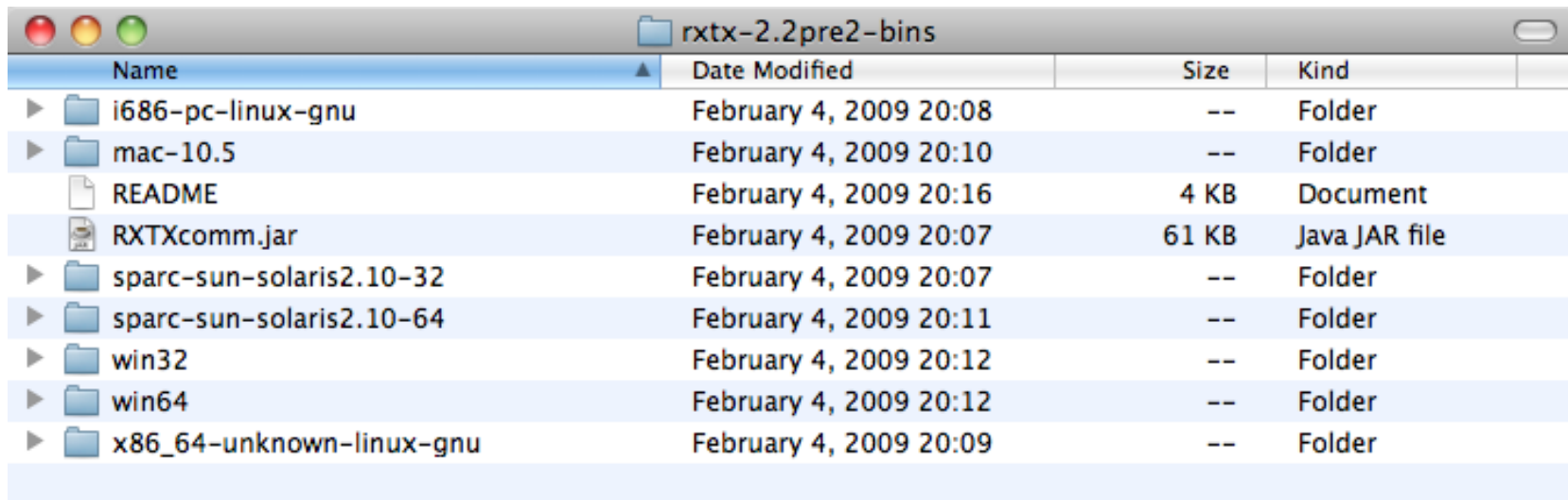
- Les applications utilisant *JavaComm* sont « portables »
 - A condition de **ne pas indiquer explicitement dans le code le nom symbolique des ports** et **d'installer** sur chaque plate-forme un **driver**
 - Bibliothèque native (.dll, .so) faisant le pont entre le système et *JavaComm*

RxTx ?

- **RxTx** (<http://www.rxtx.org>) est une **bibliothèque open-source antérieure** à *JavaComm* mais qui s'est rendue **compatible**
- RxTx est considérée comme l'implémentation de référence de l'API JavaComm, et fournit en plus
 - Le support pour la communication *I2C*, *Raw* et assure un contrôle plus fin de la communication série
 - Le support d'un très grand nombre de plates-formes
 - Windows, MacOS X, Linux-x86 : architectures 32 ou 64 bits
 - Distributions Linux embarquées (\simeq 30 plates-formes supportées)
- Version actuelle : 2.2Pre2
- Rem : Les classes de RxTx sont définies dans le paquetage **gnu.io**

Distribution binaire RxTx

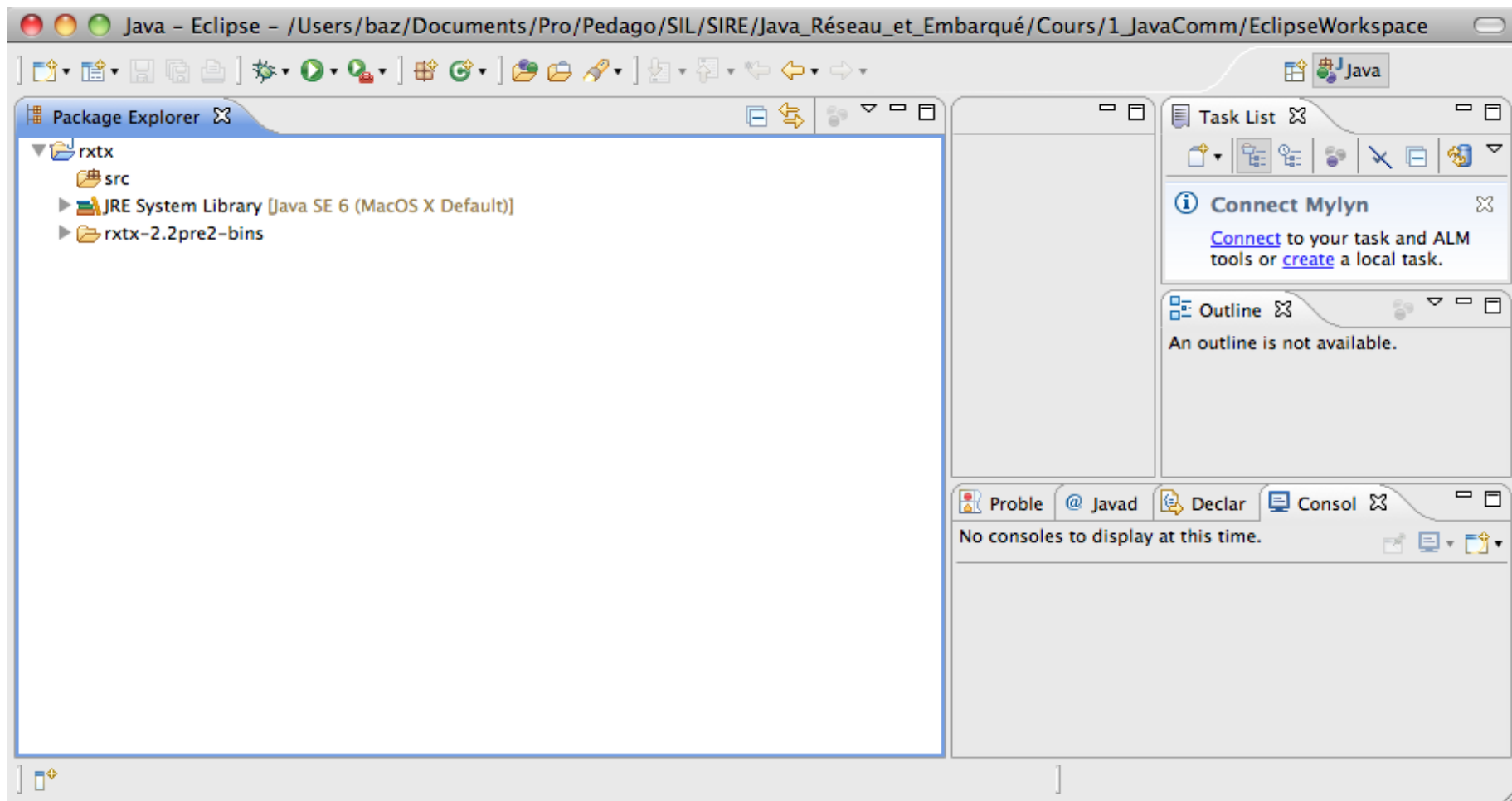
- Dernière version de la distribution binaire : `rxtx-2.2Pre2-bins.zip`
 - Version multi-plates-formes standards (Win32, Linux, Solaris, MacOS X)
 - `RXTXComm.jar` contient la **partie** Java de la bibliothèque **commune** à toutes les plates-formes
 - Les **sous-répertoires** associés aux différentes plates-formes contiennent la **partie native** de la bibliothèque (driver)



Name	Date Modified	Size	Kind
▶ i686-pc-linux-gnu	February 4, 2009 20:08	--	Folder
▶ mac-10.5	February 4, 2009 20:10	--	Folder
▶ README	February 4, 2009 20:16	4 KB	Document
▶ RXTXcomm.jar	February 4, 2009 20:07	61 KB	Java JAR file
▶ sparc-sun-solaris2.10-32	February 4, 2009 20:07	--	Folder
▶ sparc-sun-solaris2.10-64	February 4, 2009 20:11	--	Folder
▶ win32	February 4, 2009 20:12	--	Folder
▶ win64	February 4, 2009 20:12	--	Folder
▶ x86_64-unknown-linux-gnu	February 4, 2009 20:09	--	Folder

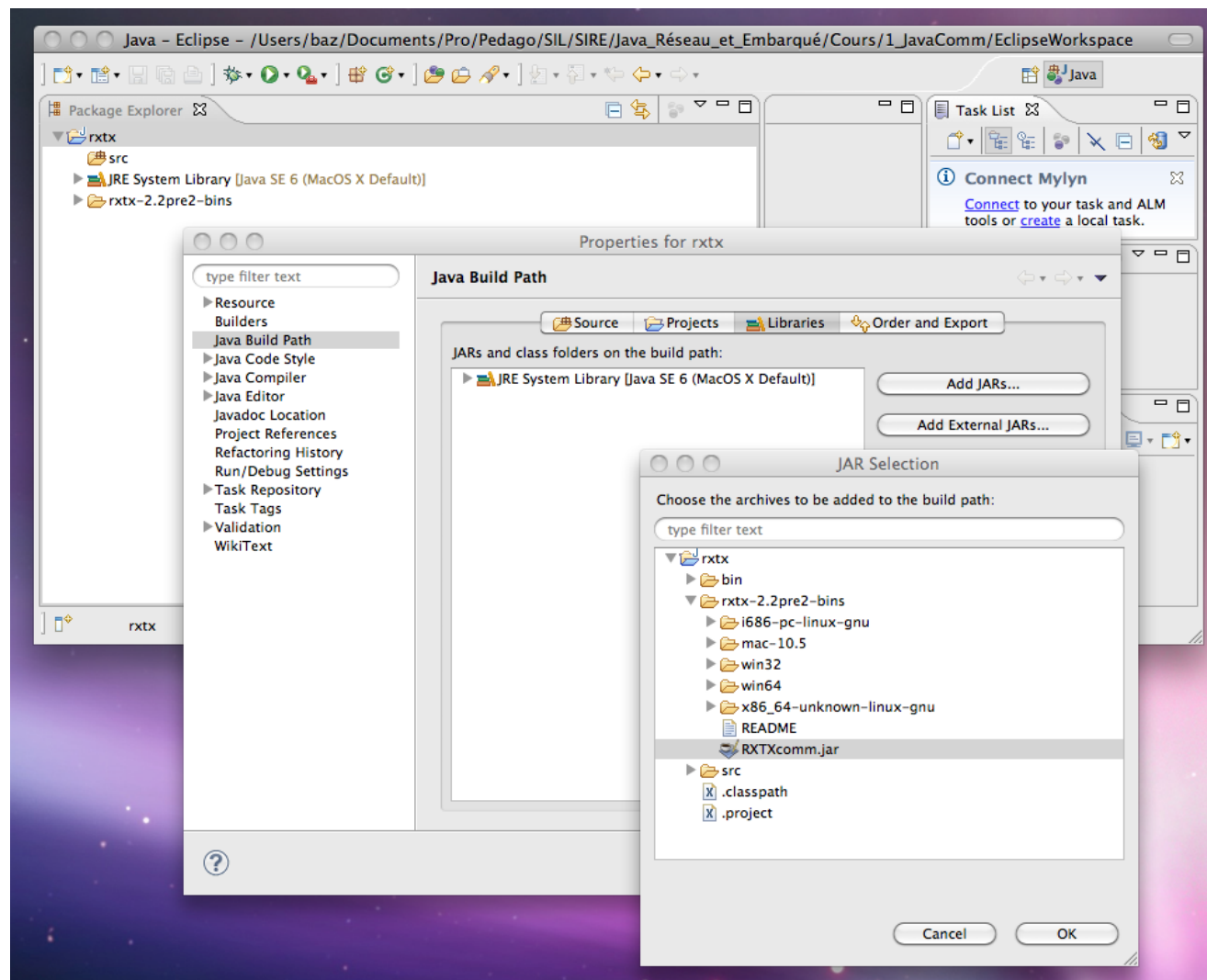
RxTx et Eclipse

- Créer un projet *Eclipse* et placer la distribution binaire de RxTx dans le répertoire du projet



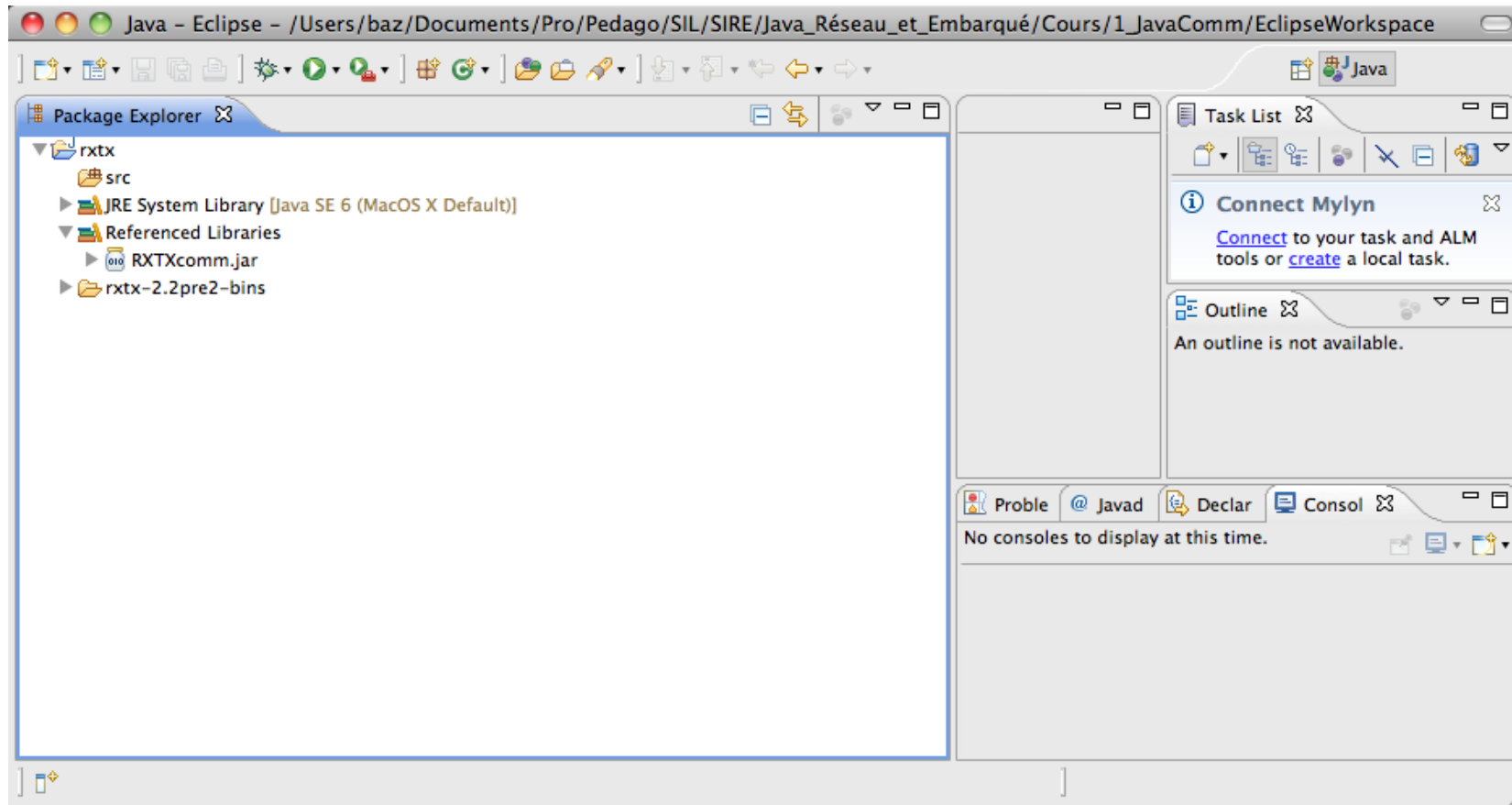
RxTx et Eclipse (suite)

- Ajouter le jar RxTx au projet (via les propriétés du projet)
 - Indispensable pour rendre les classes utilisables par l'application



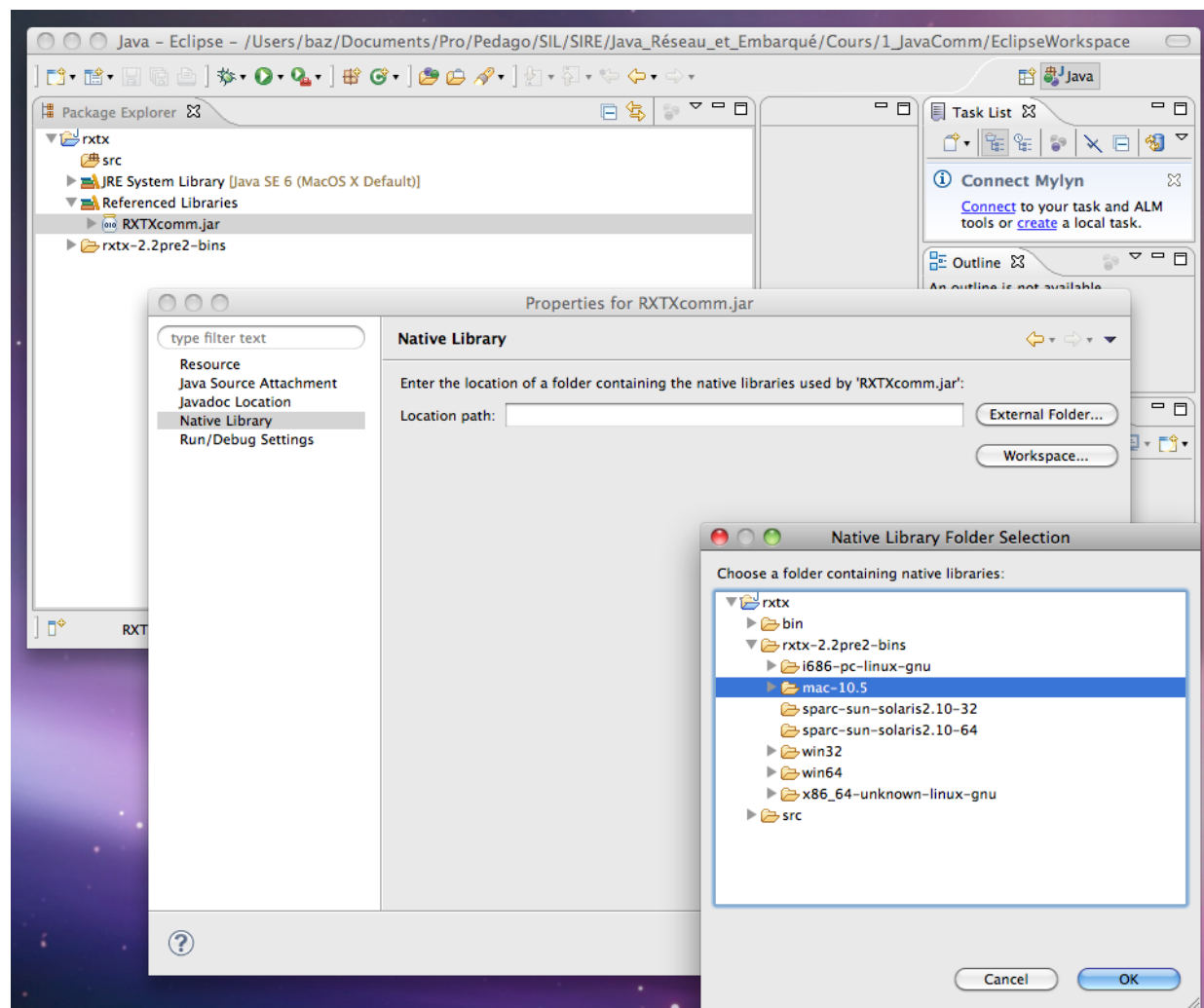
RxTx et Eclipse (suite)

- Rem : le jar doit apparaître dans « *Referenced Libraries* »



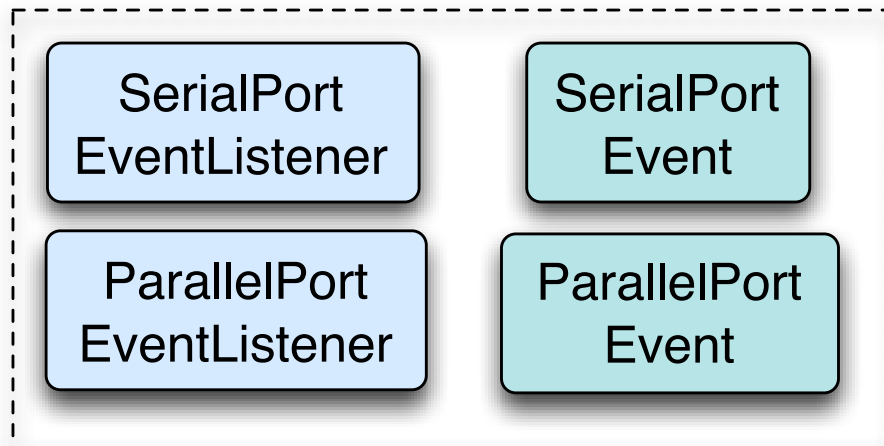
RxTx et Eclipse (fin)

- Câbler la bibliothèque native au jar (via les propriétés du jar)
 - Indispensable pour que les classes du jar puissent charger dynamiquement la bibliothèque native

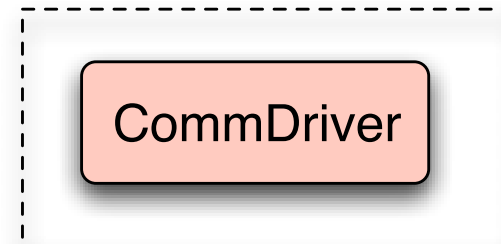


Vue globale de l'API

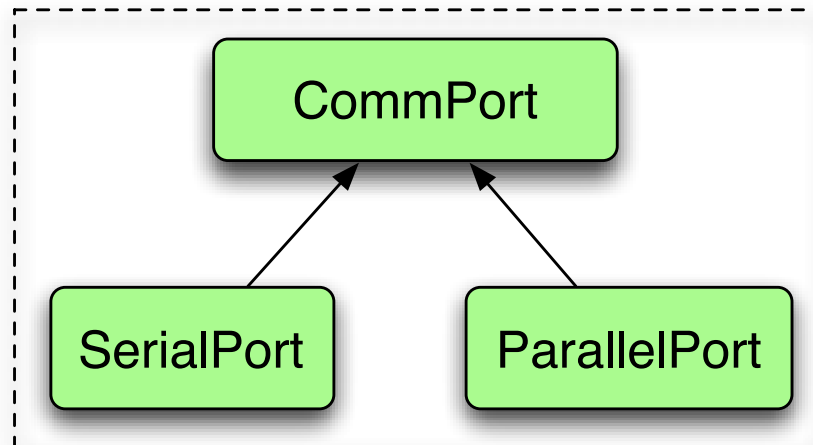
Gestion des événements



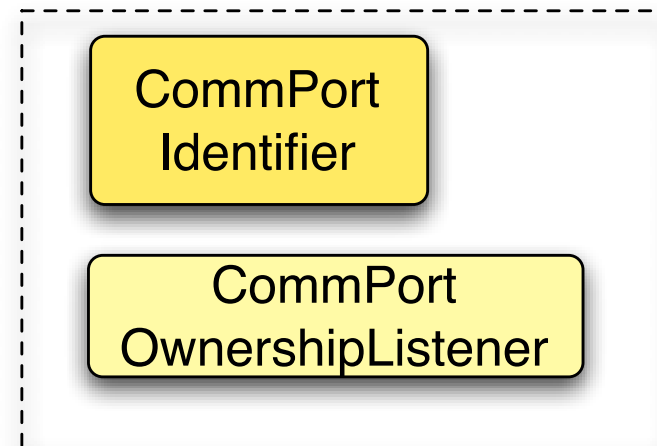
Interface système



Communication



Enumération, ouverture des ports



Enumération des ports, état

Constantes de la classe `javax.comm.CommPortIdentifier`

<code>public final static int</code>	<code>PORT_SERIAL</code>	Constante identifiant un port comme étant de type série.
<code>public final static int</code>	<code>PORT_PARALLEL</code>	Constante identifiant un port comme étant de type parallèle.

Méthodes de la classe `javax.comm.CommPortIdentifier`

<code>public static Enumeration</code>	<code>getPortIdentifiers()</code>	Obtention de tous les identifiants de ports.
<code>public static CommPortIdentifier</code>	<code>getPortIdentifier(String name)</code>	Obtention de l'identifiant de port de nom symbolique <i>name</i> . Soulève <i>NoSuchPortException</i> .
<code>public static CommPortIdentifier</code>	<code>getPortIdentifier(CommPort port)</code>	Obtention de l'identifiant de port associé au port de communication référencé par <i>port</i> . Soulève <i>UnknownHostException</i> .
<code>public String</code>	<code>getName()</code>	Obtention du nom symbolique du port.
<code>public int</code>	<code>getType()</code>	Obtention du type du port (cf. constantes).
<code>public boolean</code>	<code>isCurrentlyOwned()</code>	Test d'occupation du port. Retourne <i>true</i> si le port est occupé par une application Java.
<code>public String</code>	<code>getCurrentOwner()</code>	Obtention du nom de l'application occupant le port (si le port est occupé par une application Java).

Exemple : énumérateur de ports

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

public class PortsEnumerator
{
    public static void main(String[] args)
    {
        Enumeration ports = CommPortIdentifier.getPortIdentifiers();
        int i = 1;
        while (ports.hasMoreElements())
        {
            CommPortIdentifier port = (CommPortIdentifier) ports.nextElement();
            System.out.println("Port n°"+i++);
            System.out.println("\tNom\t:\t"+port.getName());
            String type = null;
            if (port.getPortType() == CommPortIdentifier.PORT_SERIAL) type = "Serie";
            else type = "Parallèle";
            System.out.println("\tType\t:\t"+type);
            String etat = null;
            if (port.isCurrentlyOwned()) etat = "Possédé par "+port.getCurrentOwner();
            else etat = "Libre";
            System.out.println("\tEtat\t:\t"+etat+"\n");
        }
    }
}
```

Ouverture de ports, état

Méthodes de la classe javax.comm.CommPortIdentifier		
public CommPort	open(String app, int timeout)	Ouverture du port par l'application app. Si le port n'est pas libéré après timeout abandon. Soulève <i>PortInUseException</i> .
public CommPort	open(FileDescriptor fd)	Idem précédente mais en utilisant un descripteur de fichier (si la plate-forme supporte cette fonctionnalité). Soulève <i>UnsupportedCommOperationException</i> .
public void	addPortOwnershipListener (CommPortOwnershipListener l)	Ajout d'un auditeur d'événements « occupation/libération de port ».
public void	removePortOwnershipListener (CommPortOwnershipListener l)	Retrait d'un auditeur d'événements « occupation/libération de port ».

Constantes de l'interface javax.comm.CommPortOwnershipListener		
public final static int	PORT_OWNED	Constante indiquant que le port vient d'être occupé.
public final static int	PORT_UNOWNED	Constante indiquant que le port vient d'être libéré.
public final static int	PORT_OWNERSHIP_REQUESTED	Constante indiquant que le port (déjà occupé) vient de faire l'objet d'une autre demande d'ouverture.

Méthodes de l'interface javax.comm.CommPortOwnershipListener		
public void	ownershipChange(int state)	Méthode de callback permettant à l'auditeur de prendre en compte le changement d'état du port.

Application exemple : auditeurs d'événements

- L'**application** :
 - Prend en **paramètre** un **nom symbolique de port** de communication
 - **Crée un thread** toutes les secondes
- Le **thread** :
 - Prend en **paramètre** un **nom symbolique de port** de communication et un **booléen** indiquant s'il décidera de **libérer le port** si quelqu'un d'autre le demande
 - **Tente pendant 3 secondes** d'obtenir le port
 - S'il l'obtient, **occupe le port pendant 3 secondes**
 - **Selon son paramétrage** initial, **relâche le port** si quelqu'un le demande

Application exemple : auditeurs d'événements (suite)

```
import gnu.io.*;
public class PortOpenerAndListener extends Thread implements CommPortOwnershipListener
{
    private CommPortIdentifier portID;
    private CommPort port;
    private boolean releasePortOnRequest;
    private boolean ownPort;

    public PortOpenerAndListener(String threadName, String portName, boolean strong)
        throws NoSuchPortException
    {
        super(threadName);
        this.releasePortOnRequest = strong;
        this.portID = CommPortIdentifier.getPortIdentifier(portName);
        this.port = null;
        this.ownPort = false;
    }
    ...
}
```

Application exemple : auditeurs d'événements (suite)

```
import gnu.io.*;
public class PortOpenerAndListener extends Thread implements CommPortOwnershipListener
{
    ...
    public void run()
    {
        System.out.print("[ "+this.getName()+" ] starts and try to open port "
                        + this.portID.getName()+"\n");
        this.portID.addPortOwnershipListener(this);
        try {Thread.sleep(1000);} catch (InterruptedException e) {}
        try
        {
            this.port = this.portID.open(this.getName(), 2000);
            this.ownPort = true;
        }
        catch (PortInUseException e)
        {
            System.err.print("[ "+this.getName()+" ] stated that "
                            + this.portID.getName()+" is currently in use\n");
            this.portID.removePortOwnershipListener(this);
            System.out.print("[ "+this.getName()+" ] ends\n");
            return;
        }
        System.out.print("[ "+this.getName()+" ] owns port " + this.portID.getName()+"\n");
        try {Thread.sleep(3000);} catch (InterruptedException e) {}
        System.out.print("[ "+this.getName()+" ] closes port " + this.portID.getName()+"\n");
        this.port.close();
        this.ownPort = false;
        this.portID.removePortOwnershipListener(this);
        System.out.print("[ "+this.getName()+" ] ends\n");
    }
    ...
}
```

Application exemple : auditeurs d'événements (suite)

```
import gnu.io.*;
public class PortOpenerAndListener extends Thread implements CommPortOwnershipListener
{
    ...
    public void ownershipChange(int state)
    {
        switch (state)
        {
            case CommPortOwnershipListener.PORT_OWNED :
                System.out.print("[ "+this.getName()+" ] notified that port "
                                + this.portID.getName()+" is now owned by "
                                + this.portID.getCurrentOwner()+"\n");
                return;
            case CommPortOwnershipListener.PORT_UNOWNED :
                System.out.print("[ "+this.getName()+" ] notified that port "
                                + this.portID.getName()+" has been released\n");
                return;
            case CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED :
                System.out.print("[ "+this.getName()+" ] notified that port "
                                + this.portID.getName()+" is requested\n");
                if (this.ownPort)
                {
                    if (this.releasePortOnRequest)
                    {
                        System.out.print("[ "+this.getName()+" ] releases port "
                                        + this.portID.getName()+"\n");
                        this.port.close(); this.ownPort = false; return;
                    }
                    System.out.print("[ "+this.getName()+" ] does not release port "
                                    + this.portID.getName()+"\n");
                }
                return;
        }
    }
    ...
}
```

Application exemple : auditeurs d'événements (fin)

```
import gnu.io.*;
public class PortOpenerAndListener extends Thread implements CommPortOwnershipListener
{
    ...
    public static void main(String[] args)
    {
        for (int i=0;i<5;i++)
        {
            try {new PortOpenerAndListener("POL"+i,args[0],(i%2==0)).start();}
            catch (NoSuchPortException e)
            {
                System.err.print("No such port "+args[0]+", exiting...\n");
                System.exit(1);
            }
            try {Thread.sleep(1000);} catch (InterruptedException e) {}
        }
    }
}
```

[POL0] starts and try to open port COM1
 [POL1] starts and try to open port COM1
 [POL0] notified that port COM1 is now owned by POL0
 [POL1] notified that port COM1 is now owned by POL0
 [POL0] owns port COM1
 [POL2] starts and try to open port COM1
 [POL0] notified that port COM1 is requested
 [POL0] releases port COM1
 [POL0] notified that port COM1 has been released
 [POL1] notified that port COM1 has been released
 [POL2] notified that port COM1 has been released
 [POL1] notified that port COM1 is requested
 [POL2] notified that port COM1 is requested
 [POL0] notified that port COM1 is now owned by POL1
 [POL1] notified that port COM1 is now owned by POL1
 [POL2] notified that port COM1 is now owned by POL1
 [POL1] owns port COM1
 [POL1] owns port COM1

[POL0] notified that port COM1 is requested
 [POL1] notified that port COM1 is requested
 [POL1] does not release port COM1
 [POL2] notified that port COM1 is requested
 [POL2] stated that COM1 is currently in use
 [POL2] ends
 [POL3] starts and try to open port COM1
 [POL3] stated that COM1 is currently in use
 [POL0] notified that port COM1 is requested
 [POL1] notified that port COM1 is requested
 [POL1] does not release port COM1
 [POL3] notified that port COM1 is requested
 [POL3] ends
 [POL4] starts and try to open port COM1
 [POL0] closes port COM1
 [POL0] ends ...

Abstraction générale : classe CommPort

- `javax.comm.CommPort` représente un **port de communication série ou parallèle ouvert**

Méthodes de la classe <code>javax.comm.CommPort</code>		
<code>public String</code>	<code>getName()</code>	Obtention du nom symbolique associé au port de communication.
<code>public void</code>	<code>close()</code>	Libération du port de communication.
<code>public abstract InputStream</code>	<code>getInputStream()</code>	Obtention du flux de lecture binaire sur le port de communication. Soulève <i>IOException</i> .
<code>public abstract OutputStream</code>	<code>getOutputStream()</code>	Obtention du flux d'écriture binaire sur le port de communication. Soulève <i>IOException</i> .
<code>public abstract void</code>	<code>enableReceiveThreshlod(int t)</code>	Activation du seuil de réception (si supporté par le driver). Toute lecture sur le flux sera bloquante jusqu'à ce que <i>t</i> octets de données aient été reçus. Soulève <i>UnsupportedCommOperationException</i> .
<code>public abstract void</code>	<code>disableReceiveThreshlod()</code>	Désactivation du seuil de réception.
<code>public abstract boolean</code>	<code>isReceiveThreshlodEnabled()</code>	Test d'activation du seuil de réception.
<code>public abstract int</code>	<code>getReceiveThreshlod()</code>	Obtention du seuil de réception.

Abstraction générale : classe CommPort (fin)

Méthodes de la classe javax.comm.CommPort		
public abstract void	enableReceiveTimeout(int t)	Activation du délai de réception (si supporté par le driver). Toute lecture bloquante sur le flux sera débloquée au bout de t ms si aucune donnée (ou un nombre insuffisant de données, cf. receiveThreshold) n'a été reçue. Soulève <i>UnsupportedCommOperationException</i> .
public abstract void	disableReceiveTimeout()	Désactivation du délai de réception.
public abstract boolean	isReceiveTimeoutEnabled()	Test d'activation du délai de réception.
public abstract int	getReceiveTimeout()	Obtention du délai de réception.
public abstract void	enableReceiveFraming(int b)	Activation de la délimitation de trame (si supporté par le driver). Toute lecture bloquante sur le flux sera débloquée si l'octet servant de délimiteur de trames (spécifié par l'octet de poids faible de b) a été lu. Soulève <i>UnsupportedCommOperationException</i> .
public abstract void	disableReceiveFraming()	Désactivation de la délimitation de trame.
public abstract boolean	isReceiveFramingEnabled()	Test d'activation de la délimitation de trame.
public abstract int	getReceiveFramingByte()	Obtention du délimiteur de trame.
public abstract void	setInputBufferSize(int l)	Limitation de la taille du tampon de réception.
public abstract int	getInputBufferSize()	Obtention de la taille du tampon de réception.
public abstract void	setOutputBufferSize(int l)	Limitation de la taille du tampon d'émission.
public abstract int	getOutputBufferSize()	Obtention de la taille du tampon d'émission.

Abstraction série : classe SerialPort

Constantes de la classe javax.comm.SerialPort		
public final static int	DATABITS_5	Constante indiquant que le nombre de bits de données est de 5.
public final static int	DATABITS_6	Constante indiquant que le nombre de bits de données est de 6.
public final static int	DATABITS_7	Constante indiquant que le nombre de bits de données est de 7.
public final static int	DATABITS_8	Constante indiquant que le nombre de bits de données est de 8.
public final static int	STOPBITS_1	Constante indiquant que le nombre de bits de stop est de 1.
public final static int	STOPBITS_1_5	Constante indiquant que le nombre de bits de stop est de 1.5.
public final static int	STOPBITS_2	Constante indiquant que le nombre de bits de stop est de 2.
public final static int	PARITY_ODD	Constante indiquant qu'il y a un bit de parité impaire.
public final static int	PARITY_EVEN	Constante indiquant qu'il y a un bit de parité paire.
public final static int	PARITY_NONE	Constante indiquant qu'il n'y a pas de bit de parité.
public final static int	FLOWCONTROL_NONE	Constante indiquant qu'il n'y a pas de contrôle de flux.
public final static int	FLOWCONTROL_RTSCTS_IN	Constante indiquant qu'il y a un contrôle de flux matériel en réception.
public final static int	FLOWCONTROL_RTSCTS_OUT	Constante indiquant qu'il y a un contrôle de flux matériel en émission.
public final static int	FLOWCONTROL_XONXOFF_IN	Constante indiquant qu'il y a un contrôle de flux logiciel en réception.
public final static int	FLOWCONTROL_XONXOFF_OUT	Constante indiquant qu'il y a un contrôle de flux logiciel en émission.

Abstraction série : classe SerialPort (fin)

Méthodes de la classe javax.comm.SerialPort		
public abstract void	setSerialPortParams(int r, int d, int s, int p)	Configuration du port de communication (par défaut : 9600 bauds, 8 bits de données, 1 bit de stop pas de parité) en utilisant les constantes. Soulève <i>UnsupportedCommOperationException</i> .
public abstract void	setFlowControlMode(int mode)	Configuration du contrôle de flux en utilisant les constantes. Soulève <i>UnsupportedCommOperationException</i> .
public abstract void	sendBreak(int t)	Envoi d'un signal « break » d'une durée de t ms.
public abstract int	getBaudRate()	Obtention du débit.
public abstract int	getDataBits()	Obtention du nombre de bits de données.
public abstract int	getStopBits()	Obtention du nombre de bits de stop.
public abstract int	getParity()	Obtention de la configuration de la parité.
public abstract int	getFlowControlMode()	Obtention de la configuration du contrôle de flux.
public abstract void	setDTR(boolean dtr)	Positionnement du DTR.
public abstract void	setRTS(boolean rts)	Positionnement du RTS.
public abstract boolean	isDTR()	Obtention de l'état du DTR.
public abstract boolean	isRTS()	Obtention de l'état du RTS.
public abstract boolean	isCTS()	Obtention de l'état du CTS.
public abstract boolean	isDSR()	Obtention de l'état du DSR.
public abstract boolean	isRI()	Obtention de l'état du RI.
public abstract boolean	isCD()	Obtention de l'état du CD.

Application exemple

- L'**application 1** :
 - Prend en **paramètres** :
 - Un **nom symbolique de port** de communication série
 - Une chaîne de caractères ASCII
 - **Envoie la chaîne de caractères** (terminée par un saut de ligne) sur le **port série**
- L'**application 2** :
 - Prend en **paramètre** un **nom symbolique de port** de communication série
 - **Reçoit et affiche** une chaîne de caractères ASCII (terminée par un saut de ligne) sur le port série

Application exemple (suite)

```
import java.io.*;
import gnu.io.comm.*;
public class SerialSender
{
    public static void main(String[] args)
    {
        // args[0] : nom symbolique du port
        // args[1] : chaîne à envoyer

        CommPortIdentifier portID = null;
        try {portID = CommPortIdentifier.getPortIdentifier(args[0]);}
        catch (NoSuchPortException e1) {...}
        SerialPort port = null;
        try {port = (SerialPort) portID.open("SerialSender", 2000);}
        catch (PortInUseException e2) {...}
        try
        {
            port.setSerialPortParams(9600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
                                     SerialPort.PARITY_NONE);
            port.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
        }
        catch (UnsupportedCommOperationException e3) {...}
        try {port.getOutputStream().write((args[1]+"\\n").getBytes("US-ASCII"));}
        catch (IOException e4) {...}
    }
}
```

Application exemple (fin)

```
import java.io.*;
import gnu.io.*;
public class SerialReceiver
{
    public static void main(String[] args)
    {
        // args[0] : nom symbolique du port
        CommPortIdentifier portID = null;
        try {portID = CommPortIdentifier.getPortIdentifier(args[0]);}
        catch (NoSuchPortException e1) {...}
        SerialPort port = null;
        try {port = (SerialPort) portID.open("SerialReceiver", 2000);}
        catch (PortInUseException e2) {...}
        try
        {
            port.setSerialPortParams(9600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
                                     SerialPort.PARITY_NONE);
            port.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
        }
        catch (UnsupportedCommOperationException e3) {...}
        try
        {
            BufferedReader br =
                new BufferedReader(new InputStreamReader(port.getInputStream(), "US-ASCII"));
            System.out.println(br.readLine());
        }
        catch (IOException e4) {...}
    }
}
```

Communication série événementielle

Méthodes de la classe <code>javax.comm.SerialPort</code>		
<code>public abstract void</code>	<code>addEventListener(SerialEventListener l)</code>	Ajout d'un auditeur d'événements de communication. Seuls les événement choisis sont notifiés (aucun par défaut). Il ne peut y avoir qu'un seul auditeur. Soulève <i><code>TooManyListenerException</code></i> .
<code>public abstract void</code>	<code>removeEventListener()</code>	Retrait de l'auditeur d'événements de communication (automatique à la fermeture du port).
<code>public abstract void</code>	<code>notifyOnDataAvailable(boolean s)</code>	Ajout/retrait de la notification de l'événement « données disponibles » de la liste de événements notifiés.
<code>public abstract void</code>	<code>notifyOnOutputEmpty(boolean s)</code>	Idem précédente, pour l'événement « tampon d'écriture vide ».
<code>public abstract void</code>	<code>notifyOnCTS(boolean s)</code>	Idem précédente, pour l'événement « CTS modifié ».
<code>public abstract void</code>	<code>notifyOnDSR(boolean s)</code>	Idem précédente, pour l'événement « DSR modifié ».
<code>public abstract void</code>	<code>notifyOnRingIndicator(boolean s)</code>	Idem précédente, pour l'événement « RI modifié ».

Communication série événementielle (suite)

Méthodes de la classe <code>javax.comm.SerialPort</code>		
<code>public abstract void</code>	<code>notifyOnParityError(boolean s)</code>	<i>Ajout/retrait de la notification de l'événement « erreur de parité » de la liste de événements notifiés.</i>
<code>public abstract void</code>	<code>notifyOnFramingError(boolean s)</code>	<i>Idem précédente, pour l'événement « erreur de délimitation de trame ».</i>
<code>public abstract void</code>	<code>notifyOnCarrierDetect(boolean s)</code>	<i>Idem précédente, pour l'événement « CD modifié ».</i>
<code>public abstract void</code>	<code>notifyOnOverrunError(boolean s)</code>	<i>Idem précédente, pour l'événement « erreur de recouvrement » (octet perdu par recouvrement sur un tampon plein).</i>
<code>public abstract void</code>	<code>notifyOnBreakInterrupt(boolean s)</code>	<i>Idem précédente, pour l'événement « détection du signal break ».</i>

Méthodes de l'interface <code>javax.comm.SerialEventListener</code>		
<code>public void</code>	<code>serialEvent(SerialPortEvent e)</code>	<i>Méthode de callback permettant à l'auditeur de prendre en compte un événement sur le port.</i>

Communication série événementielle (fin)

Constantes de la classe javax.comm.SerialPortEvent		
public final static int	DATA_AVAILABLE	Identification du type d'événement « données disponibles ».
public final static int	OUTPUT_BUFFER_EMPTY	Identification du type d'événement « tampon d'écriture vide ».
public final static int	CTS	Identification du type d'événement « CTS modifié ».
public final static int	DSR	Identification du type d'événement « DSR modifié ».
public final static int	RI	Identification du type d'événement « RI modifié ».
public final static int	CD	Identification du type d'événement « CD modifié ».
public final static int	OE	Identification du type d'événement « erreur de recouvrement ».
public final static int	PE	Identification du type d'événement « erreur de parité ».
public final static int	FE	Identification du type d'événement « erreur de délimitation de trame ».
public final static int	BI	Identification du type d'événement « détection de signal break ».

Méthodes de la classe javax.comm.SerialPortEvent		
public int	getEventType()	Identification du type d'événement.
public boolean	getOldValue()	Obtention de l'état précédent, dans le cas où l'événement est lié à un changement d'état du signal.
public boolean	getNewValue()	Idem précédente, mais pour obtenir le nouvel état.

Application exemple

- L'**application** :
 - Prend en **paramètre** un **nom symbolique de port** de communication série
 - **Enregistre un auditeur d'événements de communication** sur ce port et **affiche les événements reçus**
 - **S'arrête** lorsqu'il reçoit l'événement « **détection de signal break** »

Application exemple (suite)

```
import java.io.*;
import gnu.io.*;
public class SerialEventDisplayer implements SerialPortEventListener
{
    public boolean alive;
    private SerialPort port;

    public SerialEventDisplayer(String p) throws NoSuchPortException, PortInUseException,
                                                TooManyListenersException
    {
        CommPortIdentifier portID = CommPortIdentifier.getPortIdentifier(p);
        this.port = (SerialPort) portID.open("SED", 2000);
        this.port.addEventListener(this);
        this.port.notifyOnBreakInterrupt(true);
        this.port.notifyOnCarrierDetect(true);
        this.port.notifyOnCTS(true);
        this.port.notifyOnDataAvailable(true);
        this.port.notifyOnDSR(true);
        this.port.notifyOnFramingError(true);
        this.port.notifyOnOutputEmpty(true);
        this.port.notifyOnOverrunError(true);
        this.port.notifyOnParityError(true);
        this.port.notifyOnRingIndicator(true);
        this.alive = true;
    }
    ...
}
```

Application exemple : communication série active (suite)

```
import java.io.*;
import gnu.io.*;
public class SerialEventDisplayer implements SerialPortEventListener
{
    ...
    public void serialEvent(SerialPortEvent e)
    {
        switch (e.getEventType())
        {
            case SerialPortEvent.BI :
                System.out.println("BI event !");
                this.alive = false;
                this.port.removeEventListener();
                this.port.close();
                return;
            case SerialPortEvent.CD :
                System.out.println("CD event !" + e.getOldValue() + "->" + e.getNewValue());
                return;
            case SerialPortEvent.CTS :
                System.out.println("CTS event !" + e.getOldValue() + "->" + e.getNewValue());
                return;
            ...
        }
    }
}
```

Application exemple (suite)

```
import java.io.*;
import gnu.io.*;
public class SerialEventDisplayer implements SerialPortEventListener
{
    ...
    public void serialEvent(SerialPortEvent e)
    {
        ...
        case SerialPortEvent.DATA_AVAILABLE :
            System.out.println("Data available !");
            try
            {
                InputStream is = this.port.getInputStream();
                while (is.available()>1) System.out.print(""+is.read()+" ");
                System.out.println(""+is.read());
            } catch (IOException e2) {}
            return;
        case SerialPortEvent.DSR :
            System.out.println("DSR event !" + e.getOldValue() + "->" + e.getNewValue());
            return;
        case SerialPortEvent.FE :
            System.out.println("Framing error !");
            return;
        case SerialPortEvent.OE :
            System.out.println("Overrun error !");
            return;
        ...
    }
}
```

Application exemple (fin)

```
import java.io.*;
import gnu.io.*;
public class SerialEventDisplayer implements SerialPortEventListener
{
    ...
    public void serialEvent(SerialPortEvent e)
    {
        ...
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY :
            System.out.println("Output buffer empty");
            return;
        case SerialPortEvent.PE :
            System.out.println("Parity error !");
            return;
        case SerialPortEvent.RI :
            System.out.println("RI event !" + e.getOldValue() + "->" + e.getNewValue());
            return;
        }
    }

    public static void main(String[] args)
    {
        SerialEventDisplayer sed = null;
        try {sed = new SerialEventDisplayer(args[0]);} catch (Exception e) {...}
        while (sed.alive)
            try {Thread.sleep(1000);} catch (InterruptedException e) {}
    }
}
```

Fin !

