



# Unit 5

## Swing Components

Software Analysis and Design Project

**Universidad Autónoma de Madrid**

# Password: JPasswordField

Text field in which letters typed by the user are shown as \*.

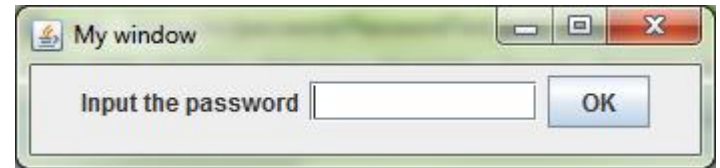
```
// create window
JFrame window = new JFrame("My window");

// get container, set layout
Container container = window.getContentPane();
container.setLayout(new FlowLayout());

// create components
JLabel label = new JLabel("Input the password");
final JPasswordField pass = new JPasswordField(10);
JButton okButton = new JButton("OK");

// associate actions to components using anonymous class
okButton.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null, String.valueOf(pass.getPassword())); }
    });

// add components to container
container.add(label);
container.add(pass);
container.add(okButton);
```



# Password: JPasswordField

Text field in which letters typed by the user are shown as \*.

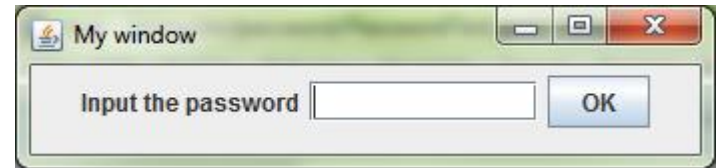
```
// create window
JFrame window = new JFrame("My window");

// get container, set layout
Container container = window.getContentPane();
container.setLayout(new FlowLayout());

// create components
JLabel label = new JLabel("Input the password");
final JPasswordField pass = new JPasswordField(10);
JButton okButton = new JButton("OK");

// associate actions to components using lambda expression
okButton.addActionListener(
    e -> JOptionPane.showMessageDialog(null, String.copyValueOf(pass.getPassword()))
);

// add components to container
container.add(label);
container.add(pass);
container.add(okButton);
```



# Check Box: JCheckBox

Each box can be selected or unselected.

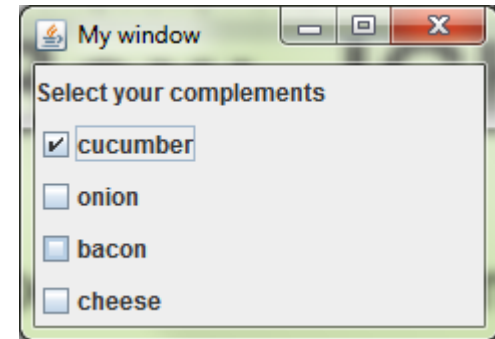
```
// create window
JFrame window = new JFrame("My window");

// Create options
JCheckBox box1 = new JCheckBox("cucumber");
JCheckBox box2 = new JCheckBox("onion");
JCheckBox box3 = new JCheckBox("bacon");
JCheckBox box4 = new JCheckBox("cheese");

// Use the isSelected method to check if a box has been selected
if (box1.isSelected())
    System.out.println("Box #1 is selected");

// Use the setSelected method to change the value of a box
box1.setSelected(true);

// Add boxes to the panel where they are going to be shown
JPanel checkboxExample = new JPanel(new GridLayout(5,1));
checkboxExample.add(new JLabel("Select your complements"));
checkboxExample.add(box1);
checkboxExample.add(box2);
checkboxExample.add(box3);
checkboxExample.add(box4);
```



# Radio buttons: JRadioButton

Only one of the options can be selected within a group

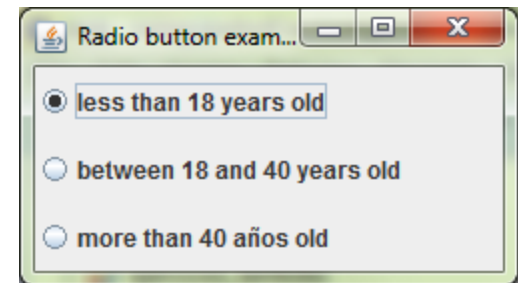
```
// Create options
JRadioButton option1 = new JRadioButton("less than 18 years old");
JRadioButton option2 = new JRadioButton("between 18 and 40 years old");
JRadioButton option3 = new JRadioButton("more than 40 años old");
option1.setSelected(true);

// Create a group for options. It will guarantee
// that only one of the options is selected
ButtonGroup group = new ButtonGroup();

// Add the options to the group
group.add(option1);
group.add(option2);
group.add(option3);

// Add the options to the panel where they are going to be shown
JPanel radioButtonExample = new JPanel(new GridLayout(3, 1));
radioButtonExample.add(option1);
radioButtonExample.add(option2);
radioButtonExample.add(option3);

// Use the isSelected method to check if an option is selected
if (option1.isSelected())
    System.out.println("You have selected the less than 18 years old option");
```



# Radio buttons: JRadioButton

## *Reaction to selection events*

```
public class RadioButtonExample extends JFrame implements ActionListener{
    private JRadioButton options[] = new JRadioButton[3];

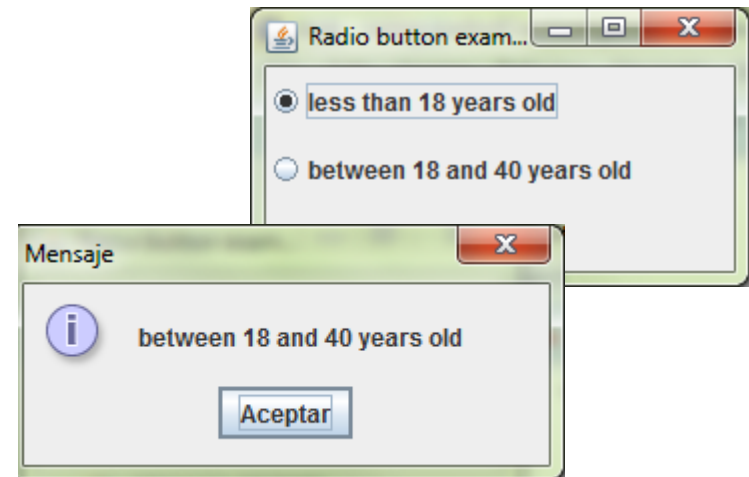
    public RadioButtonExample() {
        super("Radio button example");
        options[0] = new JRadioButton("less than 18 years old");
        options[1] = new JRadioButton("between 18 and 40 years old");
        options[2] = new JRadioButton("more than 40 años old");
        options[0].setSelected(true);

        ButtonGroup group = new ButtonGroup();
        JPanel radioButtonExample = new JPanel(new GridLayout(3, 1));

        for (JRadioButton j : this.options) {
            j.addActionListener(this);
            group.add(j); // Add the options to the group
            radioButtonExample.add(j);
        }
        this.setContentPane(radioButtonExample);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(250,140);
        this.setVisible(true);
    }

    @Override public void actionPerformed(ActionEvent ev) {
        for (JRadioButton j : this.options)
            if (j.isSelected()) JOptionPane.showMessageDialog(this, j.getText());
    }

    public static void main(String[] args) { new RadioButtonExample(); }
}
```



# Combo boxes: JComboBox

They allow to select an option from a deployable list

```
String[] days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};
```

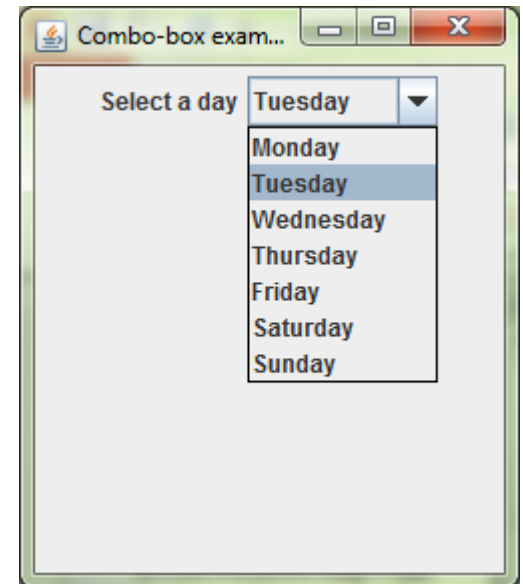
```
// Create the combo box, using the array as a parameter  
JComboBox<String> combobox = new JComboBox<String>(days);
```

```
// Use the setSelectedIndex(<posicion>) method to select an option  
combobox.setSelectedIndex(2);
```

```
// Add the combo box to the panel where it is going to be shown  
JPanel comboBoxExample = new JPanel(new FlowLayout());  
comboBoxExample.add(new JLabel("Select a day"));  
comboBoxExample.add(combobox);
```

```
// Use the getSelectedItem or getSelectedIndex methods  
// to check which option is selected  
String selectedValue = (String)combobox.getSelectedItem();  
int selectedIndex = combobox.getSelectedIndex();
```

```
this.setContentPane(comboBoxExample);  
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
this.setSize(250,140);  
this.setVisible(true);
```



# Combo boxes: JComboBox

## *Reaction to selection events*

```
// Create an array with the combo box options
String[] days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};

// Create the combo box, using the array as a parameter
JComboBox<String> combobox = new JComboBox<String>(days);

// Use the setSelectedIndex(<posicion>) method to select an option
combobox.setSelectedIndex(2);

combobox.addItemListener( itemEvent -> {
    int state = itemEvent.getStateChange();
    System.out.println((state == ItemEvent.SELECTED) ? "Selected" : "Deselected");
    System.out.println("Item: " + itemEvent.getItem());
    ItemSelectable is = itemEvent.getItemSelectable();
    System.out.println(", Selected: " + is.getSelectedObjects()[0]);
});
```

```
Deselected
Item: Wednesday
, Selected: Monday
Selected
Item: Monday
, Selected: Monday
```



# Lists: JList (immutable list)

In this example the elements of the list are given when it is created and afterwards it is not possible to add or delete elements from it

```
// Create an array with the elements of the list
String[] persons = {"anna", "ed", "esther", "joseph", "john", "louis", "maria", "michael", "zoe"};

// Create the list, using the array as parameter
JList<String> list = new JList<String>(persons);

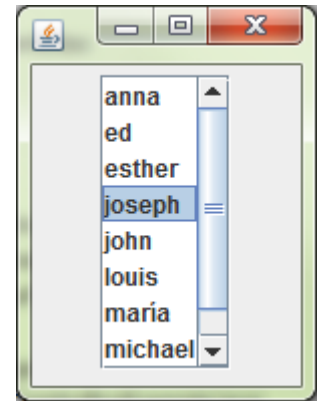
// By default it is possible to select several items in simultaneously.
// If you want just one to be selected, use the setSelectionMode method
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

// Consider the possibility of creating a scrolling bar for the list
// just in case the number of elements is bigger than expected
JScrollPane scrollPane = new JScrollPane(list);

// Add the scroll bar to the panel where it is going to be shown
JPanel listExample = new JPanel();
listExample.add(scrollPane);

// Use a listener to execute an action when an item is selected/unselected
list.addListSelectionListener(new ListSelectionListener () {
    public void valueChanged(ListSelectionEvent arg0) {
        if (arg0.getValueIsAdjusting()==false) {
            // Get a reference to the changed list
            JList<String> list = (JList<String>)arg0.getSource();
            // Use the getSelectedValue or getSelectedIndex methods to check which item is selected
            String selectedValue = (String)list.getSelectedValue();
            int selectedIndex = list.getSelectedIndex();

            System.out.println("Selected "+selectedValue+" index = "+selectedIndex);
        }
    }
});
```



# Lists: JList (dynamic list)

A DefaultListModel that contains the data has to be used in order to be able to add and delete elements from a list dynamically once it is created

```
// Create a DefaultListModel with the elements of the list
DefaultListModel<String> personsM = new DefaultListModel<String>();
personsM.addElement("anna");
personsM.addElement("ed");
personsM.addElement("esther");
personsM.addElement("joseph");
personsM.addElement("john");
personsM.addElement("louis");
personsM.addElement("maría");
personsM.addElement("michael");
personsM.addElement("zoe");

// Create the list, using the model as parameter
JList<String> list = new JList<String>(personsM);

// Use the remove(<index>) or remove(<object>) methods
// of the model to remove an element from the list
personsM.remove(0);           // ana is removed
personsM.removeElement("esther"); // esther is removed

// Use the addElement method of the model to add an item at the end of the list
personsM.addElement("winston");

// An item can be inserted at some position by means of the add(<index>,<object>) of the model
personsM.add(0, "susan");

// Use the set(<index>,<object>) method of the model to modify an existing item
personsM.set(1, "peter"); // ed is substituted by peter
```



# Tables: JTable (immutable table)

In this example the rows of the table are given when it is created, and afterwards it is not possible to add or delete rows. The cells of the table are editable, and they show the object assigned to them using the *toString* method.

```
// Create an array with the title of the columns
String[] titles = {"name", "surname", "mark", "pass"};

// Create a matrix with the rows of the table
Object[][] rows = {
    {"Alba", "Sanz", new Double(5), new Boolean(true)},
    {"Belen", "Lopez", new Double(3.2), new Boolean(false)},
    {"Luisa", "Lopez", new Double(4.5), new Boolean(false)},
    {"Marcos", "Perez", new Double(8.5), new Boolean(true)},
    {"Miguel", "Vela", new Double(7), new Boolean(true)},
    {"Sara", "Valle", new Double(10), new Boolean(true)},
};
this.rows = rows;
```

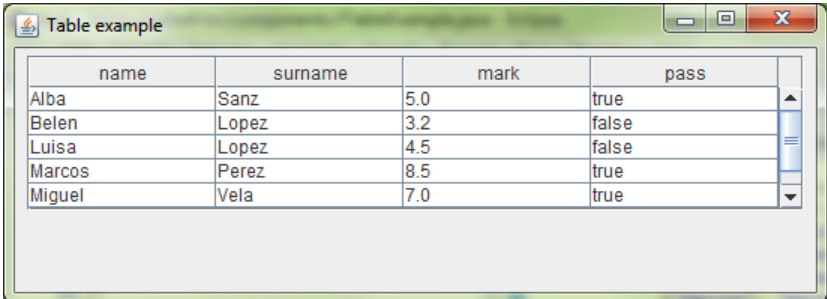
```
// Create the table, using the rows and titles as parameters
JTable table = new JTable(rows, titles);
```

```
// By default several rows of the table can be selected simultaneously.
// If you want just one row to be selected, use the setSelectionMode method
table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
// The size of the table can be fixed
table.setPreferredScrollableViewportSize(new Dimension(500, 80));
```

```
// Consider the possibility to create a scroll bar for the table just in case the number of rows is too big
JScrollPane scrollBar = new JScrollPane(table);
```

```
// Add the scroll bar to the panel where it will be shown
JPanel tableExample = new JPanel();
tableExample.add(scrollBar);
```



name	surname	mark	pass
Alba	Sanz	5.0	true
Belen	Lopez	3.2	false
Luisa	Lopez	4.5	false
Marcos	Perez	8.5	true
Miguel	Vela	7.0	true
Sara	Valle	10.0	true

# Tables: JTable (dynamic table)

A **DefaultTableModel** that contains the data has to be used in order to be able to add and delete rows from a table dynamically once it is created.

```
// Create an array of column titles
Object[] titles = {"name", "surnames", "mark", "passed"};
```

```
// Create a matrix of table contents
Object[][] contents = {
    {"Alba", "Sanz", new Double(5), new Boolean(true)},
    {"Belén", "López", new Double(3.2), new Boolean(false)},
    {"Luisa", "López", new Double(4.5), new Boolean(false)},
    {"Marcos", "Pérez", new Double(8.5), new Boolean(true)},
    {"Miguel", "Vela", new Double(7), new Boolean(true)},
    {"Sara", "Valle", new Double(10), new Boolean(true)},
};
```

```
// Create a DefaultTableModel from the previous data
DefaultTableModel dataModel = new DefaultTableModel(contents, titles);
```

```
// Create the table, using the model as parameter
JTable table = new JTable(dataModel);
```

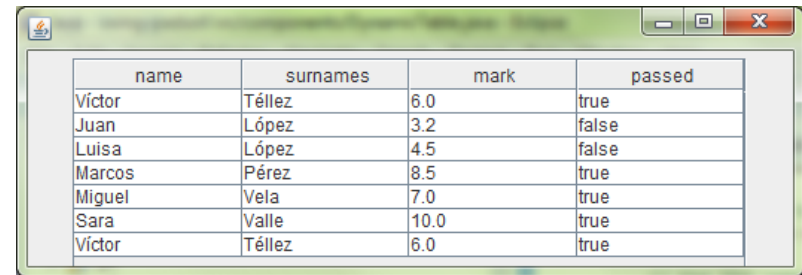
```
// Use the removeRow(<index>) method of the model to delete rows from the table
dataModel.removeRow(0); // borra Alba
```

```
// Use the addRow method of the model to add rows at the end of the table
Object[] newRow = {"Víctor", "Téllez", new Double(6), new Boolean(true)};
dataModel.addRow(newRow);
```

```
// The insertRow(<index>,<object[]>) method of the model can also be used to insert a row in a specific position
dataModel.insertRow(0, newRow);
```

```
// Use the set(<object>,<row>,<column>) method of the model to modify the contents of the table
dataModel.setValueAt("Juan", 1, 0); // Juan is substituted for Belén
```

```
// Consider the possibility to create a scroll bar for the table just in case the number of rows is too big
```

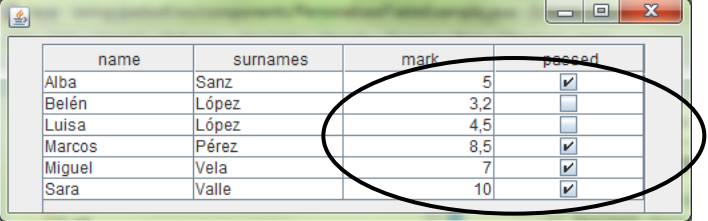


name	surnames	mark	passed
Víctor	Téllez	6.0	true
Juan	López	3.2	false
Luisa	López	4.5	false
Marcos	Pérez	8.5	true
Miguel	Vela	7.0	true
Sara	Valle	10.0	true
Víctor	Téllez	6.0	true

# Tables: JTable (personalized table)

A personalized table model (a class that extends **AbstractTableModel**) can be used in order to have control of how data are shown in a table.

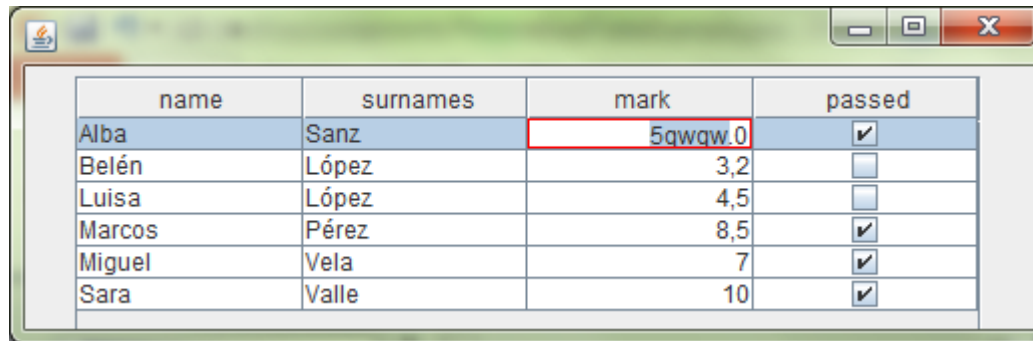
```
class MyPersonalizedModel extends AbstractTableModel {  
  
    // In this example table data are attributes of the class  
    private Object[] titles;  
    private Object[][] contents;  
  
    // In this example the constructor initializes the attributes  
    public MyPersonalizedModel () { ... }  
  
    // The class must implement the getColumnCount, getRowCount and getValueAt methods  
    public int getColumnCount() { return this.titles.length; } // Method that returns the number of columns  
    public int getRowCount() { return this.contents.length; } // Method that returns the number of rows  
    public Object getValueAt(int row, int col) { return this.contents[row][col]; } // Method that returns the contents of a cell  
  
    // By default the contents of the cells can not be edited. Overwrite the isCellEditable method to make them editable  
    public boolean isCellEditable (int row, int col) { return row==0; } // allows edition of the first row  
  
    // Overwrite the setValueAt method to specify how to change the value of cells  
    public void setValueAt (Object value, int row, int col) {  
        this.contents[row][col] = value;  
        fireTableCellUpdated(row, col);  
    }  
  
    // The getColumnName method can be overwritten to return the title of a column  
    public String getColumnName (int col) { return (String)titles[col]; }  
  
    // Cell data are rendered using the predefined format for the class returned by the getColumnClass method.  
    // Boolean data are shown by check boxes and numbers are aligned to the right (see figure).  
    public Class<?> getColumnClass (int col) { return getValueAt(0, col).getClass(); }  
}
```



name	surnames	mark	passed
Alba	Sanz	5	<input checked="" type="checkbox"/>
Belén	López	3,2	<input type="checkbox"/>
Luisa	López	4,5	<input type="checkbox"/>
Marcos	Pérez	8,5	<input checked="" type="checkbox"/>
Miguel	Vela	7	<input checked="" type="checkbox"/>
Sara	Valle	10	<input checked="" type="checkbox"/>

# Tables: JTable (personalized table)

Some error control is obtained:



name	surnames	mark	passed
Alba	Sanz	5qwgw.0	<input checked="" type="checkbox"/>
Belén	López	3,2	<input type="checkbox"/>
Luisa	López	4,5	<input type="checkbox"/>
Marcos	Pérez	8,5	<input checked="" type="checkbox"/>
Miguel	Vela	7	<input checked="" type="checkbox"/>
Sara	Valle	10	<input checked="" type="checkbox"/>

# Tables: JTable (personalized table)

Once the personalized table model is created it can be instantiated as follows:

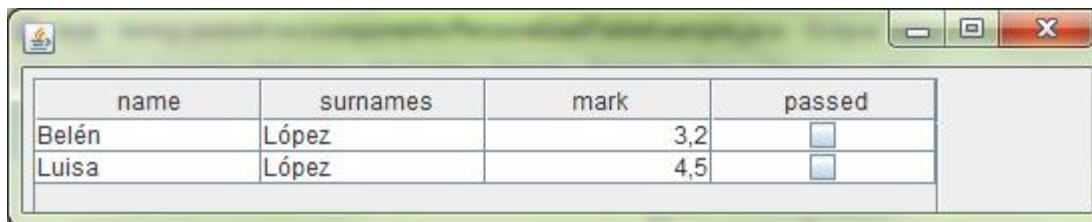
```
// Create a data model
AbstractTableModel dataMmodel = new MyPersonalizedModel();

// Create the table, using the model as an argument
JTable table = new JTable(dataModel);
```

## Other utilities of JTable: reordering and filtering

```
// The setAutoCreateRowSorter method is used to activate rows reordering when the head of a column is clicked
table.setAutoCreateRowSorter(true);
```

```
// The class TableRowSorter can be used to filter table rows. The following example
// filters the table so that only rows that contain the word "López" in some cell are shown
TableRowSorter<MyPersonalizedModel> filter = new TableRowSorter<>(dataModel);
RowFilter<MyPersonalizedModel, Integer> rf = RowFilter.regexFilter("López");
filter.setRowFilter(rf);
table.setRowSorter(filter);
```



name	surnames	mark	passed
Belén	López	3,2	<input type="checkbox"/>
Luisa	López	4,5	<input type="checkbox"/>

# Trees: JTree

A JTree provides a hierarchical view of a set of data

```
// Create the root node, passing the text to be shown
DefaultMutableTreeNode root = new DefaultMutableTreeNode("Apuntes de PADS");

// Create the tree, using its root node as argument
final JTree tree = new JTree(root);
tree.setPreferredSize(new Dimension(200,150));

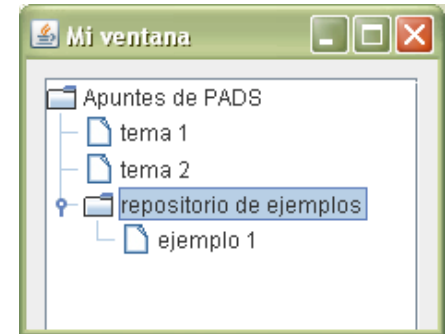
// By default the tree allows multiple node selection. Use the setSelectionModel method to activate single selection mode
tree.getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);

// Use the add(<node>) method to add sons to any node of the tree
root.add(new DefaultMutableTreeNode("subject 1"));
root.add(new DefaultMutableTreeNode("subject 2"));
DefaultMutableTreeNode folder = new DefaultMutableTreeNode("Examples folder");
folder.add(new DefaultMutableTreeNode("example 1"));
root.add(folder);

// Use a TreeSelectionListener to specify actions to be performed when a tree node is selected
tree.addTreeSelectionListener(new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent e) {
        // To get the node that has been selected:
        // - if the selection mode is SINGLE_TREE_SELECTION, use the getLastSelectedPathComponent method
        Object selectedNode = tree.getLastSelectedPathComponent();
        // - if multiple selection is activated, use the getSelectionPaths / getSelectionRows methods
        int[] selectedNodesIndex = tree.getSelectionRows();
        TreePath[] selectedNodesPath = tree.getSelectionPaths();
    }
});

// Consider the creation of a scroll bar for the tree just in case its size gets larger than expected
JScrollPane scrollbar = new JScrollPane(tree);

// Add the scroll bar
JPanel treeExample = new JPanel();
treeExample.add(scrollbar);
```





# Trees: JTree (dynamic tree)

A **DefaultTreeModel** that contains the data has to be used in order to be able to add and delete nodes from the tree dynamically.

```
// Create the root node of the tree, using the corresponding text as argument
DefaultMutableTreeNode root = new DefaultMutableTreeNode("PADS notes");

// Create the data model of the tree, using its root as argument
DefaultTreeModel dataModel = new DefaultTreeModel(root);

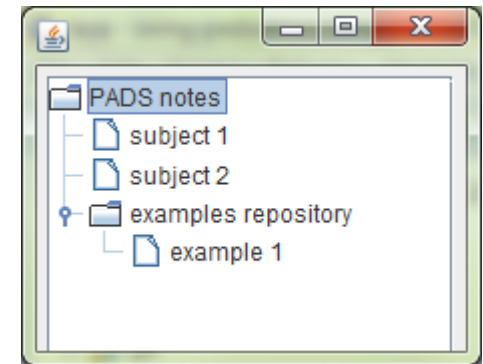
// Create the tree, using the data model as argument
JTree tree = new JTree (dataModel);

// The size of the tree can be fixed
tree.setPreferredSize(new Dimension(200, 40));

// Use the insertNodeInto method of the model to add sons to a node. Its arguments are the node to be inserted,
// the parent node where it will be inserted and the position of the node among the sons
dataModel.insertNodeInto(new DefaultMutableTreeNode("subject 1"), root, 0); // "subject 1" is the first son of root
dataModel.insertNodeInto(new DefaultMutableTreeNode("subject 2"), root, 1); // "subject 2" is the second son of root
DefaultMutableTreeNode repository = new DefaultMutableTreeNode("examples repository");
dataModel.insertNodeInto(repository, root, 2); // "examples ..." is the third son of root
dataModel.insertNodeInto(new DefaultMutableTreeNode("example 1"), repository, 0); // "example 1" is the first
// son of "example..."

// Use the removeFromParent(<node>) method to delete a node. Its sons will be deleted also
//dataModel.removeNodeFromParent(repository);

// In order to modify the value of the selected node:
TreePath path = tree.getSelectionPath(); // get the path to the selected node
dataModel.valueForPathChanged(path, "subject 20"); // assign new value
```



# Tabs: JTabbedPane

Tabs allow several layers of information to be shown or hidden alternatively. A JTabbedPane is a container, like JPanel.

```
// Create a panel for each tab
JPanel tab1 = new JPanel();
tab1.setLayout(new FlowLayout());
tab1.add(new JLabel("Name"));
tab1.add(new JTextField(10));
tab1.add(new JButton("OK"));
tab1.setPreferredSize(new Dimension(300, 100));

// Two other panels
JPanel tab2 = new JPanel();
tab2.add(new JLabel("Tab 2"));
JPanel tab3 = new JPanel();
tab3.add(new JLabel("Tab 3"));

// Create a container of type JTabbedPane (not a JPanel).
JTabbedPane tabs = new JTabbedPane();

// Add panels to container using the method addTab(<title>,<panel>)
tabs.addTab("Tab 1", tab1);
tabs.addTab("Tab 2", tab2);
// The addTab method can be used also to associate an icon and contextual help to a tab
tabs.addTab("Tab 3", new ImageIcon("Image3.gif"), tab3, "This tab can be used for something");

// A tab from the container can be selected by means of setSelectedIndex(<index>)
tabs.setSelectedIndex(2);

// A tab can be deleted from the container by means of removeTabAt(<index>)
tabs.removeTabAt(1);

// In order to specify the execution of some actions when switching between tabs we can define a ChangeListener
tabs.addChangeListener( e -> System.out.println("Changed"));
```

