Java Swing: Conheça os componentes JTextField e JFormattedTextField

[Facebook](https://www.facebook.com/sharer/sharer.php?u=http://www.devmedia.com.br/java-swing-conheca-os-componentes-jtextfield-e-jformattedtextfield/30981" \o "Compartilhe no Facebook" \t "_blank) [Twitter](https://twitter.com/home?status=Java%20Swing:%20Conhe%E7a%20os%20componentes%20JTextField%20e%20JFormattedTextField%20%0D%0Ahttp://www.devmedia.com.br/java-swing-conheca-os-componentes-jtextfield-e-jformattedtextfield/30981" \t "_blank" \o "Compartilhe no Twitter)

 (8)  (0)

Veja neste artigo como utilizar e customizar dois dos principais componentes da biblioteca Java Swing, o JTextField e o JFormattedTextField.

O [Swing é um framework](http://www.devmedia.com.br/curso/desenvolvendo-uma-aplicacao-completa-utilizando-swing/187) que disponibiliza um conjunto de elementos gráficos para ser utilizado na plataforma Java. O Swing é compatível com o [Abstract Window Toolkit (AWT)](http://www.devmedia.com.br/classes-awt-para-criacao-de-jogos-curso-manipulacao-de-imagens-com-java-parte-6/17870), mas trabalha de forma totalmente diferente. A **API Swing**, diferente do **AWT**, não delega a tarefa de renderização ao sistema operacional, ele renderiza os elementos por conta própria. Como a AWT é uma biblioteca de baixo-nível que depende de código nativo da plataforma ela traz alguns problemas de compatibilidade entre as plataformas, fazendo com que nem sempre o programa tenha a aparência desejada em todos os sistemas operacionais. Além disso, o **Swing** é mais completo e os programas têm uma aparência muito parecida, independente do sistema operacional que está sendo utilizado, possui uma enorme gama de controles extras disponíveis, tais como áreas de texto que nativamente podem mostrar conteúdo como RTF ou [HTML](http://www.devmedia.com.br/curso/curso-de-html-basico/371), botões com suporte a imagens, sliders, selecionadores de cores, alteração do tipo de borda para os componentes, maior controle de como desenhar os mínimos detalhes de apresentação e muito mais. No entanto, a performance é um pouco pior devido a alta abstração, consumindo assim mais memória RAM.

Confira os [cursos de Java](http://www.devmedia.com.br/cursos/java) da DevMedia e aprofunde seu conhecimento sobre essa linguagem.

Os principais componentes do Swing são resumidos abaixo:

* **JFrame** representa a janela do programa com barra de título, ícone, botões de comando, etc. Entre os principais métodos temos o *pack()* que compacta a janela para o tamanho dos componentes, *setSize(int, int)* que define a largura e altura da janela, *setLocation(int, int)* que define a posição da janela na tela (x,y), *setBounds(int, int, int, int)* que define posição e tamanho, *setVisible(boolean)* que exibe a janela e *setDefaultCloseOperation(int)* que define o que ocorre quando o usuário tenta fechar a janela (as opções são: *DO\_NOTHING\_ON\_CLOSE*, *HIDE\_ON\_CLOSE*, *DISPOSE\_ON\_CLOSE*, *EXIT\_ON\_CLOSE*).
* **JPanel** representa um tipo básico de container para inserção de componentes. Entre os principais métodos temos *add(Component, int)* que adiciona o componente definindo sua posição e *setLayout(LayoutManagaer)* que altera o tipo de layout.
* **JLabel** representa um rótulo de texto. Entre os principais métodos temos o *setText(String)* que altera o texto e *getText()* que retorna o texto atual.
* **JTextFiel** representa um campo de texto onde o usuário pode informar um texto em uma linha. Entre os principais métodos temos *setText(String)* que altera o texto e *getText()* que retorna o texto atual.
* **JPasswordField** representa um campo de texto protegido, subclasse de JTextField. O principal método é o *setEchoChar(char)* que define o caractere que aparece ao digitar um texto.
* **JTextArea** representa uma caixa onde o usuário pode informar várias linhas de texto. Entre os principais métodos temos o *setText(String)* que altera o texto, *getText()* que retorna o texto atual, *getSelectedText()* que retorna o texto selecionado pelo usuário e *insert(String, int)* que insere um texto na posição especificada.
* **JCheckBox** representa uma caixa de seleção e permite selecionar ou não uma opção. Entre os principais métodos temos o *setSelected(boolean)* que altera o estado da caixa de seleção e o método *isSelected()* que retorna *true* se a caixa estiver marcada e *false* se não estiver marcada.
* **JRadioButton** representa um componente que permite seleciona uma entre diversas opções. O JRadioButton é semelhante ao *JCheckBox*, inclusive com os mesmos construtores e métodos.
* **JComboBox** representa uma caixa de combinação, da qual o usuário pode selecionar uma opção. Entre os principais métodos temos o *addItem(Object)* que adiciona um item à lista de opções, *setEditable(boolean)* que permite ao usuário digitar uma opção, *getSelectedIndex()* que retorna a posição do item atualmente selecionado, *getSelectedItem()* que retorna o texto do item atualmente selecionado, *setSelectedIndex(int)* que seleciona o item da posição especificada e *setSelectedIndex(Object)* que seleciona o objeto especificado na lista.
* **JList** representa uma lista de opções que permite a seleção de mais de um item simultaneamente. Entre os principais métodos temos o *setListData(Object[])* que preenche ou altera os itens de uma lista, *getSelectedValues()* que retorna um array de objetos contendo itens selecionados na lista.
* **JButton** representa um botão destinado a executar uma ação. Entre os principais métodos temos o *setText(String)* que altera o texto do botão e *setIcon(Icon)* que altera o ícone do botão.

Na **Figura 1**podemos visualizar cada um dos componentes discutidos acima, respectivamente.

 Principais
componentes do framework Swing 

**Figura 1**. Principais componentes do framework Swing.

Existem ainda diversos outros componentes Swing como o JSlider, JProgressBar, JToolBar, JTabbedPane entre outros.

No restante do artigo abordaremos dois componentes muito utilizados no framework Swing, veremos como podemos customizar o JTextField limitando o número de caracteres permitidos e como implementar máscaras no JFormattedTextField.

**Limitando caracteres com JTextField**

Controlar a entrada de texto num componente *JTextField* não é tão simples e ainda causa muitas dúvidas para os desenvolvedores que trabalham com Swing em java. A classe *javax.swing.JTextField* é uma subclasse da classe abstrata JTextComponent e o modelo de dados padrão é um objeto da classe *PlainDocument*, subclasse de *AbstractDocument* que implementa a interface *Document*. Um dos métodos da classe *PlainDocument* é o *insertString()* que é herdado da superclasse abstrata *AbstractDocument*. A assinatura do método é:

public void insertString(int offset, String str, AttributeSet attr) throws BadLocationException

O argumento *offset* indica o deslocamento (a posição) inicial onde o objeto String *str* deverá ser inserido. O *AttributeSet* seta os atributos do *JTextField,* como os estilos.

Sempre que o conteúdo de um *JTextField* é modificado, o método *insertString()* é invocado.

Segue na **Listagem 1** um exemplo de como poderíamos utilizar o método *insertString* para criar um novo modelos de dados que seja capaz de restringir a quantidade de caracteres aceitos no modelo de dados de qualquer *JTextField*.

**Listagem 1**. Criando um novo modelo de dados para restringir o número de caracteres.

package com.exemplo;

import javax.swing.text.AttributeSet;

import javax.swing.text.BadLocationException;

import javax.swing.text.PlainDocument;

public class TamanhoFixoJTextField extends PlainDocument {

private int tamMax;

public TamanhoFixoJTextField(int tamMax) {

super();

this.tamMax = tamMax;

}

public void insertString(int offset, String str, AttributeSet attr)

throws BadLocationException {

if (str == null)

return;

//Define a condição para aceitar qualquer número de caracteres

if (tamMax <= 0)

{

super.insertString(offset, str, attr);

return;

}

int tam = (getLength() + str.length());

//Se o tamanho final for menor, chama insertString() aceitando a String

if (tam <= tamMax)

super.insertString(offset, str, attr);

}

}

O argumento recebido pelo construtor define o número máximo de caracteres aceitos. A chamada a *super()* no construtor permite que a classe ancestral faça todas as inicializações necessárias. A variável de instância *tamMax* recebe o valor do argumento do construtor. O método *insertString()* nos permite sobrepor o método definido na superclasse para que possamos defini-lo como quisermos. A exceção *BadLocationException* é disparada quando a posição de inserção na String é inválida. No corpo do método *insertString* primeiramente garantimos que não estamos recebendo uma String nula. Após isso verificamos o valor máximo de caracteres a serem aceitos, se este valor for menor ou igual a zero, tem-se que essa classe terá um comportamento igual a sua ancestral *PlainDocument*, chamando o método *insertString()* da superclasse e simplesmente a retornando. Dessa forma, se o máximo de caracteres for menor ou igual zero, o número de caracteres aceitos será indeterminado. A classe *AbstractDocument* ainda possui alguns métodos que permitem o acesso aos dados que formam o conteúdo do *JTextField*, são eles: *getLength()* ou ainda *getText()*. O método *getLength()* permite que possamos acessar o número de caracteres atual do conteúdo. O conteúdo pode ser acessado diretamente através de *getText()*. Portanto, no código acima temos "*int tam = (getLength() + str.length())*" em que *tam* é a soma do comprimento atual do conteúdo mais o comprimento da String *str* que está por ser inserida. Se a soma for menor ou igual ao número máximo de caracteres, aceitaremos *str* através da chamada ao método *insertString* da superclasse, caso contrário não ocorrerá a chamada e o conteúdo a ser inserido será ignorado.

Para testar o código presente na **Listagem 1,** vamos fazer um exemplo utilizando um *JTextField* que segue esse modelo. Na **Listagem 2** segue um exemplo.

**Listagem 2**. Testando o JTextField utilizando o modelo de dados criado.

package com.exemplo;

import javax.swing.JFrame;

import javax.swing.JTextField;

public class TestandoJTextField extends JFrame {

private static final long serialVersionUID = 1L;

private JTextField txtExemplo;

public static void main(String[] args)

{

TestandoJTextField field = new TestandoJTextField();

field.testaJTextField();

}

private void testaJTextField() {

this.setTitle("Exemplo");

this.setSize(200, 180);

//Definimos o tamanho padrão do JTextField

txtExemplo = new JTextField(10);

//Passamos para o construtor o número máximo de caracteres aceitos

txtExemplo.setDocument(new TamanhoFixoJTextField(5));

this.getContentPane().add(txtExemplo, "North");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

}

}

No exemplo mostrado no código acima, tentamos adicionar uma string na caixa de texto do programa, ou seja, o nome “Rafael”, porém como o nosso componente aceita no máximo cinco caracteres não conseguimos digitar o nome inteiro. Segue na **Figura 2** a demonstração do programa funcionando e permitindo um número máximo de caracteres.

 Executando o
programa e digitando uma string 

**Figura 2**. Executando o programa e digitando uma string.

Um pequeno problema é quando desejamos colar na nossa *JTextField* customizada uma string maior que a permitida. Se colarmos uma string que o número máximo de caracteres seja permitido não será um problema. Por exemplo, se tentarmos copiar a String “Marco” e colar no *JTextField* ele aceitará normalmente, conforme ilustra a **Figura 3.**

 Colando um texto
com cinco caracteres 

**Figura 3**. Colando um texto com cinco caracteres.

No entanto, se copiarmos a string “Rafael” e tentarmos colar veremos que essa string não será colada no *JTextField*, isso acontece porque esta string ultrapassa o número máximo de caracteres permitidos e portanto a string não é aceita. O ideal seria o campo de texto aceitar o número mínimo de caracteres e desprezar o restante, dessa forma, a string mostrada seria “Rafae”. Para modificar isso devemos fazer algumas modificações no código. Segue na **Listagem 3** o novo código com comentários das modificações.

**Listagem 3**. Modificando o código para contemplar as novas configurações.

package com.exemplo;

import javax.swing.text.AttributeSet;

import javax.swing.text.BadLocationException;

import javax.swing.text.PlainDocument;

public class TamanhoFixoJTextField extends PlainDocument {

private int tamMax;

public TamanhoFixoJTextField(int tamMax) {

super();

this.tamMax = tamMax;

}

public void insertString(int offset, String str, AttributeSet attr)

throws BadLocationException {

if (str == null)

return;

//Define a condição para aceitar qualquer número de caracteres

if (tamMax <= 0)

{

super.insertString(offset, str, attr);

return;

}

int tam = (getLength() + str.length());

//Se o tamanho final for menor, chama insertString() aceitando a String

if (tam <= tamMax) {

super.insertString(offset, str, attr);

} else {

//Caso contrário, limita a string e envia para insertString() que aceita a string

if (getLength() == tamMax) return;

String novaStr = str.substring(0, (tamMax - getLength()));

super.insertString(offset, novaStr, attr);

}

}

}

O bloco else adicionado no final do código acima primeiramente verifica se o comprimento atual do conteúdo é igual ao comprimento máximo, então simplesmente retornamos, caso contrário criamos um novo objeto String contendo apenas os caracteres de str que deveria caber no JTextField.

Agora se tentarmos copiar a String “Rafael” e colarmos no *JTextField* teremos como resultado a string limitada adicionada no campo, conforme mostra a **Figura 4** abaixo.

Colando uma
string maior que o permitido no campo 

**Figura 4**. Colando uma string maior que o permitido no campo.

Outra situação muito comum ao trabalharmos com campos de entrada é manipularmos alguns tipos de textos com valores pré-determinados como, por exemplo, CPF, CNPJ, CEP, etc. Na próxima seção veremos como fazer isso utilizando outro componente, o *JFormattedTextField* que é semelhante ao JTextField, porém mais voltado para campos formatados.

**Formatando campos com JFormattedTextField**

As máscaras são muito utilizadas em sistemas comerciais, pois elas ajudam na padronização da visualização de dados. Um exemplo de máscara é o telefone: "(51)3636-0000" ou o CEP: "92-110.310". Tudo que for digitado ficará entre os caracteres especiais da máscara seguindo a sequencia.

Antes de criarmos e utilizarmos um *JFormattedTextField* devemos criar um objeto *MaskFormatter* e configurar uma máscara. Segue na **Listagem 4** um exemplo de uma máscara definida para CPF.

**Listagem 4**. Definindo uma mascara para um CPF.

MaskFormatter mascaraCpf = new MaskFormatter("###.###.###-##");

Após isso, configuramos a máscara no JFormattedTextField, conforme o código presente na **Listagem 5**.

**Listagem 5**. Configurando a mascara no objeto JFormattedTextField.

JFormattedTextField cpf = new JFormattedTextField(mascaraCpf);

Quando criamos um *MaskFormatter* podemos utilizar ao invés de “#”, outros caracteres, dependendo do tipo de restrição que desejamos implementar no *JFormattedTextField*. Esses caracteres são definidos abaixo:

* "**#**" indica que qualquer número poderá ser inserido (0-9);
* "**U**" indica que qualquer letra (a-z) poderá ser inserida. A máscara converterá letras minúsculas em maiúsculas;
* "**L**" indica qualquer letra (a-z) poderá ser inserida. A máscara converterá letras maiúsculas em minúsculas;
* "**?**" indica qualquer letra (a-z) poderá ser inserida. A máscara manterá a letra inserida;
* "**A**" indica qualquer letra ou numero (0-9 e a-z) poderá ser inserido;
* "**H**" indica qualquer caracter hexadecimal (0-9 a-f) poderá ser inserido;
* "**\***" indica qualquer coisa, incluindo caracteres especiais poderão ser inseridos.

Segue abaixo algumas máscaras prontas que podemos utilizar em nossos projetos:

* **Telefone Internacional**: "+##(##)####-####"
* **Telefone Nacional**: "(##)####-####"
* **CEP**: "##.###-###" ou "#####-###"
* **CPF**: "###.###.###-##"
* **Placa de automóveis**: "UUU-####"
* **CNPJ**: "##.###.###/####-##"
* **Título de eleitor**: "#########/##"
* **Data de nascimento**: "##/##/####"

Para exemplificar o uso do *JFormattedTextField* segue o exemplo presente na **Listagem 6,** onde criamos quatro campos com máscaras, são eles: CEP, Telefone, CPF, Data.

**Listagem 6**. Exemplo de código utilizando máscaras para CEP, telefone, CPF e Data.

package com.exemplo;

import java.awt.Container;

import java.text.ParseException;

import javax.swing.JFormattedTextField;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.text.MaskFormatter;

public class TestandoJFormattedTextField extends JFrame {

private static final long serialVersionUID = 1L;

public static void main(String[] args)

{

TestandoJFormattedTextField field = new TestandoJFormattedTextField();

field.testaJFormattedTextField();

}

private void testaJFormattedTextField() {

Container janela = getContentPane();

setLayout(null);

//Define os rótulos dos botões

JLabel labelCep = new JLabel("CEP: ");

JLabel labelTel = new JLabel("Telefone: ");

JLabel labelCpf = new JLabel("CPF: ");

JLabel labelData = new JLabel("Data: ");

labelCep.setBounds(50,40,100,20);

labelTel.setBounds(50,80,100,20);

labelCpf.setBounds(50,120,100,20);

labelData.setBounds(50,160,100,20);

//Define as máscaras

MaskFormatter mascaraCep = null;

MaskFormatter mascaraTel = null;

MaskFormatter mascaraCpf = null;

MaskFormatter mascaraData = null;

try{

mascaraCep = new MaskFormatter("#####-###");

mascaraTel = new MaskFormatter("(##)####-####");

mascaraCpf = new MaskFormatter("#########-##");

mascaraData = new MaskFormatter("##/##/####");

mascaraCep.setPlaceholderCharacter('\_');

mascaraTel.setPlaceholderCharacter('\_');

mascaraCpf.setPlaceholderCharacter('\_');

mascaraData.setPlaceholderCharacter('\_');

}

catch(ParseException excp) {

System.err.println("Erro na formatação: " + excp.getMessage());

System.exit(-1);

}

//Seta as máscaras nos objetos JFormattedTextField

JFormattedTextField jFormattedTextCep = new JFormattedTextField(mascaraCep);

JFormattedTextField jFormattedTextTel = new JFormattedTextField(mascaraTel);

JFormattedTextField jFormattedTextCpf = new JFormattedTextField(mascaraCpf);

JFormattedTextField jFormattedTextData = new JFormattedTextField(mascaraData);

jFormattedTextCep.setBounds(150,40,100,20);

jFormattedTextTel.setBounds(150,80,100,20);

jFormattedTextCpf.setBounds(150,120,100,20);

jFormattedTextData.setBounds(150,160,100,20);

//Adiciona os rótulos e os campos de textos com máscaras na tela

janela.add(labelCep);

janela.add(labelTel);

janela.add(labelCpf);

janela.add(labelData);

janela.add(jFormattedTextCep);

janela.add(jFormattedTextTel);

janela.add(jFormattedTextCpf);

janela.add(jFormattedTextData);

setSize(400, 250);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

}

}

Segue na **Figura 5** um exemplo de como ficou a tela codificada acima.

 Exemplo da tela
com os campos de textos e as máscaras configuradas 

**Figura 5**. Exemplo da tela com os campos de textos e as máscaras configuradas.

Se tentarmos entrar com letras onde apenas se aceita número veremos que os caracteres não serão aceitos no campo.

**Referências**

Documentação oficial do JTextField. Disponível em<http://docs.oracle.com/javase/6/docs/api/javax/swing/JTextField.html>

Documentação oficial do JFormattedTextField. Disponível em<http://docs.oracle.com/javase/8/docs/api/javax/swing/JFormattedTextField.html>