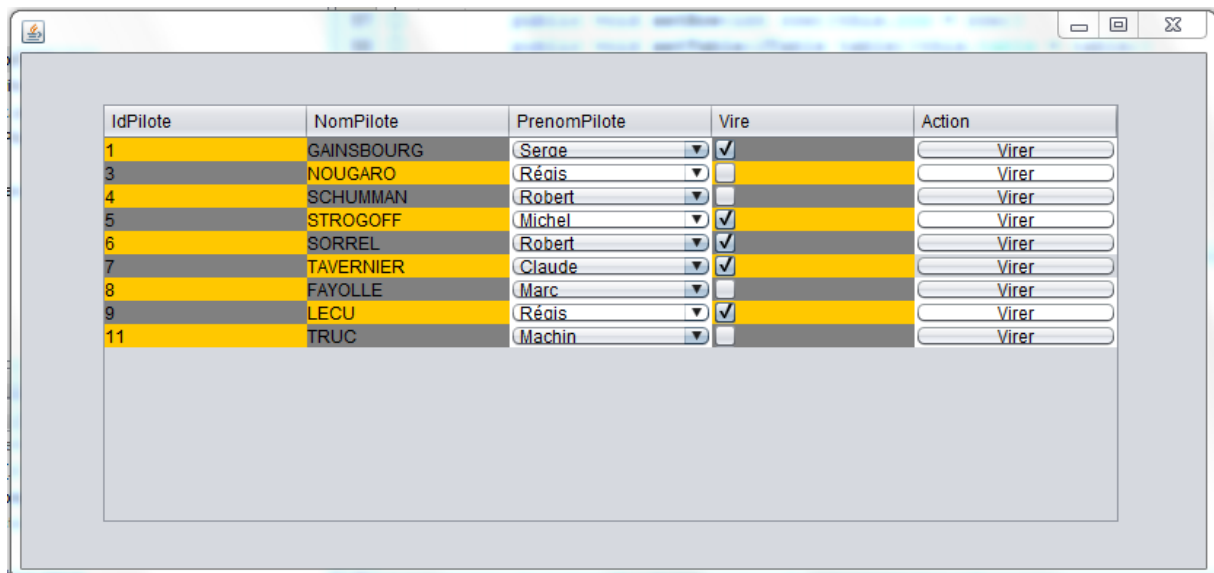


# Enrichir sa JTable avec des composants et mise à jour de la base de données associée

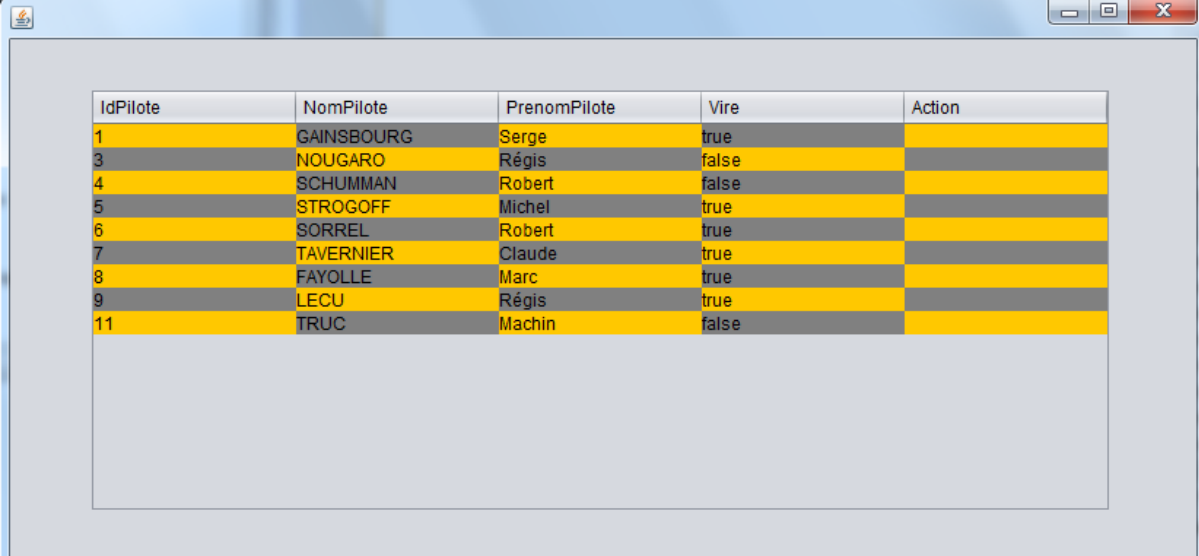


The screenshot shows a Java Swing window with a JTable. The table has five columns: IdPilote, NomPilote, PrenomPilote, Vire, and Action. The rows are numbered 1 to 11. The 'Vire' column contains checkboxes, and the 'Action' column contains buttons labeled 'Virer'.

IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBURG	Serge	<input checked="" type="checkbox"/>	Virer
3	NOUGARO	Régis	<input type="checkbox"/>	Virer
4	SCHUMMAN	Robert	<input type="checkbox"/>	Virer
5	STROGOFF	Michel	<input checked="" type="checkbox"/>	Virer
6	SORREL	Robert	<input checked="" type="checkbox"/>	Virer
7	TAVERNIER	Claude	<input checked="" type="checkbox"/>	Virer
8	FAYOLLE	Marc	<input type="checkbox"/>	Virer
9	LECU	Régis	<input checked="" type="checkbox"/>	Virer
11	TRUC	Machin	<input type="checkbox"/>	Virer

Ce travail suppose que vous sachiez créer une table, l'associer avec un modèle de données. Comme nous allons rendre des colonnes éditables, il faut empêcher que l'utilisateur ne réordonne les colonnes ( dans la propriété TableHeader, vous décochez reordering allowed ).

Voici la table initiale :



IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBOURG	Serge	true	
3	NOUGARO	Régis	false	
4	SCHUMMAN	Robert	false	
5	STROGOFF	Michel	true	
6	SORREL	Robert	true	
7	TAVERNIER	Claude	true	
8	FAYOLLE	Marc	true	
9	LECU	Régis	true	
11	TRUC	Machin	false	

# Une colonne de JCheckBox

Pour insérer une colonne de JCheckBox dans la JTable, il faut procéder en 4 étapes :

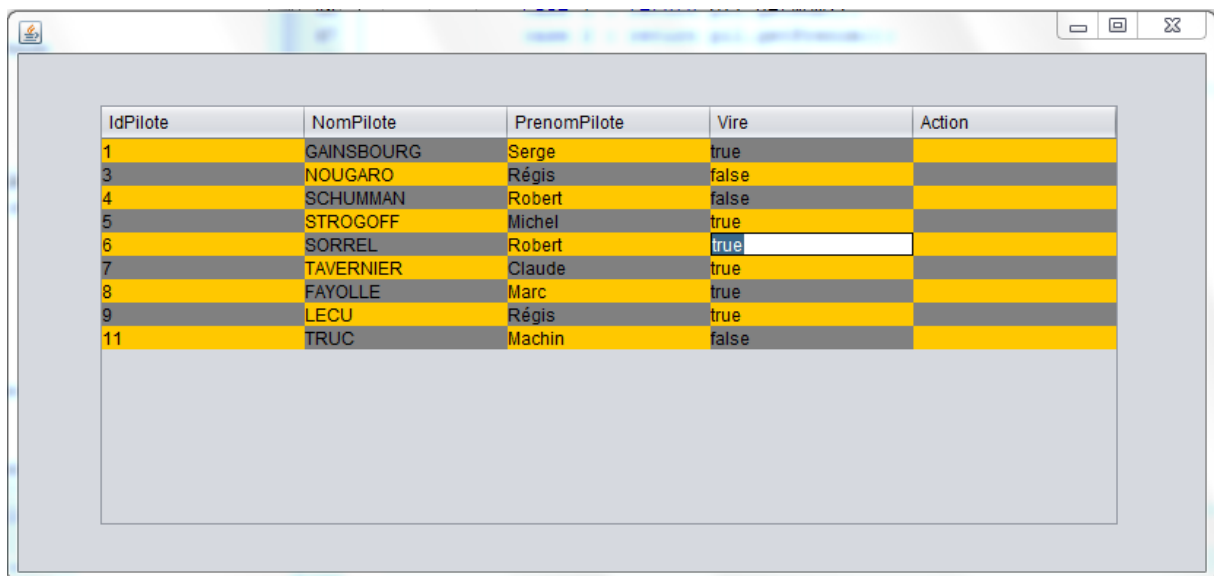
## Rendre la colonne concernée éditable :

Pour pouvoir agir sur le JCheckBox, il faut rendre la colonne concernée modifiable. Ceci se fait dans le model de données de la table, par la méthode `isCellEditable`.

@Override

```
public boolean isCellEditable(int rowIndex, int columnIndex)
{
    return columnIndex >= 2; // ici les trois ( puis quatre ) dernières colonnes sont éditables
}
```

La fenêtre devient éditable sur les colonnes concernées.



IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBOURG	Serge	true	
3	NOUGARO	Régis	false	
4	SCHUMMAN	Robert	false	
5	STROGOFF	Michel	true	
6	SORREL	Robert	true	
7	TAVERNIER	Claude	true	
8	FAYOLLE	Marc	true	
9	LECU	Régis	true	
11	TRUC	Machin	false	

## Donner un type Boolean à la colonne concernée :

En typant la colonne en Boolean, la colonne prendra l'apparence d'un JCheckBox automatiquement. Cela se fait dans le model de données de la table, en polymorphant la méthode `getColumnClass`.

@Override

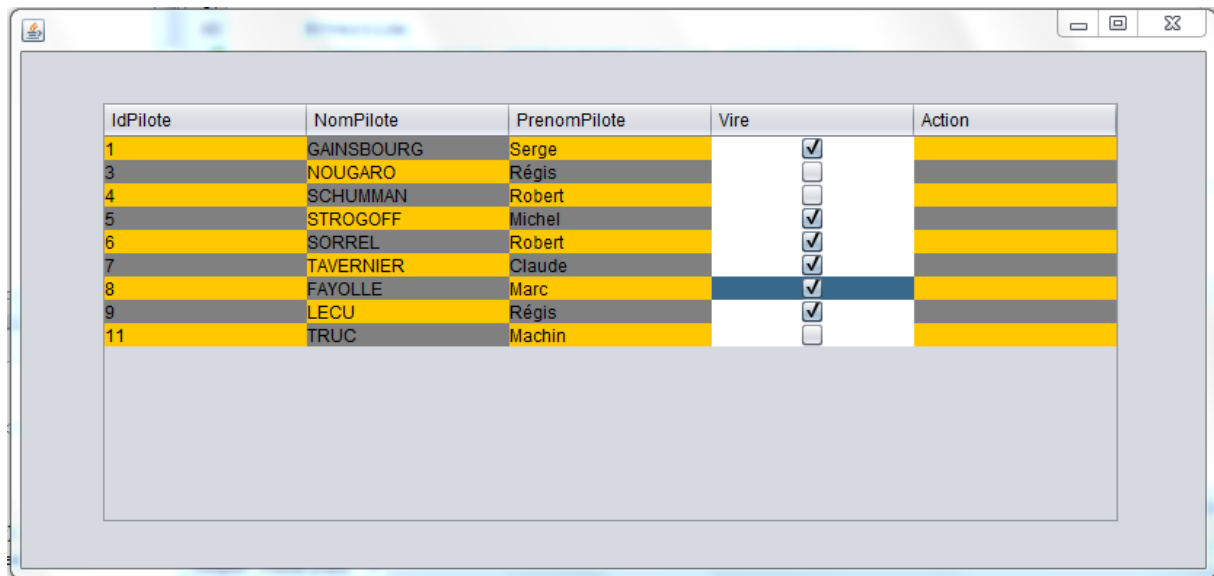
```
public Class<?> getColumnClass(int columnIndex)
```

```

{
    Class clas = String.class;
    if (columnIndex == 3) // colonne de notre JCheckBox
    {
        clas = Boolean.class; // ainsi le JCheckBox est pris en compte pour l'apparence de la colonne
    }
    return clas;
}

```

Maintenant notre table contient des cases à cocher sur la colonne concernée.



IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBURG	Serge	<input checked="" type="checkbox"/>	
3	NOUGARO	Régis	<input type="checkbox"/>	
4	SCHUMMAN	Robert	<input type="checkbox"/>	
5	STROGOFF	Michel	<input checked="" type="checkbox"/>	
6	SORREL	Robert	<input checked="" type="checkbox"/>	
7	TAVERNIER	Claude	<input checked="" type="checkbox"/>	
8	FAYOLLE	Marc	<input checked="" type="checkbox"/>	
9	LECU	Régis	<input checked="" type="checkbox"/>	
11	TRUC	Machin	<input type="checkbox"/>	

Ces cases à cocher ne sont pas cliquables, et elles sont un peu pâlottes.

## Modifier le modèle de données de la table sur une action de l'opérateur

Quand une cellule est modifiée, la méthode `setValueAt` du modèle de données est appelée. Cette méthode change la donnée correspondante du modèle de données, et lance un événement pour signaler que le contenu du modèle de données a changé. Ainsi la `JTable` sera t'elle mise à jour directement.

@Override

```

public void setValueAt(Object aValue, int rowIndex, int columnIndex)
{
    if (aValue instanceof Boolean && columnIndex == 3)
    {

```

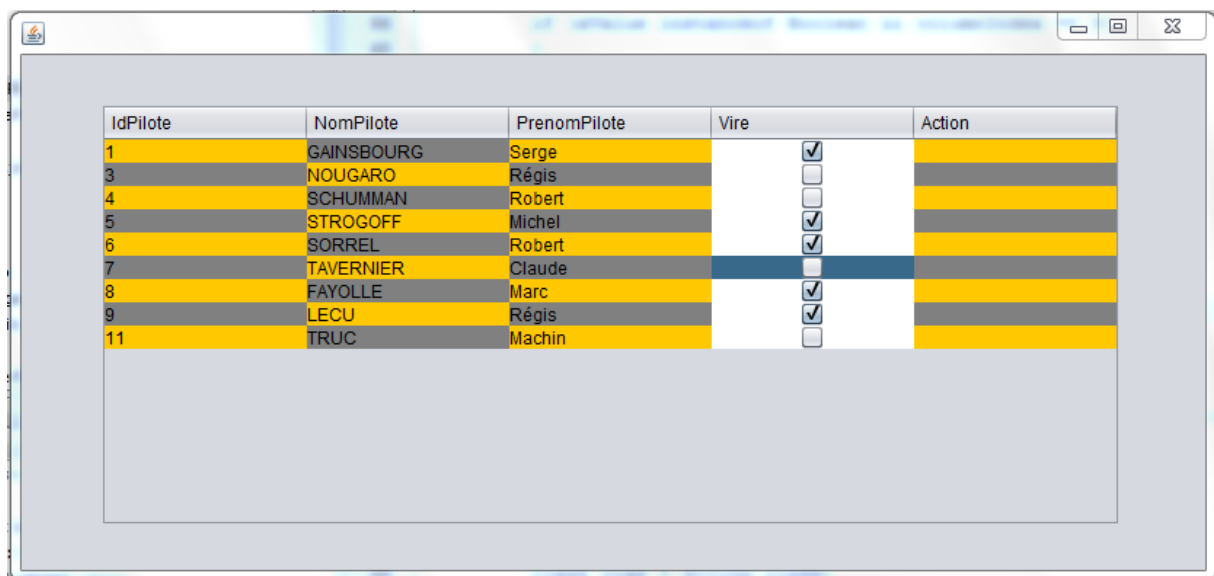
```

//Récupère un pilote dans la JTable
Pilote pil = listePilote.get(rowIndex);
// Met à jour le modèle de données

pil.setVire((Boolean) aValue);
//Prévient tout le monde du changement
fireTableCellUpdated(rowIndex, columnIndex);
}

```

Notre table prend bien en compte les modifications de l'utilisateur.



IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBURG	Serge	<input checked="" type="checkbox"/>	
3	NOUGARO	Régis	<input type="checkbox"/>	
4	SCHUMMAN	Robert	<input type="checkbox"/>	
5	STROGOFF	Michel	<input checked="" type="checkbox"/>	
6	SORREL	Robert	<input checked="" type="checkbox"/>	
7	TAVERNIER	Claude	<input type="checkbox"/>	
8	FAYOLLE	Marc	<input checked="" type="checkbox"/>	
9	LECU	Régis	<input checked="" type="checkbox"/>	
11	TRUC	Machin	<input type="checkbox"/>	

Ces modifications ne sont pas encore prises en compte par la base de données ( voir le dernier chapitre de ce document) et les couleurs de la colonne sont toujours très pâles.

## Redonner au fond du JCheckBox des couleurs

Pour cela il faut créer un rendu pour les JCheckBox, et associer ce rendu à la colonne des JCheckBox.

Nous créons une classe pour le rendu des JCheckBox :

```

public class RenduCheck extends JCheckBox implements TableCellRenderer
{
    @Override
    public Component getTableCellRendererComponent(JTable table, Object check, boolean
isSelected, boolean hasFocus,int row, int column)
    {
        setHorizontalAlignment(JCheckBox.CENTER); // centre le JCheckBox
    }
}

```

```

//Permet d'afficher un bouton activé
this.setSelected((Boolean)check);
this.setOpaque(true);
    if ((row + column)%2 == 0)
    {
        this.setBackground(Color.ORANGE);
    }
    else
    {
        this.setBackground(Color.GRAY);
    }
    return this;
}
}

```

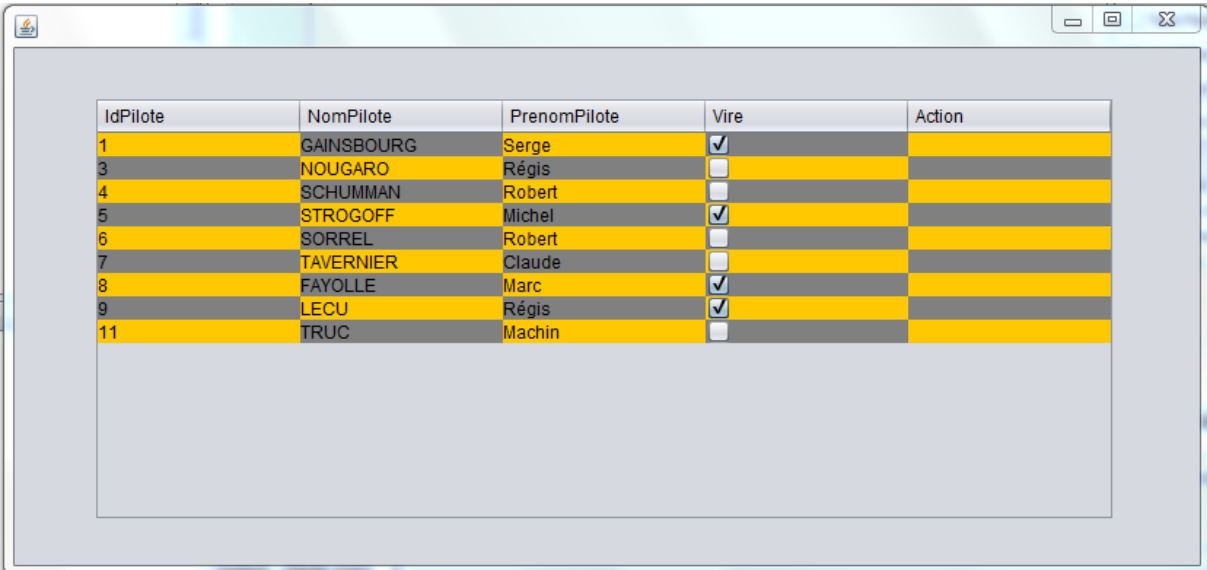
Puis nous ajoutons le code suivant au Customize Code de la JTable :

```

tableAvion.setDefaultRenderer(String.class, new RenduTablePilote());
tableAvion.setDefaultRenderer (Boolean.class, new RenduCheck());

```

Voici notre table qui a retrouvé toutes ses couleurs:



IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBOURG	Serge	<input checked="" type="checkbox"/>	
3	NOUGARO	Régis	<input type="checkbox"/>	
4	SCHUMMAN	Robert	<input type="checkbox"/>	
5	STROGOFF	Michel	<input checked="" type="checkbox"/>	
6	SORREL	Robert	<input type="checkbox"/>	
7	TAVERNIER	Claude	<input type="checkbox"/>	
8	FAYOLLE	Marc	<input checked="" type="checkbox"/>	
9	LECU	Régis	<input checked="" type="checkbox"/>	
11	TRUC	Machin	<input type="checkbox"/>	

Sur l'ensemble des images de la JTable, les JCheckBox ne sont pas centrés, mais en fait ils le sont ( bug corrigé le 28/08/2015 )

Les modifications ne sont pas encore prises en compte par la base de données ( voir le dernier chapitre de ce document, les répercussions sur la base étant les mêmes dans tous les exemples).

## Une colonne de JComboBox

---

Pour insérer une colonne de JComboBox, il faut procéder en 3 étapes :

### Rendre la colonne de JComboBox éditable

Ceci a déjà été fait au chapitre précédent.

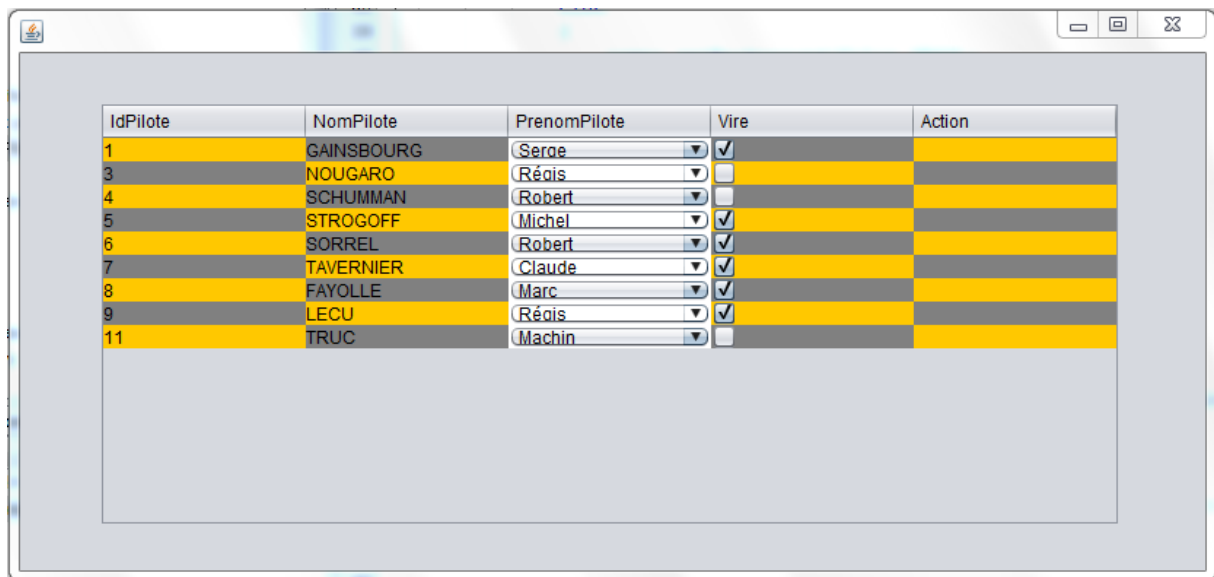
### Donner une apparence de JComboBox aux cellules de la colonne

Il va falloir créer un rendu de JComboBox aux cellules de la colonne concernée. Il faut pour cela travailler sur le CellRenderer de notre JTable. Nous allons donc modifier la méthode `getTableCellRendererComponent` de notre classe `RenduTablePilote` qui donne le rendu aux cellules de la JTable.

```
// si c'est la colonne des prénoms, je mets un combobox avec la liste de tous les prénoms
if (column == 2)
{
    compo = new JComboBox();
    ArrayList<String> lp = PiloteManageur.listePrenom();
    for (String prenom : lp)
    {
        ((JComboBox) compo).addItem(prenom);
    }
    //Permet de sélectionner le prénom du pilote concerné
    ((JComboBox<String>)compo).setSelectedItem(value);
    table.getColumnModel().getColumn("PrenomPilote").setCellEditor(new DefaultCellEditor((JComboBox) compo));
}
```

Nous profitons ici du travail de la colonne de cases à cocher. En effet, nous avons déjà associé le rendu de la table à la table.

Voici l'aperçu de notre fenêtre :



IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBURG	Serge	<input checked="" type="checkbox"/>	
3	NOUGARO	Régis	<input type="checkbox"/>	
4	SCHUMMAN	Robert	<input type="checkbox"/>	
5	STROGOFF	Michel	<input checked="" type="checkbox"/>	
6	SORREL	Robert	<input checked="" type="checkbox"/>	
7	TAVERNIER	Claude	<input checked="" type="checkbox"/>	
8	FAYOLLE	Marc	<input checked="" type="checkbox"/>	
9	LECU	Régis	<input checked="" type="checkbox"/>	
11	TRUC	Machin	<input type="checkbox"/>	

La colonne des listes déroulantes est bien remplie, les listes déroulantes fonctionnent, mais les modifications ne sont pas répercutées sur la table.

## Modifier le modèle de données de la table sur une action de l'opérateur

Par chance, la JTable gère déjà les actions sur une cellule qui est un JComboBox. Donc quand nous choisissons un nouveau prénom pour le pilote, la méthode `setValueAt` de la classe `ModelTablePilote` est appelée automatiquement. Il ne nous reste plus qu'à programmer la modification de données dans le model.

@Override

```
public void setValueAt(Object aValue, int rowIndex, int columnIndex)
{
    if (aValue instanceof Boolean && columnIndex == 3)
    {
        //Récupère un pilote dans la JTable
        Pilote pil = listePilote.get(rowIndex);
        pil.setVire((Boolean) aValue);
        //Préviens tout le monde du changement
        fireTableCellUpdated(rowIndex, columnIndex);
    }
    else if ( columnIndex == 2)
    {

```



```

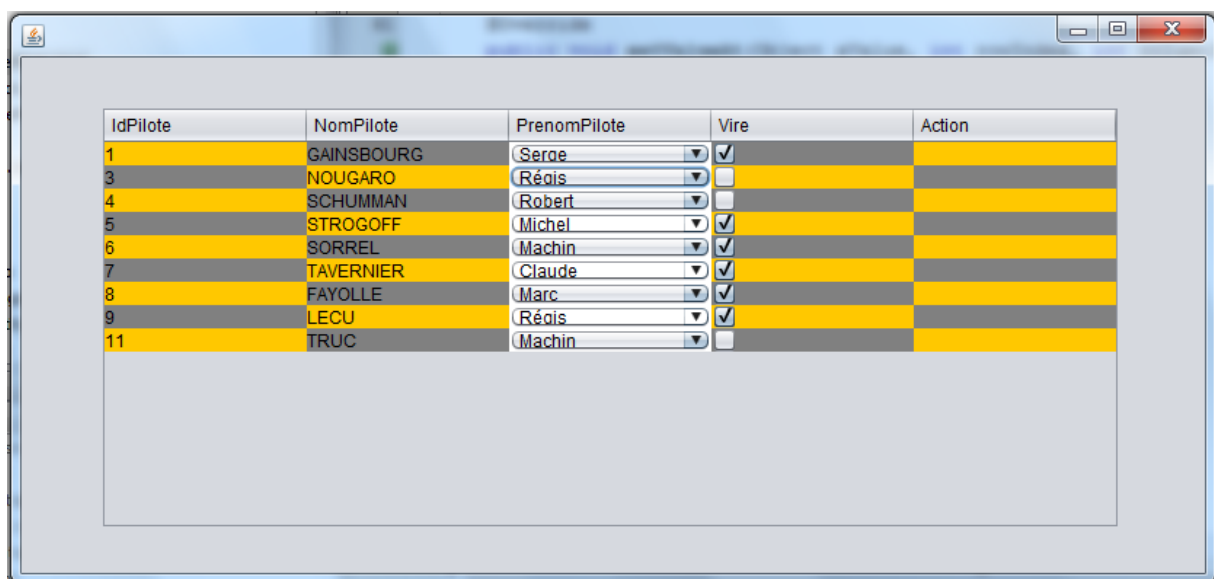
Pilote pil = listePilote.get(rowIndex);
pil.setPrenom((String)aValue);

//Préviens tout le monde du changement
fireTableCellUpdated(rowIndex, columnIndex);
}
}

```

Maintenant nous pouvons modifier le contenu de la table. Par contre la base de données n'a pas été mise à jour, ce travail est fait dans le dernier chapitre de ce document.

Voici notre table :



IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBOURG	Serge	<input checked="" type="checkbox"/>	
3	NOUGARO	Régis	<input type="checkbox"/>	
4	SCHUMMAN	Robert	<input type="checkbox"/>	
5	STROGOFF	Michel	<input checked="" type="checkbox"/>	
6	SORREL	Machin	<input checked="" type="checkbox"/>	
7	TAVERNIER	Claude	<input checked="" type="checkbox"/>	
8	FAYOLLE	Marc	<input checked="" type="checkbox"/>	
9	LECU	Régis	<input checked="" type="checkbox"/>	
11	TRUC	Machin	<input type="checkbox"/>	

Un des prénoms a été modifié, par la JComboBox.

# Une Colonne de JButton

---

Les deux composants que nous venons de voir sont assez simples, et le comportement attendu des cellules est prévisible. Nous allons maintenant nous attacher à des composants qui ont un comportement moins prévisible. Dans un premier temps nous allons créer une colonne de boutons, pour Virer un pilote. L'action attendue quand on clique sur le bouton n'est pas prévisible par la JTable, il va donc falloir programmer le comportement en plus d'avoir programmé la vue.

Pour insérer une colonne de JButton, il faut procéder en 4 étapes :

## Rendre la colonne de JButton éditable

Ceci a déjà été fait au chapitre précédent.

## Donner une apparence de JButton aux cellules de la colonne

Cela va être fait dans la classe de rendu de la table pilote ( RenduTablePilote ). Nous allons ajouter un nouveau cas pour donner une apparence de bouton aux cellules de la colonne.

Dans la méthode `getTableCellRendererComponent` nous ajoutons le code :

```
if ( column == 4)
{
    compo = new JButton("Virer");
}
```

Nous obtenons la vue suivante de la table :

IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBOURG	Serge	<input checked="" type="checkbox"/>	Virer
3	NOUGARO	Régis	<input type="checkbox"/>	Virer
4	SCHUMMAN	Robert	<input type="checkbox"/>	Virer
5	STROGOFF	Michel	<input checked="" type="checkbox"/>	Virer
6	SORREL	Robert	<input checked="" type="checkbox"/>	Virer
7	TAVERNIER	Claude	<input checked="" type="checkbox"/>	Virer
8	FAYOLLE	Marc	<input checked="" type="checkbox"/>	Virer
9	LECU	Régis	<input checked="" type="checkbox"/>	Virer
11	TRUC	Machin	<input type="checkbox"/>	Virer

Les boutons sont inutilisables, il va donc falloir leur associer un comportement.

## Création d'un comportement pour la colonne de boutons

Dans le rendu de cellule, nous avons construit un bouton, afin que la colonne de la JTable ait l'apparence d'une colonne de boutons. Nous allons créer un comportement pour la colonne de boutons, pour cela il faudra associer un bouton à la cellule pour gérer son comportement. En annexe de ce chapitre je vous mets comment créer un seul bouton pour une cellule pour gérer à la fois son apparence, et son comportement.

Le comportement d'une cellule est défini par un CellEditor. Le plus simple est de prendre un DefaultCellEditor, et de l'adapter à notre besoin.

```
public class EditeurBoutonTable extends DefaultCellEditor
{
```

```
    private JButton button;
```

```
    private final ButtonListener bListener = new ButtonListener();
```

```
// Constructeur avec une checkBox ( c'est par défaut ! ), ici la checkbox ne nous sert à rien !!!
```

```
public EditeurBoutonTable(JCheckBox checkBox)
```

```
{
```

```
    //Par défaut, ce type d'objet travaille avec un JCheckBox
```

```
    super(checkBox);
```

```
    //On crée à nouveau notre bouton
```

```
    button = new JButton("Virer");
```

```
    button.setOpaque(true);
```

```

//On lui attribue un listener
button.addActionListener(bListener);
}
// ici c'est le comportement qui est défini.
@Override
public Component getTableCellEditorComponent(JTable table, Object value,
        boolean isSelected, int row, int column)
{
    //On définit le numéro de ligne à notre listener
    bListener.setRow(row);
    //Ici on se moque du numéro de colonne
    //On passe aussi le tableau pour des actions potentielles
    bListener.setTable(table);
    //On renvoie le bouton
    return button;
}

// écouteur qui nous donne ce qu'il faut faire quand le bouton est appuyé
class ButtonListener implements ActionListener
{
    private int row;
    private JTable table;

    public void setRow(int row){this.row = row;}
    public void setTable(JTable table){this.table = table;}

    @Override
    public void actionPerformed(ActionEvent event)
    {
        // 1. recuperer le modele de donnees de table
        TableModel tm = table.getModel();
        // modifier le booléen virer à vrai sur cette ligne
        tm.setValueAt(Boolean.TRUE, row, 3); // je mets la valeur de la colonne 3 à vrai
    }
}

```

```

    }
}

}

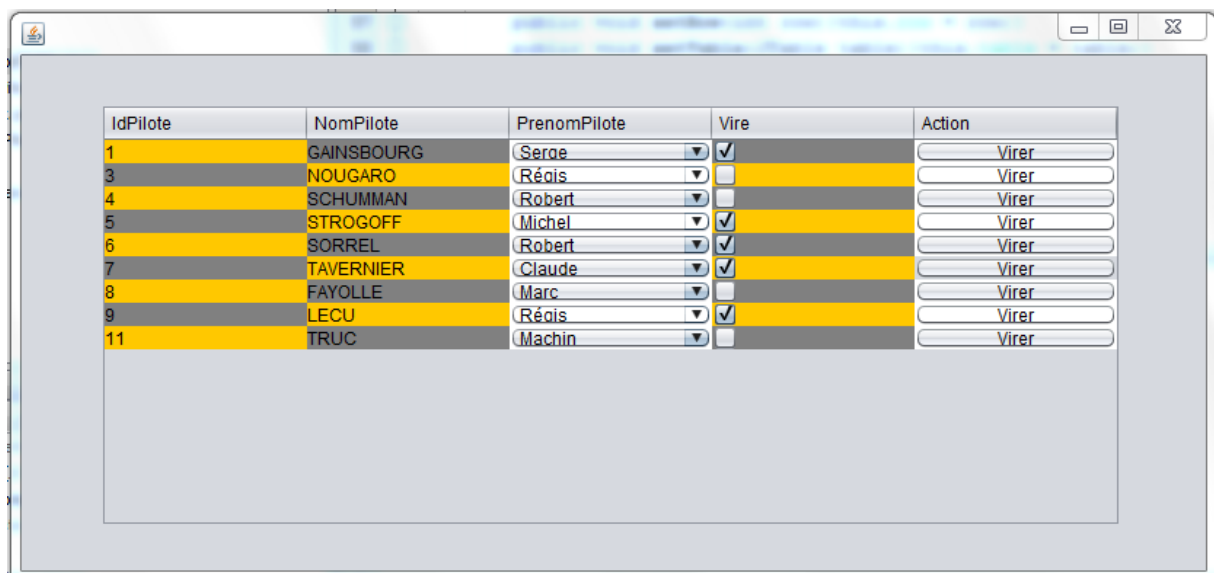
```

## Association du comportement à la colonne de boutons

Maintenant il va falloir associer ce comportement aux boutons de la colonne Action. Avec le concepteur d'interface, nous allons faire un customize code sur la table : nous allons ajouter le code suivant pour associer un comportement à la colonne action :

```
tableAvion.getColumnModel("Action").setCellEditor(new EditeurBoutonTable(new JCheckBox()));
```

Maintenant nos boutons sont actifs:



The screenshot shows a Java Swing window with a table containing pilot information. The table has five columns: IdPilote, NomPilote, PrenomPilote, Vire, and Action. The 'Action' column contains buttons labeled 'Virer'. The 'Vire' column contains checkboxes. The table is styled with alternating yellow and grey rows. The window has standard Mac OS X window controls (red, yellow, green buttons) in the top-left corner.

IdPilote	NomPilote	PrenomPilote	Vire	Action
1	GAINSBURG	Serae	<input checked="" type="checkbox"/>	Virer
3	NOUGARO	Régis	<input type="checkbox"/>	Virer
4	SCHUMMAN	Robert	<input type="checkbox"/>	Virer
5	STROGOFF	Michel	<input checked="" type="checkbox"/>	Virer
6	SORREL	Robert	<input checked="" type="checkbox"/>	Virer
7	TAVERNIER	Claude	<input checked="" type="checkbox"/>	Virer
8	FAYOLLE	Marc	<input type="checkbox"/>	Virer
9	LECU	Régis	<input checked="" type="checkbox"/>	Virer
11	TRUC	Machin	<input type="checkbox"/>	Virer

## Comment ajouter des boutons pour certaines lignes seulement

Pour ajouter des boutons sur certaines lignes seulement ( ici je ne vais mettre des boutons que sur les lignes où le pilote n'est pas encore viré), il faut procéder en trois étapes :

### Ne mettre en modifiable que les lignes nécessaires :

Nous allons reprendre la méthode isCellEditable du modèle de données :

```

@Override
public boolean isCellEditable(int rowIndex, int columnIndex)
{

```

```

return columnIndex == 2 || columnIndex == 3 || columnIndex == 5 ||
    (columnIndex == 4 && getValueAt(rowIndex, 3) == Boolean.FALSE);
// ici les 4 dernières colonnes sont éditables, et seulement les boutons de la 4
}

```

## Modifier le rendu des cellules pour mettre les boutons là où il faut

Ici il faut que le rendu soit un label si le pilote est viré, un bouton pour pouvoir le virer sinon. Nous modifions donc le rendu de la table :

```

@Override
public Component getTableCellRendererComponent(JTable table, Object value, boolean
isSelected, boolean hasFocus, int row, int column) {
    Component compo = null;
    if (column != 2 || (column == 4 && table.getModel().getValueAt(row, 3) == Boolean.TRUE))
    {
        String texte;

        texte = value.toString();
        compo = new JLabel(texte);
        ((JLabel) compo).setOpaque(true);

        if ((row + column) % 2 == 0)
        {
            compo.setBackground(Color.ORANGE);
        }
        else
        {
            compo.setBackground(Color.GRAY);
        }
    }
    if (column == 2)
    {
        compo = new JComboBox();
        ArrayList<String> lp = PiloteManageur.listePrenom();
        for (String prenom : lp)
        {

```

```

        ((JComboBox) compo).addItem(prenom);
    }
    //Permet de sélectionner le prénom du pilote concerné
    ((JComboBox<String>)compo).setSelectedItem(value);
    table.getColumnModel("PrenomPilote").setCellEditor(new DefaultCellEditor((JComboBox) compo));

}
if ( column == 4 && table.getModel().getValueAt(row, 3)== Boolean.FALSE)
// ici je ne mets un bouton que quand nous pouvons virer le pilote
{

    compo = new JButton("Virer");
}

// si column == 3 le type du modèle de table donne un checkbox par défaut
return compo;
}

```

## Assurer la synchronisation entre les deux colonnes

Dans l'éditeur de bouton, il faut signaler que l'édition de termine pour remettre à jour la table : cela se fait dans l'ActionListener du bouton :

@Override

```

public void actionPerformed(ActionEvent event)
{
    // 1. recuperer le modele de donnees de table
    TableModel tm = table.getModel();
    // modifier le booléen virer à faux sur cette ligne
    tm.setValueAt(Boolean.TRUE, row, 3); // je mets la valeur de la colonne 3 à faux

    stopCellEditing(); // force la remise à jour de la table

}

```

Enfin le modèle de données, dans la méthode setValueAt prévient des changements sur la table. Quand la colonne 3 ( information viré ) est mise à jour, il faut prévenir de modifier les colonnes 3 et 4 de la table. D'où le code :

@Override

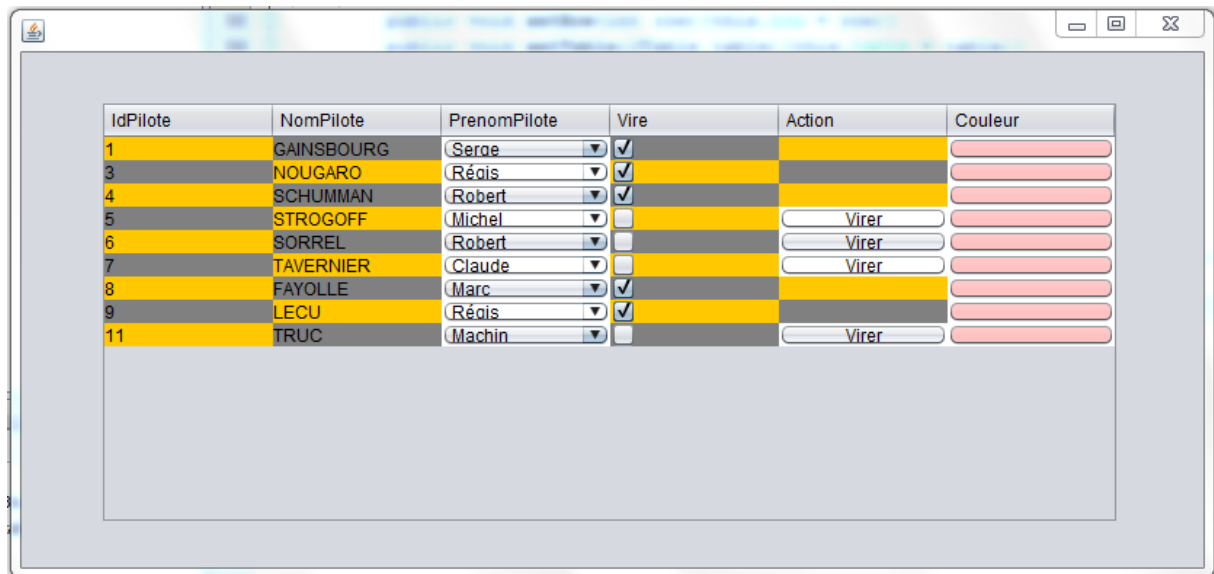
```
public void setValueAt(Object aValue, int rowIndex, int columnIndex)
{
    if (aValue instanceof Boolean && columnIndex == 3)
    {
        //Récupère un pilote dans la JTable
        Pilote pil = listePilote.get(rowIndex);
        pil.setVire((Boolean) aValue);
        //Prévient tout le monde du changement
        fireTableCellUpdated(rowIndex, columnIndex);
        fireTableCellUpdated(rowIndex, 4); // mettre à jour également la colonne des boutons
    }
    else if ( columnIndex == 2)
    {
        Pilote pil = listePilote.get(rowIndex);
        pil.setPrenom((String)aValue);

        //Prévient tout le monde du changement
        fireTableCellUpdated(rowIndex, columnIndex);
    }
    else if (columnIndex == 5)
    {
        Pilote pil = listePilote.get(rowIndex);
        pil.setCouleurCheveux((Color)aValue);

        //Prévient tout le monde du changement
        fireTableCellUpdated(rowIndex, columnIndex);
    }
}
```

Voici le résultat sur la table :





Il n'y a des boutons qu'en face des cases à cocher décochées. D'autre part, les composants des deux colonnes sont synchronisés.

## Annexe un bouton pour le comportement et pour l'apparence

Le code est extrait d'un exemple, je vous le livre dans son jus...

```
public class SexeCellEditor extends AbstractCellEditor implements
TableCellEditor, ActionListener {
    private boolean sexe;
    private JButton bouton;

    public SexeCellEditor() {
        super();

        bouton = new JButton();
        bouton.addActionListener(this);
        bouton.setBorderPainted(false);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        sexe ^= true;

        fireEditingStopped();
    }

    @Override
    public Object getCellEditorValue() {
        return sexe;
    }

    @Override
    public Component getTableCellEditorComponent(JTable table, Object
value, boolean isSelected, int row, int column) {
```

```
        sexe = (Boolean)value;
        return bouton;
    }
}
```

# Une colonne adaptée à son besoin

---

Ici nous allons voir de manière plus globale comment associer un traitement à une cellule de la JTable, ici le choix d'une couleur à l'aide d'un JColorChooser.

**Nous allons procéder en 7 étapes :**

## Rendre la colonne de JButton éditable

Ceci a déjà été fait au chapitre précédent.

## Donner une apparence de JButton coloré aux cellules de la colonne

```
public class ColorCellRenderer extends JButton implements TableCellRenderer {
    @Override
    public Component getTableCellRendererComponent(JTable table, Object value, boolean
isSelected, boolean hasFocus, int row, int column) {

        Color color = (Color) value;

        setText("");
        setBackground(color);

        return this;
    }
}
```

## Nous allons définir le type Color de la colonne

Dans le modèle de données de la table :

```
@Override
public Class<?> getColumnClass(int columnIndex)
{
    Class clas = String.class;
```

```

    if (columnIndex == 3) // colonne de notre JCheckBox
    {
        clas = Boolean.class; // ainsi le JCheckBox est pris en compte pour l'apparence de la colonne
    }
    if ( columnIndex == 5)
    {
        clas = Color.class;
    }
    return clas;
}

```

## Nous modifions le modèle de table pour prendre en compte la couleur

```

public ModelTablePilote()
{
    listePilote = PiloteManageur.listePilotes() ;
    listeColonne = PiloteManageur.columnPilote();
    listeColonne.add("Couleur");
}

@Override
public int getRowCount()
{
    return listePilote.size();
}

@Override
public int getColumnCount()
{
    return listeColonne.size();
}

@Override
public Object getValueAt(int rowIndex, int columnIndex) {
    Pilote pil = listePilote.get(rowIndex);
    switch (columnIndex)

```

```

{
    case 0 : return pil.getNumero();
    case 1 : return pil.getNom();
    case 2 : return pil.getPrenom();
    case 3 : return pil.getVire();
    case 5 : return pil.getCouleurCheveux();
    //Ne doit pas être nul car on ne peut pas ajouter un objet null
    default : return "";
}
}

```

```

@Override
public String getColumnName(int column)
{
    return listeColonne.get(column);
}

```

```

@Override
public void setValueAt(Object aValue, int rowIndex, int columnIndex)
{
    if (aValue instanceof Boolean && columnIndex == 3)
    {
        //Récupère un pilote dans la JTable
        Pilote pil = listePilote.get(rowIndex);
        pil.setVire((Boolean) aValue);
        //Préviens tout le monde du changement
        fireTableCellUpdated(rowIndex, columnIndex);
    }
    else if ( columnIndex == 2)
    {
        Pilote pil = listePilote.get(rowIndex);
        pil.setPrenom((String)aValue);

        //Préviens tout le monde du changement
        fireTableCellUpdated(rowIndex, columnIndex);
    }
}

```

```

else if (columnIndex == 5)
{
    Pilote pil = listePilote.get(rowIndex);
    pil.setCouleurCheveux((Color)aValue);

    //Prévient tout le monde du changement
    fireTableCellUpdated(rowIndex, columnIndex);
}
}

```

## Nous allons associer la couleur et le rendu de la couleur

Dans le concepteur graphique, customize code sur la table : rajouter le lien avec le rendu de la couleur

```
tableAvion.setDefaultRenderer(Color.class, new ColorCellRenderer());
```

## Nous allons définir un comportement à ce bouton couleur

```

public class ColorCellEditor extends AbstractCellEditor implements
TableCellEditor, ActionListener {
    private Color couleur;
    private final JButton bouton;
    private final JColorChooser colorChooser;
    private final JDialog dialog;

    public ColorCellEditor() {
        super();

        bouton = new JButton();
        bouton.setActionCommand("change");
        bouton.addActionListener(this);

        colorChooser = new JColorChooser();
        // ici l'écouteur (this) sera abonné au bouton OK du JColorChooser
        dialog = JColorChooser.createDialog(bouton, "Choisissez votre
couleur", true, colorChooser, this, null);
    }
}

```

```

@Override
public void actionPerformed(ActionEvent e) {
    if ("change".equals(e.getActionCommand())) {
        // ici le bouton de couleur a été actionné
        bouton.setBackground(couleur);
        colorChooser.setColor(couleur);
        dialog.setVisible(true);

        fireEditingStopped();
        // redemande au rendu de redessiner la table
    } else {
        // ici le bouton OK du JColorChooser a été actionné
        couleur = colorChooser.getColor();
    }
}

@Override
public Object getCellEditorValue() {
    return couleur;
}

@Override
public Component getTableCellEditorComponent(JTable table, Object
value, boolean isSelected, int row, int column) {
    couleur = (Color)value;

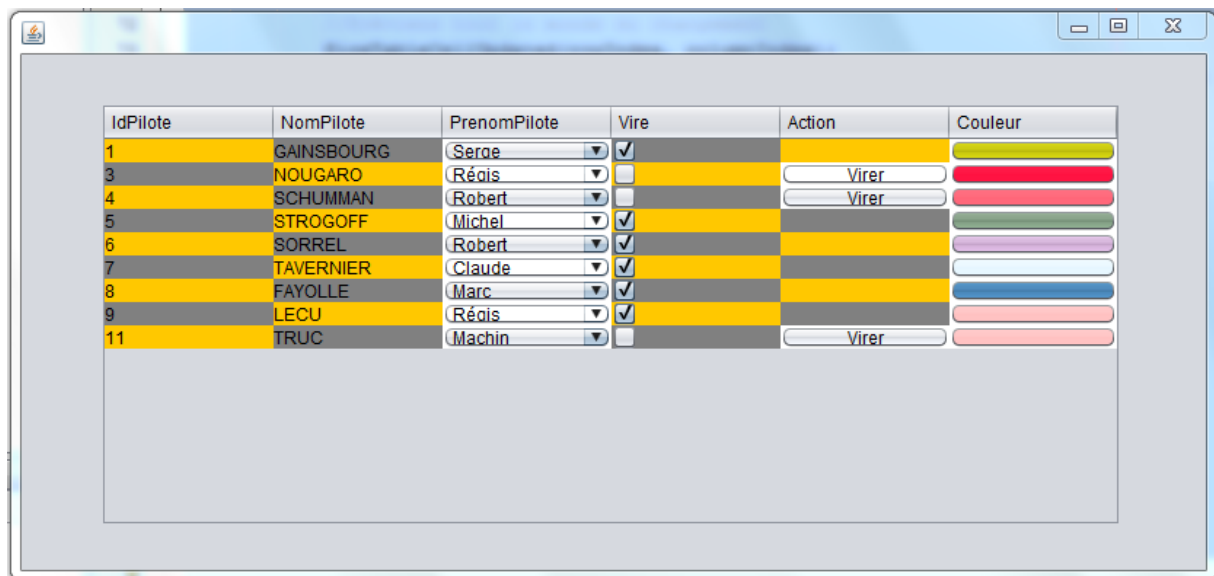
    return bouton;
}
}

```

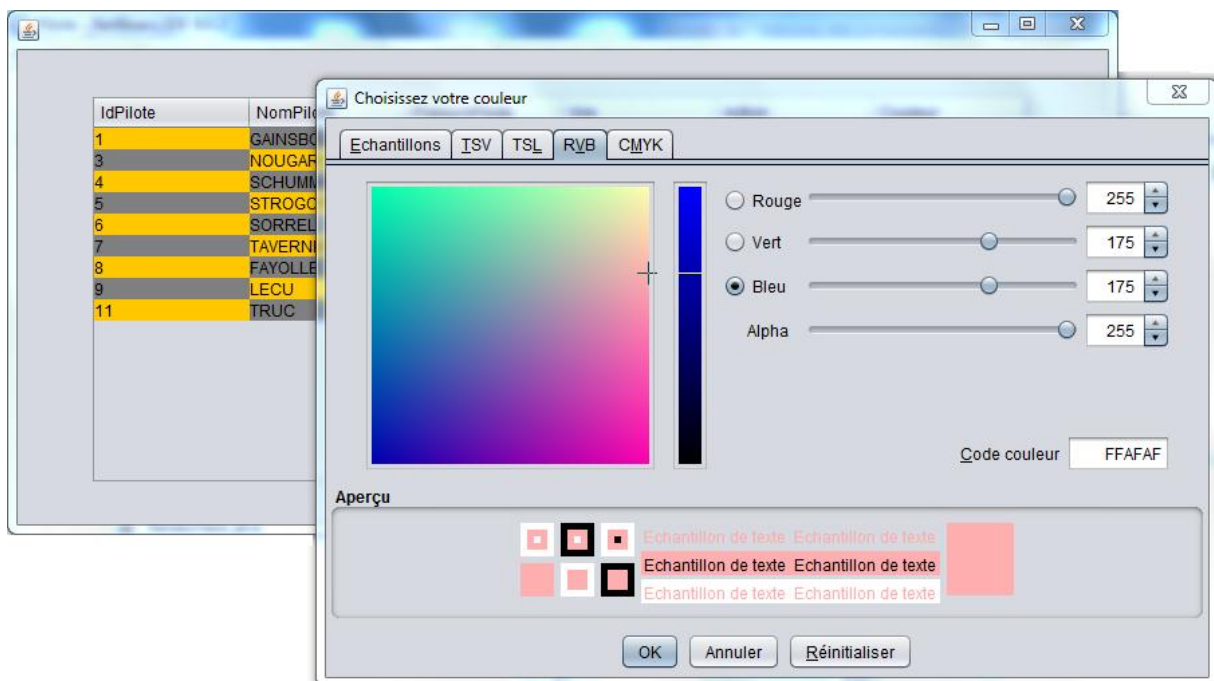
## Nous allons associer ce comportement à la colonne couleur

```
tableAvion.getColumn("Couleur").setCellEditor(new ColorCellEditor());
```

Voici le résultat de l'application :



Et quand on appuie sur un bouton couleur :



Pour la couleur, je n'ai pas pris le temps de la sauvegarder dans la base de données...



# Mettre à jour la base de données

---

## Créer un écouteur sur les changements du modèle de données

Nous allons créer un écouteur sur le modèle de données, qui sera prévenu à chaque changement sur le modèle de données. Il suffit alors d'appeler une méthode du manager pour répercuter cette modification sur la base de données elle-même. Ici les modifications concernent soit le prénom du pilote, soit son état ( viré ou pas ).

```
public class EcouteurModelTable implements TableModelListener
{
    private ModelTablePilote mta;

    public EcouteurModelTable(ModelTablePilote mt) {
        // Nous récupérons le modèle de données, et nous nous abonnons à lui pour être prévenu des
        // changements
        mta = mt;
        mta.addTableModelListener(this);
    }

    @Override
    public void tableChanged(TableModelEvent e)
    {
        if ( e.getColumn() == 3) // changement du booléen viré
        {
            // récupération des informations id du pilote et état de viré.
            Boolean b =(Boolean) mta.getValueAt(e.getFirstRow(), e.getColumn());
            int id = (Integer) mta.getValueAt(e.getFirstRow(), 0);

            // appel du manager pour graver l'information sur la base de données
            PiloteManager.virerPilote(id, b);
        }
        else if ( e.getColumn() == 2)// changement de prénom
        {
```

```

        // récupération des informations id du pilote et son nouveau prénom
        String prenom = (String)mta.getValueAt(e.getFirstRow(), e.getColumn());
        int id = (Integer) mta.getValueAt(e.getFirstRow(), 0);
        // appel du manageur pour graver l'information sur la base de données
        if (PiloteManageur.changerPrenomPilote(id, prenom)== false)
            JOptionPane.showMessageDialog(null, "opération impossible");
    }
}
}

```

## Créer une instance de l'écouteur ( l'abonnement se faisant automatiquement )

Cliquer droit sur la table, et lancer customize code. Ajouter la ligne suivante pour que les modifications du modèle soient répercutées sur la base de données.

```
EcouteurModelTable emt = new EcouteurModelTable((ModelTablePilote) tableAvion.getModel());
```

Voila, notre table est maintenant opérationnelle.