

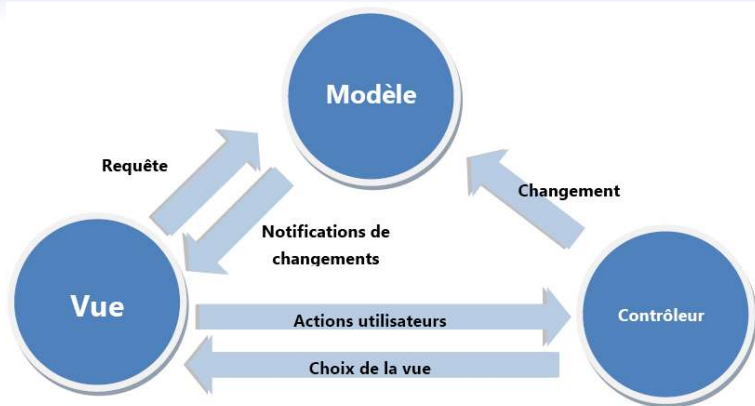


# Java et le pattern MVC

# Plan

- 1 Le pattern MVC
- 2 Le pattern MVC dans le JDK

# Le pattern MVC : Modèle Vue Contrôleur



## Intérêt

- Séparation entre **données**, **présentation** et **traitements**.

# Le pattern MVC : Modèle Vue Contrôleur

## Modèle

- représente le comportement de l'application : traitements des données, interactions avec une BD, etc.
- Assure la **gestion des données** et garantit leur **intégrité**. etc.

## Vue

- l'interface avec laquelle l'utilisateur interagit.
- présente l'**état du modèle** et gère les **requêtes utilisateurs**.

## Contrôleur

- gère les événements : **met à jour** la vue et/ou le modèle.
- n'effectue **aucun traitement**, ni ne **modifie aucune donnée**.

## Un exemple de Vue : *javax.swing.JList*

### javax.swing.JList

javax.swing

## Class JList

[java.lang.Object](#)

└ [java.awt.Component](#)

└ [java.awt.Container](#)

└ [javax.swing.JComponent](#)

└ [javax.swing.JList](#)

### All Implemented Interfaces:

[ImageObserver](#), [MenuContainer](#), [Serializable](#), [Accessible](#), [Scrollable](#)

---

```
public class JList
extends JComponent
implements Scrollable, Accessible
```

A component that displays a list of objects and allows the user to select one or more items. A separate model, `ListModel`, maintains the contents of the list.

## Exemple de Modèle :

# *javax.swing.DefaultListModel*

### javax.swing.DefaultListModel

javax.swing

## Class DefaultListModel

[java.lang.Object](#)

└ [javax.swing.AbstractListModel](#)

└ [javax.swing.DefaultListModel](#)

**All Implemented Interfaces:**

[Serializable](#), [ListModel](#)

---

```
public class DefaultListModel
extends AbstractListModel
```

This class loosely implements the `java.util.Vector` API, in that it implements the 1.1.x version of `java.util.Vector`, has no collection class support, and notifies the `ListDataListeners` when changes occur. Presently it delegates to a `Vector`, in a future release it will be a real `Collection` implementation.

## Exemple de Modèle :

# *javax.swing.AbstractListModel*

### javax.swing.AbstractListModel

javax.swing

## Class AbstractListModel

[java.lang.Object](#)

└ javax.swing.AbstractListModel

### All Implemented Interfaces:

[Serializable](#), [ListModel](#)

### Direct Known Subclasses:

[BasicDirectoryModel](#), [DefaultComboBoxModel](#), [DefaultListModel](#),  
[MetalFileChooserUI.DirectoryComboBoxModel](#),  
[MetalFileChooserUI.FilterComboBoxModel](#)

---

```
public abstract class AbstractListModel
extends Object
implements ListModel, Serializable
```

The abstract definition for the data model that provides a List with its contents.

## Exemple de Modèle : *javax.swing.ListModel*

### javax.swing.ListModel

javax.swing

## Interface ListModel

### All Known Subinterfaces:

[ComboBoxModel](#), [MutableComboBoxModel](#)

### All Known Implementing Classes:

[AbstractListModel](#), [BasicDirectoryModel](#), [DefaultComboBoxModel](#),  
[DefaultListModel](#), [MetalFileChooserUI.DirectoryComboBoxModel](#),  
[MetalFileChooserUI.FilterComboBoxModel](#)

---

```
public interface ListModel
```

This interface defines the methods components like JList use to get the value of each cell in a list and the length of the list. Logically the model is a vector, indices vary from 0 to `ListDataModel.getSize() - 1`. Any change to the contents or length of the data model must be reported to all of the `ListDataListeners`.



## Exemple de Contrôleur :

# *javax.swing.event.ListSelectionListener*

### javax.swing.event.ListSelectionListener

javax.swing.event

## Interface ListSelectionListener

### All Superinterfaces:

[EventListener](#)

### All Known Implementing Classes:

[BasicComboPopup.ListSelectionHandler](#), [BasicFileChooserUI.SelectionListener](#),  
[BasicListUI.ListSelectionHandler](#), [DefaultTableColumnModel](#), [JList.AccessibleJList](#), [JTable](#),  
[JTable.AccessibleJTable](#)

---

```
public interface ListSelectionListener
extends EventListener
```

The listener that's notified when a lists selection value changes.

# Exemple de Contrôleur :

## *javax.swing.event.ListDataListener*

### javax.swing.event.ListDataListener

javax.swing.event

#### Interface `ListDataListener`

##### All Superinterfaces:

[EventListener](#)

##### All Known Implementing Classes:

[BasicComboBoxUI.ListDataHandler](#), [BasicComboPopup.ListDataHandler](#), [BasicListUI.ListDataHandler](#), [JComboBox](#), [JList.AccessibleJList](#)

---

```
public interface ListDataListener
    extends EventListener
```

ListDataListener

---

#### Method Summary

void	<a href="#">contentsChanged</a> ( <a href="#">ListDataEvent</a> e)	Sent when the contents of the list has changed in a way that's too complex to characterize with the previous methods.
void	<a href="#">intervalAdded</a> ( <a href="#">ListDataEvent</a> e)	Sent after the indices in the index0,index1 interval have been inserted in the data model.
void	<a href="#">intervalRemoved</a> ( <a href="#">ListDataEvent</a> e)	Sent after the indices in the index0,index1 interval have been removed from the data model.

# Exemple de code

## Un *ListSelectionListener*

```
class SharedListSelectionHandler implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent e) {
        ListSelectionModel lsm = (ListSelectionModel)e.getSource();
        int firstIndex = e.getFirstIndex();
        int lastIndex = e.getLastIndex();
        boolean isAdjusting = e.getValueIsAdjusting();
        output.append("Event for indexes "
            + firstIndex + " - " + lastIndex
            + "; isAdjusting is " + isAdjusting
            + "; selected indexes:");

        if (lsm.isEmpty()) {
            output.append(" <none>");
        } else {
            // Find out which indexes are selected.
            int minIndex = lsm.getMinSelectionIndex();
            int maxIndex = lsm.getMaxSelectionIndex();
            for (int i = minIndex; i <= maxIndex; i++) {
                if (lsm.isSelectedIndex(i)) {
                    output.append(" " + i);
                }
            }
        }
        output.append(newline);
    }
}
```