Les composants de la bibliothèque Windowsbuilder

1) Le composant JButton

symbole



Bien souvent lié à l'événement actionperformed

2) Le composant JToggleButton

Cette classe définit un bouton à deux états : c'est la classe mère des composants JCheckBox et JRadioButton.

symbole



Méthodes

La méthode isSelected() permet de connaître l'état enclenché du bouton.

```
boolean isSelected()
```

3) Le composant RadioButton et Buttongroup

Un objet de type **JRadioButton** représente un bouton radio d'un groupe de boutons. A un instant donné, un seul des boutons radio associés à un même groupe peut être sélectionné. La classe JRadioButton hérite de la classe AbstractButton.

symbole



méthodes

La méthode isSelected() permet de savoir si le bouton est sélectionné ou non.

boolean isSelected()

Un groupe de boutons radio est encapsulé dans un objet de **type ButtonGroup**.
Il faut ajouter tous les JRadioButton du groupe en utilisant la méthode add() de la classe ButtonGroup. Lors de la sélection d'un bouton, c'est l'objet de type ButtonGroup qui se charge de déselectionner le bouton précédemment sélectionné dans le groupe.

```
private ButtonGroup bt= new ButtonGroup();
JRadioButton radiobutton1 = new JRadioButton("radio 1");
JRadioButton radiobutton2 = new JRadioButton("radio 2");
panel_1.add(radiobutton1);
panel_1.add(radiobutton2);
bt.add(radiobutton1);
bt.add(radiobutton2);
```

4) Le composant JTextfield

symbole



Vous pouvez utiliser les composants JText Field et JTextArea pour collecter les entrées de texte. Un *champ de texte* n'accepte qu'une ligne de texte, et les *zones de texte* peuvent en accueillir plusieurs. Ces deux classes dérivent d'une classe appelée JTextComponent.

Méthodes

javax. swing.text.JTextComponent

✓	void setText(String t)

Modifie le texte d'un composant texte.

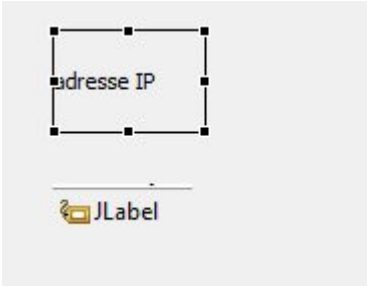
Paramètre : t Le nouveau texte.

✓	String getText ()

Renvoie le texte contenu dans le composant texte.

5) Le composant JLabel

symbole

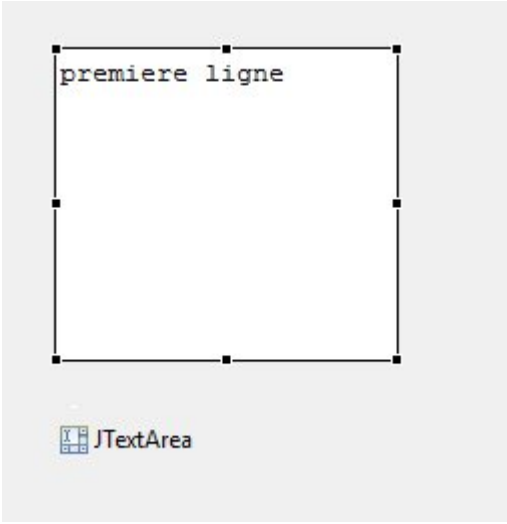


Méthodes identiques au composant JTextfield
Pour changer la couleur d'un texte en vert on se sert de la méthode setForeground qui reçoit comme paramètre un objet statique du type Color :

```
lblLed.setForeground(Color.Green);
```

6) Le composant JTestarea et JAreascrollpane

symbole



JTextArea (String text , int rows, int cols)

Construit une nouvelle zone de texte avec un texte initial.

Paramètres :

Le texte initial.

Le nombre de lignes.

Le nombre de colonnes.

méthodes

Si le texte entré dépasse la capacité d'affichage de la zone de texte, le reste du texte est coupé. Pour éviter que des longues lignes ne soient tronquées, vous pouvez activer le retour automatique à la ligne avec la méthode :

`setLinewrap(boolean saut)`

Pour afficher une nouvelle ligne

```
append (string texte)
```

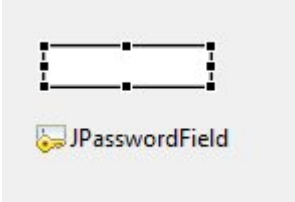
Il est utile de disposer d'une barre de defilement horizontal ou vertical. Il nous faut inclure dans un panneau le composant JAreascrollpane qui intégrera le composant JTextArea. Sous windowsbuilder on commence à dessiner le composant JTextArea puis on modifie et on rajoute les lignes de code suivantes :

```
JScrollPane areaScrollPane = new JScrollPane();
areaScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
areaScrollPane.setBounds(10, 104, 249, 222);//areaScrollPane englobe textArea
contentPane.add(areaScrollPane);
textArea = new JTextArea();
areaScrollPane.setViewportView(textArea);// integration du JTextArea dans le
scrollPane
textArea.setLineWrap(true);
textArea.setEditable(false);
```

7) Le composant JPasswordField

C'est une simple zone de saisie de texte ou le texte est caché par le caractère étoile *.

symbole



méthodes

String getText ()

Renvoie le texte contenu dans le composant texte.

La classe Timer

Swing possède une classe temporisateur intégrée (Timer) qui est facile à utiliser. Vous pouvez construire un temporisateur en fournissant un objet d'une classe qui implémente l'interface ActionListener (permet de préciser l'objet qui est en écoute d'un événement particulier comme ici le temps écoulé), et le délai de temporisation en millisecondes. Quand le temps programmé est écoulé un événement est déclenché.

La méthode constructeur.

```
Timer(int délai, ActionListener écouteur);
```

Cette méthode Crée un nouveau temporisateur qui envoie des événement temporels à un écouteur.

délai : le délai, en milliseconde, entre deux événement.
écouteur : écouteur d'action qui doit être averti lorsque le délai est écoulé (le plus souvent la fenêtre).

Autres méthodes

```
void start();
```

Lance le temporisateur. Après cet appel, le chronomètre commence à envoyer des événements à son écouteur d'action.

```
void stop();
```

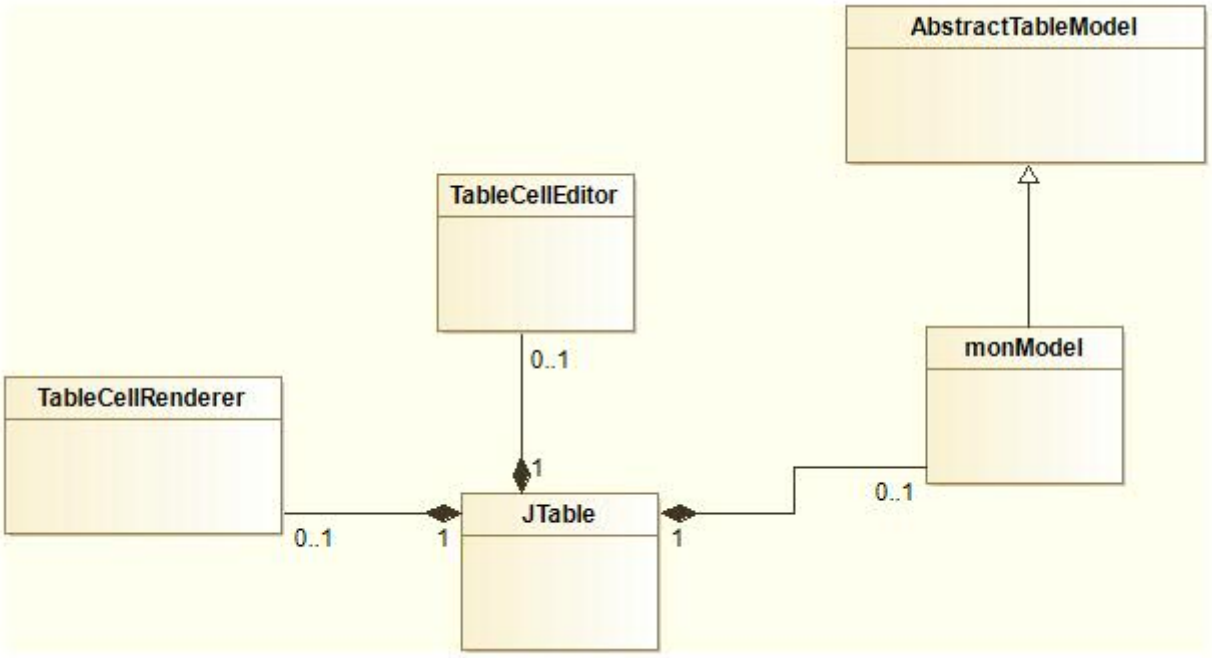
Stoppe le temporisateur. Après cet appel, le chronomètre arrête d'envoyer des événements à son écouteur d'action.

La classe JTable

Le composant JTable permet d'afficher des tables de données, en permettant éventuellement l'édition de ces données. Un JTable ne contient pas ses données mais les Obtient à partir d'un tableau d'objets à 2 dimensions, ou à partir d'un modèle de données. Le rendu et le mode d'édition des cellules de la table peuvent être modifiés.

1) Création d'un modèle de table

Le composant JTable est un «visualisateur» qui prend les données à afficher dans un modèle monModel qui dérive de la classe abstraite AbstractTableModel



Le JTable utilise différents concepts de Swing :

- Un modèle pour stocker les données. Un JTable utilise une classe implémentant **TableModel**. Nous verrons plus loin comment spécifier un modèle pour un JTable.
- Un *renderer* pour le rendu des cellules. On peut spécifier un **TableCellRenderer** pour chaque classe de données.
- Un éditeur pour l'édition du contenu d'une cellule. On peut spécifier un **TableCellEditor** pour chaque classe de données.

2) Exemple de table avec modeledynamique

Dans l'exemple ci-dessous on va créer la classe d'un modèle et redéfinir les méthodes de la classe abstraite AbstractTableModel . Le tableau est le suivant :

Prénom	nom	age
durand	pierre	30
martin	nicolas	25

```
public class monModel extends AbstractTableModel {

private final Object[][] donnees;

    private final String[] entetes = {"Prénom", "Nom", "age"}; // les noms des colonnes

    public monModel () {
      super();

       donnees = new Object[][]{// le contenu des colonnes dans un tableau bidimensionnel
              {"durand", "pierre", 30},
              {"martin", "nicolas", 25},
                                };
    }

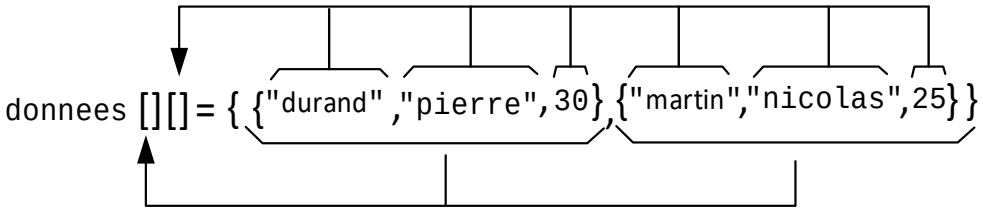
    public int getRowCount() {
        return donnees.length;
    }

    public int getColumnCount() {
        return entetes.length;
    }

    public String getColumnName(int columnIndex) {
        return entetes[columnIndex];
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        return donnees[rowIndex][columnIndex];
    }
}
```

On peut représenter le tableau donnees comme un tableau à deux dimensions dont le premier indice correspond aux numéros de lignes et le deuxième indice les numéros des colonnes :



Pour créer un objet m1 de la classe JTable dans la méthode main on peut écrire :

```
monModel m1 =new MonModele();
 JTable table  = new JTable(m1);
```

Si le tableau est trop grand on peut associer un JscrollPane

```
JScrollPane areaScrolltable = new JScrollPane(table);
areaScrolltable.setBounds(286, 103, 403, 328);
areaScrolltable.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
contentPane.add(areaScrolltable);
```

Pour ajouter ou supprimer des lignes et rendre dynamique notre tableau, il faut bien évidemment utiliser une structure de données qui soit dynamique. Le tableau n'est pas du tout adapté, On doit utiliser dans ce cas **une ArrayList**.

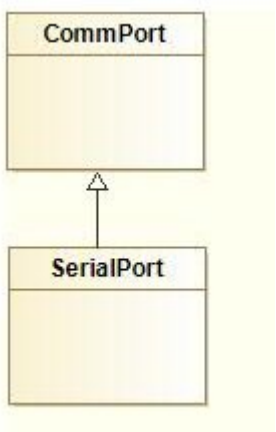
La classe RXTXComm

1) Présentation

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 est multiplateforme Windows, MacOs X, Linux-x86 : architectures 32 ou 64 bits
Distributions Linux embarquées ARM ….(' 30 plates-formes supportées)
Les classes de RxTx sont définies dans le paquetage gnu.io

2) Les différentes classes de gestion du port série

Pour la communication avec le port série on dispose de la classe **SerialPort** :



Méthodes de la classe javax.comm.CommPort		
public String	getName()	*Obtention du nom symbolique associé au port de communication.*
public void	**close()**	***Libération du port de communication.***
public abstract InputStream	**getInputStream()**	***Obtention du flux de lecture binaire Sur leport de communication. Soulève IOException.***

public abstract OutputStream	**getOutputStream()**	***Obtention du flux d'écriture binaire Sur leport de communication. Soulève IOException.***
public abstract void	enableReceiveThreshlod(int t)	*Activation du seuil de réception (si sup porté par le driver). Toute lecture sur le flux sera bloquante jusqu'à ce que t oc- tets de données aient été reçus.* *Soulève UnsupportedCommOperationException.*
public abstract void	disableReceiveThreshlod()	*Désactivation du seuil de réception.*
public abstract boolean	isReceiveThreshlodEnabled()	*Test d'activation du seuil de réception.*
public abstract int	getReceiveThreshlod()	*Obtention du seuil de réception.*

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 UnsupportedCommOperationException.*
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 blo- quante 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 UnsupportedCommOperationException.*
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.*

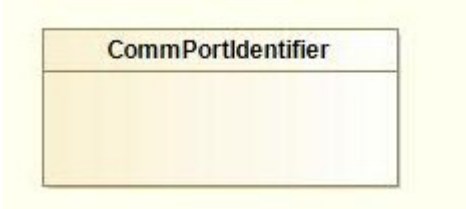
Constantes de la classe javax.comm.SerialPort		
Public Final static int	DATABITS_5	*Constante indiquant que le nombre de bits de données est de 5.*
Public Final static int	DATABITS_6	*Constante indiquant que le nombre de bits de données est de 6.*
Public Final static int	DATABITS_7	*Constante indiquant 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.*

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 UnsupportedCommOperationException.***
public abstract void	**setFlowControlMode(int mode)**	***Configuration du contrôle en utilisant les constantes. UnsupportedCommOperationException.***
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 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	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.*

Pour la communication avec le système et l'ouverture des ports, on dispose de la classe **CommPortIdentifier**



Méthodes de la classe javax.comm.CommPortIdentifier		
public static Enumeration	getPortIdentifiers()	*Obtention de tous les identifiants de ports.*
public static CommPortIdentifier	**getPortIdentifier(String name)**	***Obtention de l'identifiant de port De nom symbolique name. Soulève NoSuchPortException.***
public static CommPortIdentifier	getPortIdentifier(CommPort port)	*Obtention de l'identifiant de port associé au port de communication référencé par port. Soulève UnknownHostException.*
public String	getName()	*Obtention du nom symbolique du port.*
public int	getType()	*Obtention du type du port (cf. constantes).*
public boolean	**isCurrentlyOwned()**	***Test d'occupation du port. Retourne true si le port est occupé par une application Java.***
public String	getCurrentOwner()	*Obtention du nom de l'application oc cupant le port (si le port est occupé par une application Java.*

Constantes de la classe javax.comm.CommPortIdentifier	
PORT_SERIAL	*Constante identifiant un port comme étant de type série.*

Méthodes de la classe		
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 PortInUseException.*
public CommPort	open(FileDescriptor fd)	*Idem précédente mais en utilisant un descripteur de fichier (si la plate-forme supporte cette fonctionnalité). Soulève UnsupportedCommOperationException.*
public void	addPortOwnershipListener (CommPortOwnershipListener l)	*Ajout d'un auditeur d'événements tion/libération de port ».*
public void	removePortOwnershipListener (CommPortOwnershipListener l)	*Retrait d'un auditeur d'événements « occupation/libération de port ».*

Pour la gestion des événements :



Les principaux types de conversion de format en JAVA

Conversion d'un type primitif en une chaîne

Nous avons déjà vu comment l'opérateur "+" effectuait un "formatage" en convertissant n'importe quel type primitif en une chaîne. Vous pouvez effectuer directement de telles conversions, en utilisant la méthode valueOf de la classe String.

La méthode valueOf peut recevoir un argument de type quelconque et pas seulement de type entier.

```
int n = 427;
String ch = String.valueOf(n);
// fourni une chaîne obtenue par formatage de
// la valeur contenue dans n, soit ici "427"
ch = "" + n;
// utilisation artificielle d'une chaîne vide
// pour pouvoir recourir à l'opérateur "+"
```

Conversion d'une chaîne en un type primitif

On peut réaliser les conversions inverses des précédentes. Il faut alors recourir à une méthode de la classe enveloppe associée au type primitif : Integer pour int, Double pour double, etc...

Par exemple, pour convertir une chaîne en un entier de type int, on utilisera la méthode statique parseInt de la classe enveloppe Integer, comme ceci :

```
String ch = "3587"
int n = Integer.parseInt(ch);
```

```
Byte.parseByte,
Short.parseShort,
Integer.parseInt,
Long.parseLong,
Float.parseFloat,
Double.parseDouble
```

D'une manière générale, nous disposons des méthodes pour chaque type primitif comme cela est précisé ci-dessus.

Conversion d'un tableau de byte en une chaine de caractère

```
private static String toHexadecimal(byte[] digest){
    String hash = "";
//for(byte aux : digest) {
    for(int i=0; i<digest.length;i++) {
      byte aux=digest[i];
    int b = aux & 0xff;
    if (Integer.toHexString(b).length() == 1) hash += "0";
    hash += Integer.toHexString(b);
}
return hash;
}
```