Laboratory Manual 3
Using Swing Components in NetBeans IDE

1.1 JComponent

With the exception of top-level containers, all Swing components whose names begin with "J" descend from the `JComponent` class. For example, `JPanel`, `JScrollPane`, `JButton`, and `JTable` all inherit from `JComponent`. However, `JFrame` and `JDialog` don't because they implement top-level containers.

The `JComponent` class extends the `Container` class, which itself extends `Component`. The `Component` class includes everything from providing layout hints to supporting painting and events. The `Container` class has support for adding components to the container and laying them out.

JComponent Features

The JComponent class provides the following functionality to its descendants:

- Tool tips
  By specifying a string with the *setToolTipText* method, you can provide help to users of a component. When the cursor pauses over the component, the specified string is displayed in a small window that appears near the component.
- Painting and borders
  The *setBorder* method allows you to specify the border that a component displays around its edges.
- Application-wide pluggable look and feel
  Behind the scenes, each *JComponent* object has a corresponding *ComponentUI* object that performs all the drawing, event handling, size determination, and so on for that *JComponent*. Exactly which *ComponentUI* object is used depends on the current look and feel, which you can set using the *UIManager*.*setLookAndFeel* method.
- Custom properties
  You can associate one or more properties (name/object pairs) with any *JComponent*. For example, a layout manager might use properties to associate a constraints object with each *JComponent* it manages. You put and get properties using the *putClientProperty* and *getClientProperty* methods.
- Support for layout
  Although the Component class provides layout hint methods such as getPreferredSize and getAlignmentX, it doesn't provide any way to set these layout hints, short of creating a subclass and overriding the methods. To give you another way to set layout hints, the JComponent class adds setter methods — setMinimumSize, setMaximumSize, setAlignmentX, and setAlignmentY.
- Support for accessibility
  The JComponent class provides API and basic functionality to help assistive technologies such as screen readers get information from Swing components
- Support for drag and drop
  The JComponent class provides API to set a component's transfer handler, which is the basis for Swing's drag and drop support

- Double buffering
  Double buffering smooths on-screen painting.
- Key bindings
  This feature makes components react when the user presses a key on the keyboard. For example, in many look and feels when a button has the focus, typing the Space key is equivalent to a mouse click on the button. The look and feel automatically sets up the bindings between pressing and releasing the Space key and the resulting effects on the button.

1.2 The JComponent API

The JComponent class provides many new methods and inherits many methods from Component and Container. The following tables summarize the methods we use the most.

Customizing Component Appearance

Method	Purpose
void setBorder(Border) Border getBorder()	Set or get the border of the component. See How to Use Borders for details.
void setForeground(Color) void setBackground(Color)	Set the foreground or background color for the component. The foreground is generally the color used to draw the text in a component. The background is (not surprisingly) the color of the background areas of the component, assuming that the component is opaque.
Color getForeground() Color getBackground()	Get the foreground or background color for the component.
void setOpaque(boolean) boolean isOpaque()	Set or get whether the component is opaque. An opaque component fills its background with its background color.
void setFont(Font) Font getFont()	Set or get the component's font. If a font has not been set for the component, the font of its parent is returned.
void setCursor(Cursor) Cursor getCursor()	Set or get the cursor displayed over the component and all components it contains (except for children that have their own cursor set). Example: `aPanel.setCursor( Cursor.getPredefinedCursor( Cursor.WAIT_CURSOR));`

Setting and Getting Component State

Method	Purpose

void setComponentPopupMenu(JPopupMenu)	Sets the `JPopupMenu` for this `JComponent`. The UI is responsible for registering bindings and adding the necessary listeners such that the `JPopupMenu` will be shown at the appropriate time. When the `JPopupMenu` is shown depends upon the look and feel: some may show it on a mouse event, some may enable a key binding. If `popup` is null, and `getInheritsPopupMenu` returns `true`, then `getComponentPopupMenu` will be delegated to the parent. This provides for a way to make all child components inherit the `popupmenu` of the parent.
void setTransferHandler(TransferHandler) TransferHandler getTransferHandler()	Set or remove the `transferHandler` property. The `TransferHandler` supports exchanging data via cut, copy, or paste to/from a clipboard as well a drag and drop. See Introduction to DnD for more details.
void setToolTipText(String)	Set the text to display in a tool tip. See How to Use Tool Tips for more information.
void setName(String) String getName()	Set or get the name of the component. This can be useful when you need to associate text with a component that does not display text.
boolean isShowing()	Determine whether the component is showing on screen. This means that the component must be visible, and it must be in a container that is visible and showing.
void setEnabled(boolean) boolean isEnabled()	Set or get whether the component is enabled. An enabled component can respond to user input and generate events.
void setVisible(boolean) boolean isVisible()	Set or get whether the component is visible. Components are initially visible, with the exception of top-level components.

Handling Events
(see Writing Event Listeners for details)

Method	Purpose
void addHierarchyListener(hierarchyListener l) void removeHierarchyListener(hierarchyListener l)	Adds or removes the specified hierarchy listener to receive hierarchy changed events from this component when the hierarchy to which this container belongs changes. If listener l is null, no exception is thrown and no action is performed.
void addMouseListener(MouseListener) void removeMouseListener(MouseListener)	Add or remove a mouse listener to or from the component. Mouse listeners are notified when the user uses the mouse to interact with the listened-to component.

void addMouseMotionListener(MouseMotionListener) void removeMouseMotionListener(MouseMotionListener)	Add or remove a mouse motion listener to or from the component. Mouse motion listeners are notified when the user moves the mouse within the listened-to component's bounds.
void addKeyListener(KeyListener) void removeKeyListener(KeyListener)	Add or remove a key listener to or from the component. Key listeners are notified when the user types at the keyboard and the listened-to component has the keyboard focus.
void addComponentListener(ComponentListener) void removeComponentListener(ComponentListener)	Add or remove a component listener to or from the component. Component listeners are notified when the listened-to component is hidden, shown, moved, or resized.
boolean contains(int, int) boolean contains(Point)	Determine whether the specified point is within the component. The argument should be specified in terms of the component's coordinate system. The two `int` arguments specify *x* and *y* coordinates, respectively.
Component getComponentAt(int, int) Component getComponentAt(Point)	Return the component that contains the specified *x*, *y* position. The top-most child component is returned in the case where components overlap. This is determined by finding the component closest to the index 0 that claims to contain the given point via `Component.contains()`.
Component setComponentZOrder(component comp, int index)	Moves the specified component to the specified z-order index in the container. If the component is a child of some other container, it is removed from that container before being added to this container. The important difference between this method and `java.awt.Container.add(Component, int)` is that this method doesn't call `removeNotify` on the component while removing it from its previous container unless necessary and when allowed by the underlying native windowing system. This way, if the component has the keyboard focus, it maintains the focus when moved to the new position. **Note:** The z-order determines the order that components are painted. The component with the highest z-order paints first and the component with the lowest z-order paints last. Where components overlap, the component with the lower z-order paints over the component with the higher z-order.

Component getComponentZOrder(component comp)	Returns the z-order index of the component inside the container. The higher a component is in the z-order hierarchy, the lower its index. The component with the lowest z-order index is painted last, above all other child components.

Painting Components

Method	Purpose
void repaint() void repaint(int, int, int, int)	Request that all or part of the component be repainted. The four `int` arguments specify the bounds (*x*, *y*, width, height, in that order) of the rectangle to be painted.
void repaint(Rectangle)	Request that the specified area within the component be repainted.
void revalidate()	Request that the component and its affected containers be laid out again. You should not generally need to invoke this method unless you explicitly change a component's size/alignment hints after it's visible or change a containment hierarchy after it is visible. Always invoke `repaint` after `revalidate`.
void paintComponent(Graphics)	Paint the component. Override this method to implement painting for custom components.

Dealing with the Containment Hierarchy
(see Using Top-Level Containers for more information)

Method	Purpose
Component add(Component) Component add(Component, int) void add(Component, Object)	Add the specified component to this container. The one-argument version of this method adds the component to the end of the container. When present, the `int` argument indicates the new component's position within the container. When present, the `Object` argument provides layout constraints to the current layout manager.
void remove(int) void remove(Component) void removeAll()	Remove one of or all of the components from this container. When present, the `int` argument indicates the position within the container of the component to remove.
JRootPane getRootPane()	Get the root pane that contains the component.
Container getTopLevelAncestor()	Get the topmost container for the component — a `Window`, `Applet`, or null if the component has not been added to any container.
Container getParent()	Get the component's immediate container.
int getComponentCount()	Get the number of components in this container.
Component getComponent(int) Component[] getComponents()	Get the one of or all of the components in this container. The `int` argument indicates the position of the component to get.

Component getComponentZOrder(int) Component[] getComponentZOrder()	Returns the z-order index of the component inside the container. The higher a component is in the z-order hierarchy, the lower its index. The component with the lowest z-order index is painted last, above all other child components.

Laying Out Components
(see Laying Out Components Within a Container for more information)

Method	Purpose
void setPreferredSize(Dimension) void setMaximumSize(Dimension) void setMinimumSize(Dimension)	Set the component's preferred, maximum, or minimum size, measured in pixels. The preferred size indicates the best size for the component. The component should be no larger than its maximum size and no smaller than its minimum size. Be aware that these are hints only and might be ignored by certain layout managers.
Dimension getPreferredSize() Dimension getMaximumSize() Dimension getMinimumSize()	Get the preferred, maximum, or minimum size of the component, measured in pixels. Many JComponent classes have setter and getter methods. For those non-`JComponent` subclasses, which do not have the corresponding setter methods, you can set a component's preferred, maximum, or minimum size by creating a subclass and overriding these methods.
void setAlignmentX(float) void setAlignmentY(float)	Set the alignment along the *x*- or *y*- axis. These values indicate how the component would like to be aligned relative to other components. The value should be a number between 0 and 1 where 0 represents alignment along the origin, 1 is aligned the furthest away from the origin, and 0.5 is centered, and so on. Be aware that these are hints only and might be ignored by certain layout managers.
float getAlignmentX() float getAlignmentY()	Get the alignment of the component along the *x*- or *y*- axis. For non-`JComponent` subclasses, which do not have the corresponding setter methods, you can set a component's alignment by creating a subclass and overriding these methods.
void setLayout(LayoutManager) LayoutManager getLayout()	Set or get the component's layout manager. The layout manager is responsible for sizing and positioning the components within a container.
void applyComponentOrientation (ComponentOrientation) void setComponentOrientation (ComponentOrientation)	Set the `ComponentOrientation` property of this container and all the components contained within it. See Setting the Container's Orientation for more information.

Getting Size and Position Information

Method	Purpose
int getWidth() int getHeight()	Get the current width or height of the component measured in pixels.
Dimension getSize() Dimension getSize(Dimension)	Get the component's current size measured in pixels. When using the one-argument version of this method, the caller is responsible for creating the `Dimension` instance in which the result is returned.

int getX() int getY()	Get the current *x* or y coordinate of the component's origin relative to the parent's upper left corner measured in pixels.
Rectangle getBounds() Rectangle getBounds(Rectangle)	Get the bounds of the component measured in pixels. The bounds specify the component's width, height, and origin relative to its parent. When using the one-argument version of this method, the caller is responsible for creating the `Rectangle` instance in which the result is returned.
Point getLocation() Point getLocation(Point)	Gets the current location of the component relative to the parent's upper left corner measured in pixels. When using the one-argument version of `getLocation` method, the caller is responsible for creating the `Point` instance in which the result is returned.
Point getLocationOnScreen()	Returns the position relative to the upper left corner of the screen.
Insets getInsets()	Get the size of the component's border.

Specifying Absolute Size and Position
(see Doing Without a Layout Manager (Absolute Positioning) for more information)

Method	Purpose
void setLocation(int, int) void setLocation(Point)	Set the location of the component, in pixels, relative to the parent's upper left corner. The two `int` arguments specify *x* and *y*, in that order. Use these methods to position a component when you are not using a layout manager.
void setSize(int, int) void setSize(Dimension)	Set the size of the component measured in pixels. The two `int` arguments specify width and height, in that order. Use these methods to size a component when you are not using a layout manager.
void setBounds(int, int, int, int) void setBounds(Rectangle)	Set the size and location relative to the parent's upper left corner, in pixels, of the component. The four `int` arguments specify *x*, *y*, width, and height, in that order. Use these methods to position and size a component when you are not using a layout manager.

1.3 Using Text Components

Swing text components display text and optionally allow the user to edit the text. Programs need text components for tasks ranging from the straightforward (enter a word and press Enter) to the complex (display and edit styled text with embedded images in different language).

Swing provides six text components, along with supporting classes and interfaces that meet even the most complex text requirements. In spite of their different uses and capabilities, all Swing text components inherit from the same superclass, *JTextComponent*, which provides a highly-configurable and powerful foundation for text manipulation.

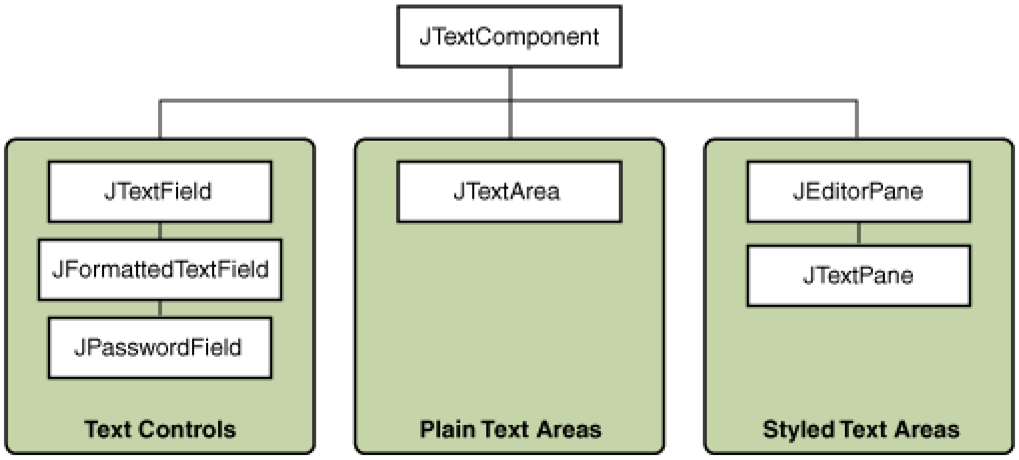
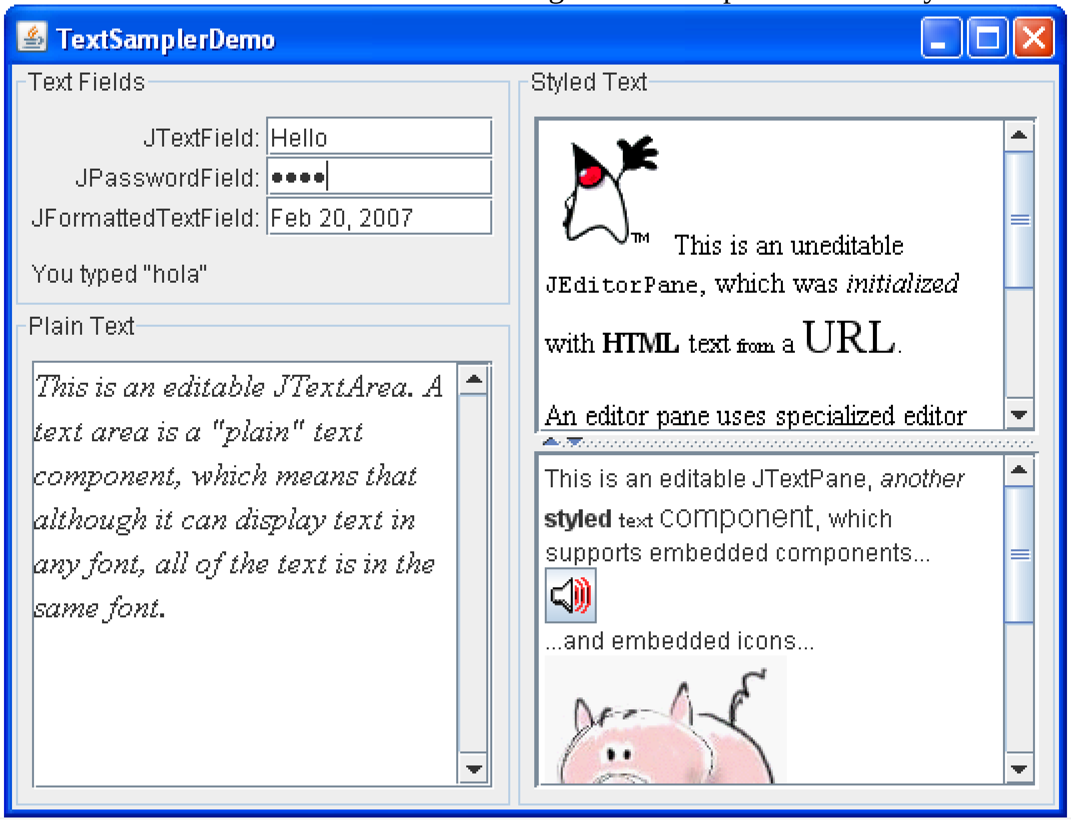


Fig : JTextComponent hierarchy



This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

This is an uneditable JEditorPane, which was *initialized* with **HTML** text from a URL.

An editor pane uses specialized editor

This is an editable JTextPane, *another* **styled** text component, which supports embedded components...

...and embedded icons...

Group	Description	Swing Classes
Text Controls	Also known simply as text fields, text controls can display only one line of editable text. Like buttons, they generate action events. Use them to get a small amount of textual information from the user and perform an action after the text entry is complete.	`JTextField` and its subclasses `JPasswordField` and `JFormattedTextField`

Plain Text Areas	`JTextArea` can display multiple lines of editable text. Although a text area can display text in any font, all of the text is in the same font. Use a text area to allow the user to enter unformatted text of any length or to display unformatted help information.	`JTextArea`
Styled Text Areas	A styled text component can display editable text using more than one font. Some styled text components allow embedded images and even embedded components. Styled text components are powerful and multi-faceted components suitable for high-end needs, and offer more avenues for customization than the other text components. Because they are so powerful and flexible, styled text components typically require more initial programming to set up and use. One exception is that editor panes can be easily loaded with formatted text from a URL, which makes them useful for displaying uneditable help information.	`JEditorPane` and its subclass `JTextPane`

Text Component Features

The JTextComponent class is the foundation for Swing text components. This class provides the following customizable features for all of its descendants:

- A model, known as a *document*, that manages the component's content.
- A view, which displays the component on screen.
- A controller, known as an *editor kit*, that reads and writes text and implements editing capabilities with actions.
- Support for infinite undo and redo.
- A pluggable caret and support for caret change listeners and navigation filters.

The Text Component API

his section lists commonly used parts of the API that are shared by text components. Much of this API is defined by the `JTextComponent` class. Text Component Features discusses how to use some of this API.

The JComponent Class describes the API that text components inherit from `JComponent`. For information about the API related to specific text components, see the how-to page for that component: text field, password field, formatted text field, text area, or editor pane and text pane.

For complete details about the text API, see the API documentation for `JTextComponent` and for the various classes and interfaces in the text package.

The API listed in this section includes the following categories:

- Setting Attributes

- Manipulating the Selection
- Converting Positions Between the Model and the View
- Text Editing Commands
- Classes and Interfaces That Represent Documents
- Working With Documents
- Manipulating Carets and Selection Highlighters
- Reading and Writing Text

Setting Attributes
These methods are defined in the `JTextComponent` class.

Method	Description
void setEditable(boolean) boolean isEditable()	Sets or indicates whether the user can edit the text in the text component.
void setDragEnabled(boolean) boolean getDragEnabled()	Sets or gets the `dragEnabled` property, which must be true to enable drag handling on this component. The default value is false. See Drag and Drop and Data Transfer for more details.
void setDisabledTextColor(Color) Color getDisabledTextColor()	Sets or gets the color used to display text when the text component is disabled.
void setMargin(Insets) Insets getMargin()	Sets or gets the margin between the text and the text component's border.

Manipulating the Selection
These methods are defined in the `JTextComponent` class.

Method	Description
String getSelectedText()	Gets the currently selected text.
void selectAll() void select(int, int)	Selects all text or selects text within a start and end range.
void setSelectionStart(int) void setSelectionEnd(int) int getSelectionStart() int getSelectionEnd()	Sets or gets the extent of the current selection by index.
void setSelectedTextColor(Color) Color getSelectedTextColor()	Sets or gets the color of selected text.
void setSelectionColor(Color) Color getSelectionColor()	Sets or gets the background color of selected text.

Converting Positions Between the Model and the View
These methods are defined in the `JTextComponent` class.

Method	Description
int viewToModel(Point)	Converts the specified point in the view coordinate system to a position within the text.
Rectangle modelToView(int)	Converts the specified position within the text to a rectangle in the view coordinate system.

Text Editing Commands

Class or Method	Description
void cut() void copy() void paste() void replaceSelection(String) (in `JTextComponent`)	Cuts, copies, and pastes text using the system clipboard, or replaces the selected text with the string specified by an argument, respectively.

EditorKit	Provides a text component's view factory, document, caret, and actions, as well as reading and writing documents of a particular format.
DefaultEditorKit	A concrete subclass of `EditorKit` that provides the basic text editing capabilities.
StyledEditorKit	A subclass of `Default EditorKit` that provides additional editing capabilities for styled text.
String *xxxx*Action *(in* `DefaultEditorKit`*)*	The names of all the actions supported by the default editor kit. See Associating Text Actions with Menus and Buttons.
BeepAction CopyAction CutAction DefaultKeyTypedAction InsertBreakAction InsertContentAction InsertTabAction PasteAction *(in* `DefaultEditorKit`*)*	Inner classes that implement various text editing commands.
AlignmentAction BoldAction FontFamilyAction FontSizeAction ForegroundAction ItalicAction StyledTextAction UnderlineAction *(in* `StyledEditorKit`*)*	Inner classes that implement various editing commands for styled text.
Action[] getActions() *(in* `JTextComponent`*)*	Gets the actions supported by this component. This method gets the array of actions from the editor kit if one is used by the component.
InputMap getInputMap() *(in* `JComponent`*)*	Gets the input map that binds key strokes to actions. See Associating Text Actions with Key Strokes.
void put(KeyStroke, Object) *(in* `InputMap`*)*	Binds the specified key to the specified action. You generally specify the action by its name, which for standard editing actions is represented by a string constant such as `DefaultEditorKit.backwardAction`.

Classes and Interfaces That Represent Documents

Interface or Class	Description
Document	An interface that defines the API that must be implemented by all documents.
AbstractDocument	An abstract superclass implementation of the `Document` interface. This is the superclass for all documents provided by the Swing text package.
PlainDocument	A class that implements the `Document` interface. This is the default document for the plain text components (text field, password field, and text area). Additionally, this class is used by the editor panes and text panes when loading plain text or text of an unknown format.
StyledDocument	A `Document` subinterface. Defines the API that must be implemented by documents that support styled text. `JTextPane` requires that its document be of this type.
DefaultStyledDocument	A class that implements the `StyledDocument` interface. The default document for `JTextPane`.

Working With Documents

Class or Method	Description

DocumentFilter	The superclass of all document filters. You can use a document filter to change what gets inserted or removed from a document, without having to implement a document yourself. See Implementing a Document Filter.
void setDocumentFilter(DocumentFilter) (in `AbstractDocument`)	Sets the document filter.
void setDocument(Document) Document getDocument() (in `JTextComponent`)	Sets or gets the document for a text component.
Document createDefaultModel() (in `JTextField`)	Creates a default PlainDocument model. Override this method to create a custom document instead of the default `PlainDocument`.
void addDocumentListener(DocumentListener) void removeDocumentListener(DocumentListener) (in `Document`)	Adds or removes a document listener. See Listening for Changes on a Document.
void addUndoableEditListener(UndoableEditListener) void removeUndoableEditListener(UndoableEditlistener) (in `Document`)	Adds or removes an undoable edit listener. Undoable edit listeners are used in Implementing Undo and Redo.
int getLength() Position getStartPosition() Position getEndPosition() String getText(int, int) (in `Document`)	`Document` methods that return various descriptive information about the document.
Object getProperty(Object) void putProperty(Object, Object) (in `Document`) void setDocumentProperties(Dictionary) Dictionary getDocumentProperties() (in `AbstractDocument`)	A `Document` maintains a set of properties that you can manipulate with these methods.

Interface, Class, or Method	**Description**
Caret	An interface that defines the API for objects that represent an insertion point within documents.
DefaultCaret	The default caret used by all text components.
void setCaret(Caret) Caret getCaret()	Sets or gets the caret object used by a text component.
void setCaretColor(Color) Color getCaretColor()	Sets or gets the color of the caret.
void setCaretPosition(int) void moveCaretPosition(int) int getCaretPosition()	Sets or gets the current position of the caret within the document.
void addCaretListener(CaretListener) void removeCaretListener(CaretListener)	Adds or removes a caret listener from a text component.
NavigationFilter	The superclass for all navigation filters. A navigation filter lets you modify caret changes that are about to occur for a text component.
void setNavigationFilter(NavigationFilter)	Attaches a navigation filter to a text component.

Highlighter	An interface that defines the API for objects used to highlight the current selection.
DefaultHighlighter	The default highlighter used by all text components.
void setHighlighter(Highlighter) Highlighter getHighlighter()	Sets or gets the highlighter used by a text component.

Reading and Writing Text

Method	Description
void read(Reader, Object) void write(Writer) (in `JTextComponent`)	Reads or writes text.
void read(Reader, Document, int) void read(InputStream, Document, int) (in `EditorKit`)	Reads text from a stream into a document.
void write(Writer, Document, int, int) void write(OutputStream, Document, int, int) (in `EditorKit`)	Writes text from a document to a stream.

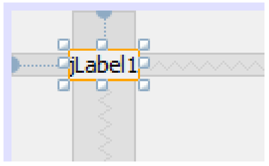
1.4 Using Swing Controls

1.4.1 Labels

With the JLabel class, you can display unselectable text and images. If you need to create a component that displays a string, an image, or both, you can do so by using or extending JLabel.

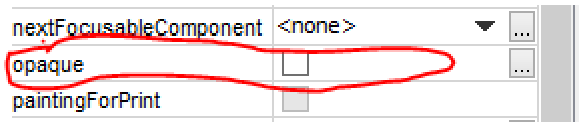
Add JLabel to the a form

- Drag Label from the Swing Control group of the palette window drop on to the form



Note that labels are not opaque by default. If you need to paint the label's background, it is recommended that you turn its opacity property to "true".

```
label.setOpaque(true);
```



The API for using labels:

- Setting or Getting the Label's Contents
- Fine Tuning the Label's Appearance
- Supporting Accessibility

Setting or Getting the Label's Contents

Method or Constructor	Purpose

JLabel(Icon)
JLabel(Icon, int)
JLabel(String)
JLabel(String, Icon, int)
JLabel(String, int)
JLabel()
	Creates a `JLabel` instance, initializing it to have the specified text/image/alignment. The `int` argument specifies the horizontal alignment of the label's contents within its drawing area. The horizontal alignment must be one of the following constants defined in the `SwingConstants` interface (which `JLabel` implements): `LEFT`, `CENTER`, `RIGHT`, `LEADING`, or `TRAILING`. For ease of localization, we strongly recommend using `LEADING` and `TRAILING`, rather than `LEFT` and `RIGHT`.

void setText(String) String getText()	Sets or gets the text displayed by the label. You can use HTML tags to format the text, as described in Using HTML in Swing Components.
void setIcon(Icon) Icon getIcon()	Sets or gets the image displayed by the label.
void setDisplayedMnemonic(char) char getDisplayedMnemonic()	Sets or gets the letter that should look like a keyboard alternative. This is helpful when a label describes a component (such as a text field) that has a keyboard alternative but cannot display it. If the labelFor property is also set (using `setLabelFor`), then when the user activates the mnemonic, the keyboard focus is transferred to the component specified by the labelFor property.
void setDisplayedMnemonicIndex(int) int getDisplayedMnemonicIndex()	Sets or gets a hint as to which character in the text should be decorated to represent the mnemonic. This is useful when you have two instances of the same character and wish to decorate the second instance. For example, `setDisplayedMnemonicIndex(5)` decorates the character that is at position 5 (that is, the 6th character in the text). Not all types of look and feel may support this feature.
void setDisabledIcon(Icon) Icon getDisabledIcon()	Sets or gets the image displayed by the label when it is disabled. If you do not specify a disabled image, then the look and feel creates one by manipulating the default image.

Fine Tuning the Label's Appearance

Method	Purpose
void setHorizontalAlignment(int) void setVerticalAlignment(int) int getHorizontalAlignment() int getVerticalAlignment()	Sets or gets the area on the label where its contents should be placed. The `SwingConstants` interface defines five possible values for horizontal alignment: `LEFT`, `CENTER` (the default for image-only labels), `RIGHT`, `LEADING` (the default for text-only labels), `TRAILING`. For vertical alignment: `TOP`, `CENTER` (the default), and `BOTTOM`.
void setHorizontalTextPosition(int) void setVerticalTextPosition(int) int getHorizontalTextPosition() int getVerticalTextPosition()	Sets or gets the location where the label's text will be placed, relative to the label's image. The `SwingConstants` interface defines five possible values for horizontal position: `LEADING`, `LEFT`, `CENTER`, `RIGHT`, and `TRAILING` (the default). For vertical position: `TOP`, `CENTER` (the default), and `BOTTOM`.
void setIconTextGap(int) int getIconTextGap()	Sets or gets the number of pixels between the label's text and its image.

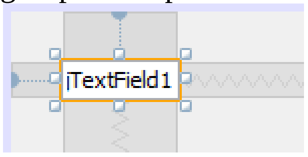
Supporting Accessibility

Method	Purpose
void setLabelFor(Component) Component getLabelFor()	Sets or gets which component the label describes.

1.4.2 Text Fields

A text field is a basic text control that enables the user to type a small amount of text.
Add JTextField to the a form

- Drag Text Field from the Swing Control group of the palette window drop on to the form



The Text Field API

Setting or Obtaining the Field's Contents

Method or Constructor	Purpose

JTextField()
JTextField(String)
JTextField(String, int)
JTextField(int)
	Creates a text field. When present, the `int` argument specifies the desired width in columns. The `String` argument contains the field's initial text.

void setText(String) String getText() *(defined in `JTextComponent`)*	Sets or obtains the text displayed by the text field.

Fine Tuning the Field's Appearance

Method	Purpose
void setEditable(boolean) boolean isEditable() *(defined in `JTextComponent`)*	Sets or indicates whether the user can edit the text in the text field.
void setColumns(int); int getColumns()	Sets or obtains the number of columns displayed by the text field. This is really just a hint for computing the field's preferred width.
void setHorizontalAlignment(int); int getHorizontalAlignment()	Sets or obtains how the text is aligned horizontally within its area. You can use `JTextField.LEADING`, `JTextField.CENTER`, and `JTextField.TRAILING` for arguments.

Implementing the Field's Functionality

Method	Purpose
void addActionListener(ActionListener) void removeActionListener(ActionListener)	Adds or removes an action listener.
void selectAll() *(defined in `JTextComponent`)*	Selects all characters in the text field.

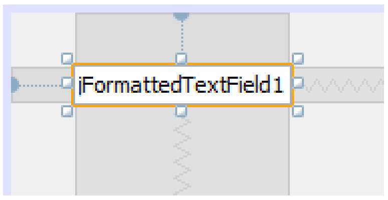
1.4.3 Formatted Text Fields

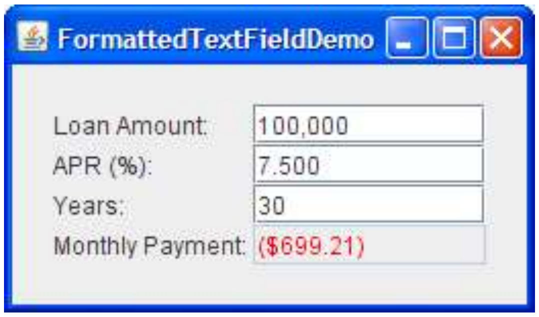
Formatted text fields provide a way for developers to specify the valid set of characters that can be typed in a text field. Specifically, the *JFormattedTextField* class adds a formatter and an object value to the features inherited from the *JTextField* class. The formatter translates the field's value into the text it displays, and the text into the field's value.

Using the formatters that Swing provides, you can set up formatted text fields to type dates and numbers in localized formats. Another kind of formatter enables you to use a character mask to specify the set of characters that can be typed at each position in the field. For example, you can specify a mask for typing phone numbers in a particular format, such as (XX) X-XX-XX-XX-XX.

Add *JFormattedTextField* to the a form

- ➢ Drag *Formatted Field* from the Swing Control group of the palette window drop on to the form

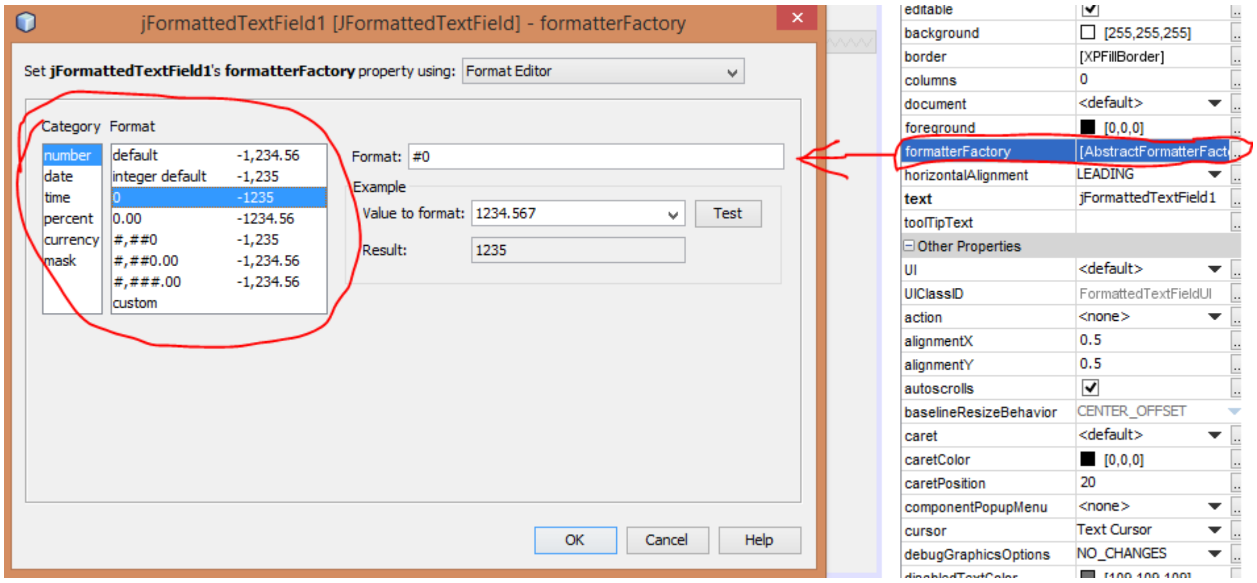




Loan Amount: 100,000
APR (%): 7.500
Years: 30
Monthly Payment: ($699.21)

To set the input format, use the formatterFactory property of the Formatted Text Field.
To do

- Select the jFormattedTextField1 => properties window =>set formatterFactory property


jFormattedTextField1 [JFormattedTextField] - formatterFactory
OK
Cancel
Help
Test

Set **jFormattedTextField1**'s **formatterFactory** property using: Format Editor

Category	Format	
number	default	-1,234.56
date	integer default	-1,235
time	0	-1235
percent	0.00	-1234.56
currency	#,##0	-1,235
mask	#,##0.00	-1,234.56
	#,###.00	-1,234.56
	custom	

Format: #0
Example
Value to format: 1234.567
Result: 1235

editable	✓
background	☐ [255,255,255]
border	[XPFillBorder]
columns	0
document	<default>
foreground	■ [0,0,0]
formatterFactory	[AbstractFormatterFact
horizontalAlignment	LEADING
text	jFormattedTextField1
toolTipText	
Other Properties	
UI	<default>
UIClassID	FormattedTextFieldUI
action	<none>
alignmentX	0.5
alignmentY	0.5
autoscrolls	✓
baselineResizeBehavior	CENTER_OFFSET
caret	<default>
caretColor	■ [0,0,0]
caretPosition	20
componentPopupMenu	<none>
cursor	Text Cursor
debugGraphicsOptions	NO_CHANGES

Formatted Text Field API

The following tables list some of the commonly used APIs for using formatted text fields.

- Classes Related to Formatted Text Fields
- JFormattedTextField Methods
- DefaultFormatter Options

Classes Related to Formatted Text Fields

Class or Interface	Purpose
JFormattedTextField	Subclass of `JTextField` that supports formatting arbitrary values.
JFormattedTextField.AbstractFormatter	The superclass of all formatters for `JFormattedTextField`. A formatter enforces editing policies and navigation policies, handles string-to-object conversions, and manipulates the `JFormattedTextField` as necessary to enforce the desired policy.
JFormattedTextField.AbstractFormatterFactory	The superclass of all formatter factories. Each `JFormattedTextField` uses a formatter factory to obtain the formatter that best corresponds to the text field's state.

DefaultFormatterFactory	The formatter factory normally used. Provides formatters based on details such as the passed-in parameters and focus state.
DefaultFormatter	Subclass of `JFormattedTextField.AbstractFormatter` that formats arbitrary objects by using the `toString` method.
MaskFormatter	Subclass of `DefaultFormatter` that formats and edits strings using a specified character mask. (For example, seven-digit phone numbers can be specified by using "###-####".)
InternationalFormatter	Subclass of `DefaultFormatter` that uses an instance of `java.text.Format` to handle conversion to and from a `String`.
NumberFormatter	Subclass of `InternationalFormatter` that supports number formats by using an instance of `NumberFormat`.
DateFormatter	Subclass of `InternationalFormatter` that supports date formats by using an instance of `DateFormat`.

JFormattedTextField Methods

Method or Constructor	Purpose

JFormattedTextField()
JFormattedTextField(Object)
JFormattedTextField(Format)
JFormattedTextField(AbstractFormatter)
JFormattedTextField(AbstractFormatterFactory)
JFormattedTextField(AbstractFormatterFactory, Object)
	Creates a new formatted text field. The `Object` argument, if present, specifies the initial value of the field and causes an appropriate formatter factory to be created. The `Format` or `AbstractFormatter` argument specifies the format or formatter to be used for the field, and causes an appropriate formatter factory to be created. The `AbstractFormatterFactory` argument specifies the formatter factory to be used, which determines which formatters are used for the field.

void setValue(Object) Object getValue()	Sets or obtains the value of the formatted text field. You must cast the return type based on how the `JFormattedTextField` has been configured. If the formatter has not been set yet, calling `setValue` sets the formatter to one returned by the field's formatter factory.
void setFormatterFactory(AbstractFormatterFactory)	Sets the object that determines the formatters used for the formatted text field. The object is often an instance of the `DefaultFormatterFactory` class.
AbstractFormatter getFormatter()	Obtains the formatter of the formatted text field. The formatter is often an instance of the `DefaultFormatter` class.
void setFocusLostBehavior(int)	Specifies the outcome of a field losing the focus. Possible values are defined in `JFormattedTextField` as `COMMIT_OR_REVERT` (the default), `COMMIT` (commit if valid, otherwise leave everything the same), `PERSIST` (do nothing), and `REVERT` (change the text to reflect the value).
void commitEdit()	Sets the value to the object represented by the field's text, as determined by the field's formatter. If the text is invalid, the value remains the same and a `ParseException` is thrown.
boolean isEditValid()	Returns true if the formatter considers the current text to be valid, as determined by the field's formatter.

DefaultFormatter Options

Method	Purpose
void setCommitsOnValidEdit(boolean) boolean getCommitsOnValidEdit()	Sets or obtains values when edits are pushed back to the `JFormattedTextField`. If `true`, `commitEdit` is called after every valid edit. This property is `false` by default.

void setOverwriteMode(boolean) boolean getOverwriteMode()	Sets or obtains the behavior when inserting characters. If `true`, new characters overwrite existing characters in the model as they are inserted. The default value of this property is `true` in `DefaultFormatter` (and thus in `MaskFormatter`) and `false` in `InternationalFormatter` (and thus in `DateFormatter` and `NumberFormatter`).
void setAllowsInvalid(boolean) boolean getAllowsInvalid()	Sets or interprets whether the value being edited is allowed to be invalid for a length of time. It is often convenient to enable the user to type invalid values until the `commitEdit` method is attempted. `DefaultFormatter` initializes this property to `true`. Of the standard Swing formatters, only `MaskFormatter` sets this property to `false`.

The following table shows the characters that you can use in the formatting mask:

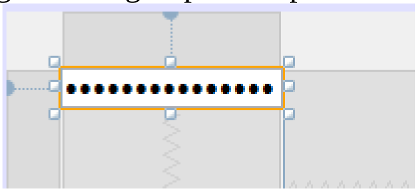
Character	Description
#	Any valid number (`Character.isDigit`).
' *(single quote)*	Escape character, used to escape any of the special formatting characters.
U	Any character (`Character.isLetter`). All lowercase letters are mapped to uppercase.
L	Any character (`Character.isLetter`). All uppercase letters are mapped to lowercase.
A	Any character or number (`Character.isLetter` or `Character.isDigit`).
?	Any character (`Character.isLetter`).
*	Anything.
H	Any hex character (0-9, a-f or A-F).

1.4.4 Password Fields

The JPasswordField class, a subclass of JTextField, provides specialized text fields for password entry. For security reasons, a password field does not show the characters that the user types. Instead, the field displays a character different from the one typed, such as an asterisk '*'. As another security precaution, a password field stores its value as an array of characters, rather than as a string.

Add *JPasswordField* to the a form

- Drag *Password Field* from the Swing Control group of the palette window drop on to the form



The Password Field API

Commonly Used JPasswordField Constructors and Methods	
Constructor or Method	**Purpose**

JPasswordField()
JPasswordField(String)
JPasswordField(String, int)
JPasswordField(int)
JPasswordField(Document, String, int)
	Creates a password field. When present, the `int` argument specifies the desired width in columns. The `String` argument contains the field's initial text. The `Document` argument provides a custom model for the field.

char[] getPassword()	Returns the password as an array of characters.
void setEchoChar(char) char getEchoChar()	Sets or gets the echo character which is displayed instead of the actual characters typed by the user.
void addActionListener(ActionListener) void removeActionListener(ActionListener) *(defined in `JTextField`)*	Adds or removes an action listener.
void selectAll() *(defined in `JTextComponent`)*	Selects all characters in the password field.

```
char[] input = passwordField.getPassword();
        if (isPasswordCorrect(input)) {
            JOptionPane.showMessageDialog(controllingFrame,
                "Success! You typed the right password.");
        } else {
            JOptionPane.showMessageDialog(controllingFrame,
                "Invalid password. Try again.",
                "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }

        //Zero out the possible password, for security.
        Arrays.fill(input, '0');

        passwordField.selectAll();
        resetFocus();
    }
```

1.4.5 Text Areas

The JTextArea class provides a component that displays multiple lines of text and optionally allows the user to edit the text.