

Mascarando campos de um formulário usando Java para Desktop

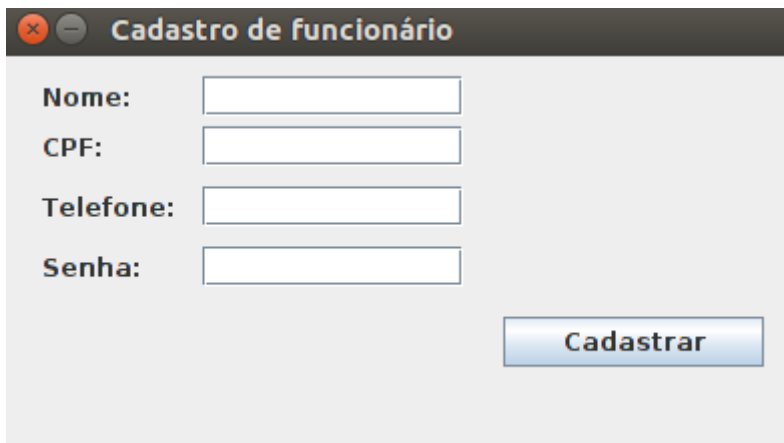
Fonte: <https://docs.oracle.com/javase/tutorial/uiswing/components/formattedtextfield.html>

Vamos supor que temos um sistema desktop que gerencia os funcionários de uma certa empresa. Para realizar tal tarefa, o sistema necessita que todos os usuários estejam cadastrados.

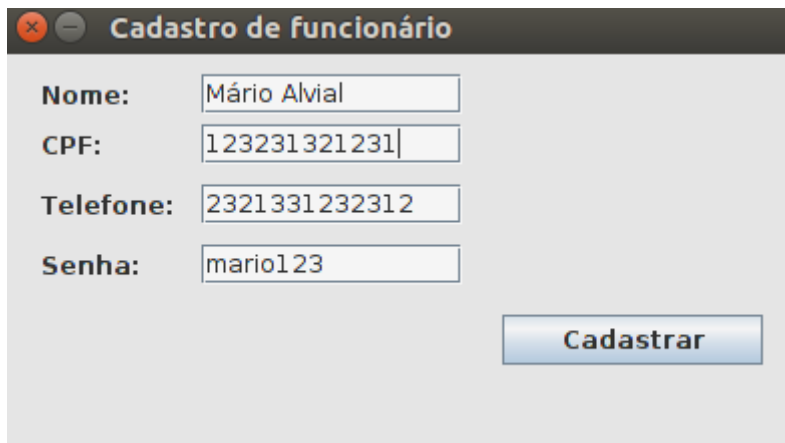
Temos nosso formulário de cadastro representado pelo seguinte código:

```
public class FormularioCadastro extends JFrame {  
  
    private JTextField nome = new JTextField();  
    private JTextField cpf = new JTextField();  
    private JTextField telefone = new JTextField();  
    private JTextField senha = new JTextField();  
  
    //resto do código  
}
```

Ao executarmos a aplicação, temos o seguinte resultado:

A imagem mostra uma janela de aplicativo com o título "Cadastro de funcionário". A janela possui um fundo cinza claro e uma barra de título escura com botões de controle padrão do Windows. No interior, há quatro rótulos de texto ("Nome:", "CPF:", "Telefone:", "Senha:") alinhados à esquerda, cada um seguido por um campo de entrada de texto branco com uma borda cinza. Abaixo dos campos, no canto inferior direito, há um botão de "Cadastro" com um gradiente de azul claro para escuro e uma borda cinza.

Mas perceba que este formulário não está legal, veja o que acontece quando inserimos dados nesses campos:



Note os campos de CPF, telefone e senha. Perceba que os campos não têm limite de caracteres. Além disso, a forma correta para o telefone seria algo desse jeito: (11) 99713-3837, para o CPF 893.920.408-57 e, também, não é nada legal deixar a senha explícita desse jeito. Como formatar tais dados?

JTextFormattedField

Precisamos colocar uma máscara em cima dos nossos campos para que, conforme os dados sejam inseridos, nossos campos já consigam formatá-los. Infelizmente a classe [JTextField](#) não dá o suporte para formatação dos dados, mas para isso existe outra classe.

A classe [JTextFormattedField](#) estende a classe `JTextField`. Essa classe adiciona suporte para mascarar os dados do seu campo. Com isso podemos limitar e formatar os dados inseridos pelo usuário.

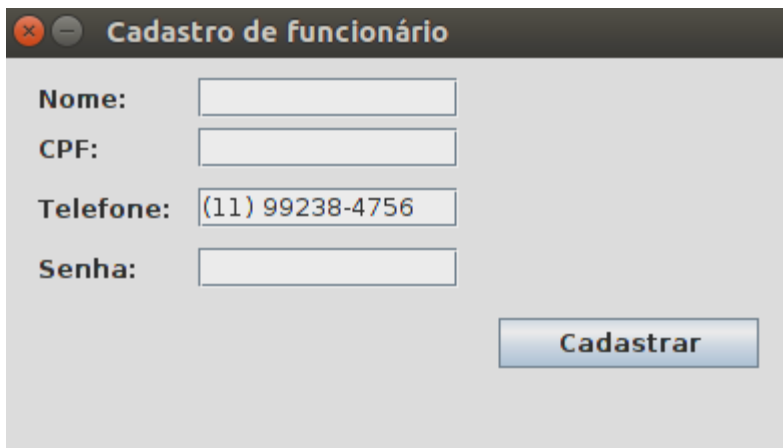
Agora que já sabemos o que precisamos usar, vamos por a mão na massa. Primeiro vamos colocar máscara no campo de telefone, para isso vamos alterar o atributo `telefone`:

```
private JTextField telefone = new JFormattedTextField(new  
MaskFormatter("(##) #####-####"));
```

Perceba que só foi necessário trocar o lado direito da declaração, pois como `JFormattedTextField` estende `JTextField` o mesmo não deixa de ser um `JTextField`.

Também podemos observar que passamos um objeto do tipo [MaskFormatter](#) e nele passamos a máscara que nosso campo terá. No caso, passamos "(##) #####-####", essa "#" significa que o campo só irá aceitar números.

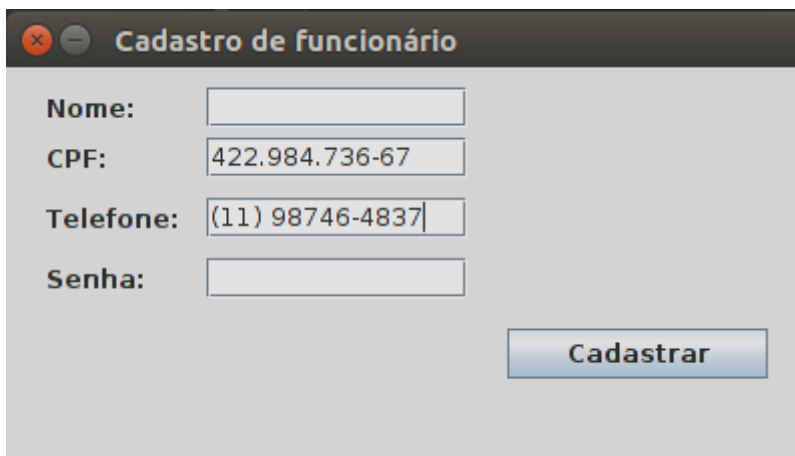
Se rodarmos nossa aplicação novamente veremos o resultado:



Boa, deu certo. Agora para o CPF, queremos, também, que só seja permitido dígitos e a formatação específica do CPF, certo? Bom, para isso vamos fazer a mesma coisa que fizemos com o telefone:

```
private JTextField cpf = new JFormattedTextField(new MaskFormatter("###.###.###-##"));
```

Executando o código:



E por último, a senha. Como encobrir os caracteres da senha com aquela bolinha preta?

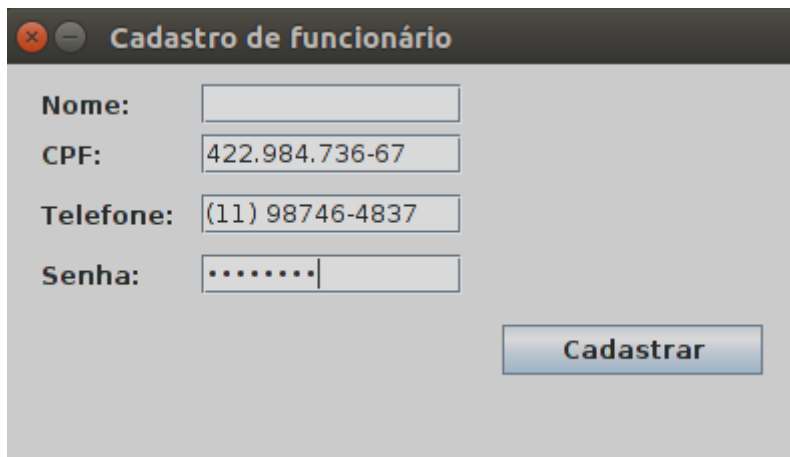
JPasswordField

Igual ao `JFormattedTextField` o [JPasswordField](#) também estende `JTextField`. Essa classe, por sua vez, é um componente que representa um campo para digitar senhas ou informações sigilosas, pois ela esconde o valor original digitado e, por padrão, converte para as bolinhas pretas que estamos habituados a ver em campos de senha.

Para adquirir o resultado esperado, vamos trocar o atributo `senha`. Dessa forma:

```
private JTextField senha = new JPasswordField();
```

Ao executar a aplicação temos:



Cadastro de funcionário

Nome:

CPF:

Telefone:

Senha:

Cadastrar

Pronto, todos os campos foram devidamente formatados!

Conclusão

Praticamente todos os formulários têm a necessidade de usar máscara para seus campos para auxiliar o usuário que está inserindo os dados, evitando erros e complicações no momento de preenchê-los.

Outra vantagem de fazer isso é que você vai conseguir ter um conhecimento melhor de como os dados vão chegar no seu servidor, logo, facilita na hora de tratar os dados.