



# Desenvolvimento de Sistemas

---

## Introdução

A validação é uma técnica que viabiliza o controle da entrada de dados vindos de formulários, impedindo, por exemplo, que um usuário digite caracteres em campos nos quais devem conter apenas números, ou não digite o “@” em campos de *e-mail*, digite obrigatoriamente letras maiúsculas e minúsculas em uma senha, controle a quantidade de caracteres em um campo, ou seja, respeite os tipos de valores que são esperados pelo sistema.

Esse conhecimento é muito importante para os programadores, pois, quando utilizado corretamente, permite que seja mantida a integridade, a consistência e a segurança das informações que entram em um sistema, ao mesmo tempo que é essencial para o bom funcionamento dele. Um sistema que não tenha validação poderá enfrentar problemas, pois os dados muitas vezes estarão incorretos ou incompletos, e também ficar sujeito a ataques à sua segurança em virtude de comandos maliciosos inseridos onde deveriam constar informações, como, por exemplo, CPF, data de nascimento etc.

Como você estará aprendendo a criar interfaces em Java com Java Swing, é muito importante o conhecimento das validações e o uso de expressões regulares que você aprenderá no decorrer do conteúdo, pois a validação é uma das técnicas que influenciam a interação e que permitem que o uso de uma interface seja agradável para o usuário, buscando eliminar os possíveis erros que se teria com a inserção de dados diferentes dos esperados pelo sistema.

## Formatação de dados

É muito importante para um sistema que os dados de um formulário sejam validados, formatados e convertidos caso isso seja necessário.

O primeiro passo é aprender a receber os dados corretamente, filtrar o que é e o que não é permitido e nunca esquecer de mandar uma mensagem final para o usuário informando se os dados foram cadastrados corretamente, ou, em caso de erro, explicando de modo direto e simples qual erro ocorreu e o que o usuário poderá fazer para resolvê-lo.

Considere o exemplo inicial em um formulário simples feito no NetBeans 13, no qual se está somente recebendo a idade de uma pessoa e depois disso, clica-se no botão **Enviar**. O resultado será apresentado na tela e, com esse formulário inicial, será possível começar a analisar a diferença entre um formulário que apresenta formatação de dados e um formulário que não apresenta tal técnica.

O formulário conterà apenas uma entrada de dados para idade e, mesmo com um exemplo bem simples, você conseguirá notar a importância da validação dos formulários. Analise a imagem a seguir:

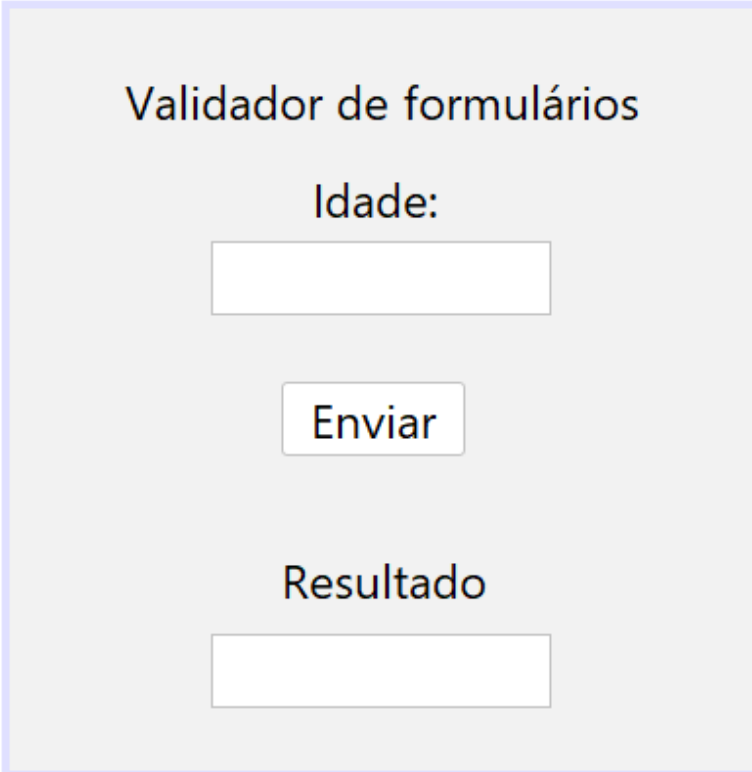


Figura 1 – Formulário de validação de um campo de texto vazio

Fonte: NetBeans (2022)

Nesse formulário, utiliza-se o seguinte:

- ◆ *Label (JLabel)* para os textos “validador de formulário”, “Idade:” e “Resultado”
- ◆ *Text Field (JTextField)* para inserir a idade e mostrar o resultado
- ◆ *Button (JButton)* ou “botão”, para ser clicado e apresentar o resultado

Porém, com esse pequeno exemplo, foi possível demonstrar que, mesmo em apenas um campo de formulário, já é importante validar a entrada de dados para que o seu funcionamento seja consistente e a resposta ao usuário seja correta.

Neste primeiro momento, você verá um exemplo sem validação. O código será todo inserido dentro do evento e depois se clica no botão:

Clique com o botão direito em **Enviar**.



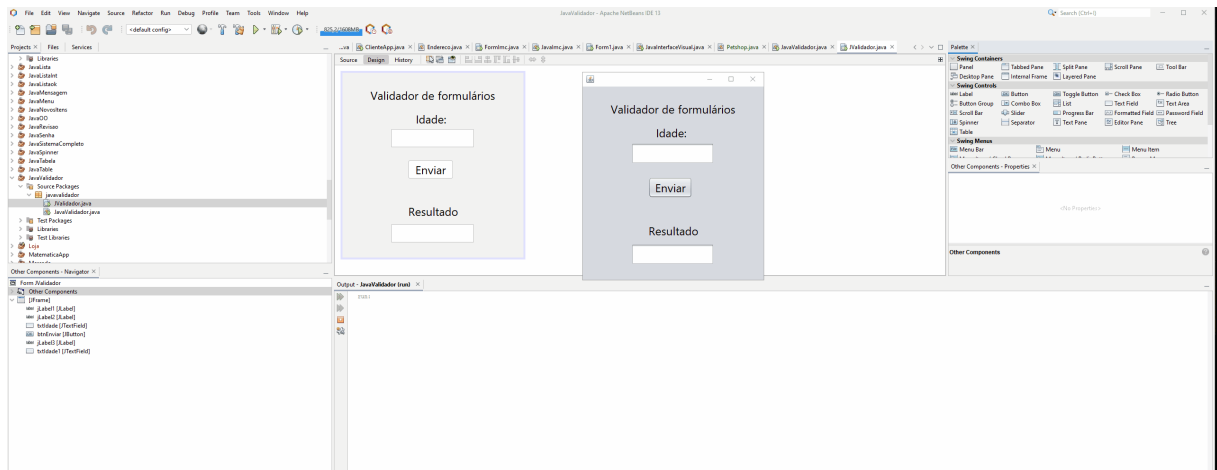
Escolha a opção **Events**, depois a opção **Action** e, por fim, clique em **ActionPerformed**.

Depois desses passos, todo o código a seguir pode ser inserido no programa:

```
private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt){  
    String strIdade = txtIdade.getText();  
    int idade = Integer.parseInt(strIdade);  
    String strResultado = String.valueOf(idade);  
    txtResultado.setText(strResultado);  
}
```

Como um campo de texto foi utilizado para inserir a idade, é preciso criar uma variável “strIdade” para receber esse valor. Depois, utiliza-se um comando interno de Java chamado **Integer.parseInt()** para converter o valor do tipo texto para inteiro, e aqui é o momento em que o erro ocorre caso a entrada de dados não seja um valor válido. Esse erro será mostrado em seguida. Por fim, converte-se a “idade” em *string* por meio do comando **String.valueOf()**, pois precisa-se desse valor como *string* para mostrar na caixa de texto “txtResultado”, que é o nome da variável dado à última caixa de texto do formulário.

Observe o GIF (*graphics interchange format*) criado a partir do processo no NetBeans (2022) para demonstrar o que acontece quando se coloca um número inteiro – que é o esperado pelo programa – e o que acontece quando se coloca qualquer outro tipo de valor, como texto, número com vírgula, data etc.



Percebe-se pela demonstração que existe também um primeiro que deve ser considerado quando se valida um formulário, é o caso de um campo não ser preenchido. A primeira coisa a ser feita é importar a biblioteca, mostrada no código a seguir, para permitir a inserção de mensagens que serão utilizadas quando o campo estiver vazio.

```
import javax.swing.JOptionPane;
```

Depois de importar essa biblioteca, podem-se utilizar caixas de mensagem em qualquer tipo de validação de formulários. Observe então a solução do primeiro problema, que ocorre quando nada é digitado no campo idade. Inclua o código a seguir no corpo do método **btnEnviarActionPerformed()**.

```
if(txtIdade.getText().isEmpty()) {
    JOptionPane.showMessageDialog(null, "O campo idade precisa ser preenchido: ");
}
else {
    String strIdade = txtIdade.getText();
    int idade = Integer.parseInt(strIdade);
    String strResultado = String.valueOf(idade);
    txtResultado.setText(strResultado);
}
```

Analise agora este comando:

```
txtIdade.getText().isEmpty()
```

Note que foi possível testar se o campo está vazio. Se estiver, utilize o seguinte comando:

```
JOptionPane.showMessageDialog(null, "O campo idade precisa ser preenchido: ");
```

Esse comando deve ser utilizado para mostrar a mensagem que indica que o campo **Idade** precisa ser preenchido. Caso ele já esteja preenchido, o código continua como no exemplo, sem validação.

Outra situação que deve ser considerada a respeito do campo **Idade** é ajustá-lo para que ele não aceite números negativos nem iguais a zero. Para isso, faça uma pequena modificação do código, como mostrado a seguir:

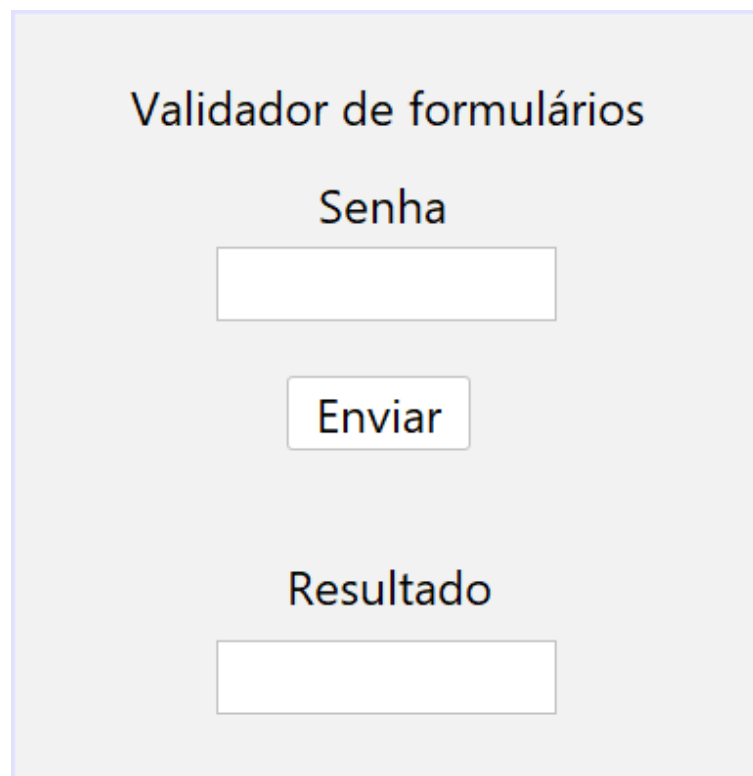
```
private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {  
    if(txtIdade.getText().isEmpty())  
    {  
        JOptionPane.showMessageDialog(null, "O campo idade precisa ser preenchido e  
precisa ter um valor maior do que 0 ");  
    }  
    else if(Integer.parseInt(txtIdade.getText())<0){  
        JOptionPane.showMessageDialog(null, "O campo idade precisa ser preenchido e  
precisa ter um valor maior do que 0 ");  
    }  
    else {  
        String strIdade = txtIdade.getText();  
        int idade = Integer.parseInt(strIdade);  
        String strResultado = String.valueOf(idade);  
        txtResultado.setText(strResultado);  
    }  
}
```

Perceba que foi possível testar se um número não é negativo ou igual a zero por meio deste comando:

```
Integer.parseInt(txtIdade.getText())<0)
```

Com esse código, pode-se agora testar ao mesmo tempo se o campo foi preenchido e se ele é maior do que zero.

Agora, o primeiro formulário será modificado para mostrar um segundo tipo de validação que você precisa conhecer para controlar a quantidade de caracteres inseridos em um campo de texto. Essa validação é muito utilizada em senhas e em alguns campos de bancos de dados:



Validador de formulários

Senha

Enviar

Resultado

Figura 2 – Formulário de validação de tamanho para campo de texto

Fonte: Adaptado de NetBeans 13 (2022)

Com o código a seguir, você pode testar se o campo **Senha** tem no mínimo seis caracteres. Caso não tenha, uma mensagem de erro deve ser apresentada.

```
private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {  
    if(txtSenha.getText().isEmpty() || txtSenha.getText().length()<6)  
    {  
        JOptionPane.showMessageDialog(null, "O campo idade precisa ser preenchido e  
ter no mínimo 6 caracteres: ");  
    }  
    else {  
        String strSenha = txtSenha.getText();  
        int Senha = Integer.parseInt(strSenha);  
        String strResultado = String.valueOf(Senha);  
        txtResultado.setText(strResultado);  
    }  
}
```

Note que, no primeiro **If** do código, existe um comando novo que testa se a senha não tem menos do que seis caracteres. Este comando é:

```
txtSenha.getText().length()<6
```

Depois dessas validações iniciais, conheça um conceito importante na validação de dados, as “expressões regulares”.

Crie uma interface com um campo de texto para inserção de um nome e permita que o usuário insira no máximo 20 caracteres, caso contrário, o sistema deve apresentar um erro.



# Funções da linguagem e expressões regulares

Uma expressão regular ou Regex (*regular expression*) é um texto com uma formatação especial que mostra um padrão de pesquisa e substituição de textos, ou seja, são padrões utilizados para identificar determinadas combinações ou cadeias de caracteres em uma variável do tipo *string*. Uma expressão regular tem três usos principais: busca de elementos em um texto, validação de elementos e substituição de formatos.

Por exemplo, você pode procurar a data 20/10/022 em um texto e testar se ela está no formato correto com uso do caractere barra `"/"`. Caso seja informado um texto em formato incorreto, como 20-10-2022 com hifens, por exemplo, é possível detectar e ainda substituir esse valor pelo formato correto a partir de expressões regulares. Além disso, podem-se isolar partes da data como dia, mês ou ano.

Antes de testar esse tipo de validação – que facilita muito, pois permite que sejam utilizados menos códigos de condição **If**, tornando o código menor e mais limpo –, é necessário entender como funcionam algumas expressões regulares.

Primeiro, você precisa compreender o conceito de **metamarcadores**, que são caracteres especiais de delimitam o que pode e o que não pode aparecer em um campo. Eles indicam, quando é necessário, por exemplo, a ocorrência de números ou textos, diferenciam maiúsculas e minúsculas e podem exigir ainda a ocorrência de caracteres especiais, como se pode ver nos seguintes exemplos:

**A**

[0-9] ou //d: busca qualquer número de 0 a 9

**B**

[^0-9] ou //D: busca qualquer caractere que não seja número, pois antes do 0-9 consta o símbolo “^”.

**C**

[a-z]: busca qualquer letra de “a” a “z” que seja minúscula

**D**

[A-Z]: busca qualquer letra de “a” a “z” que seja maiúscula

**E**

[a-zA-Z]: busca qualquer letra, independentemente se maiúscula ou minúscula

**F**

//w: busca qualquer caractere de letras e números

**G**

//W: busca qualquer caractere que não seja letras e números

Depois desses conceitos, você tem que entender o uso dos **quantificadores**, que são caracteres que permitem informar quantas vezes um metamarcador pode ser repetido, como demonstrado no exemplo a seguir.

Expressão	Exemplo	Efeito
{n}	[0-9]{2}	Busca a ocorrência de dois números em uma expressão
{n,}	[0-9]{2,}	Busca a ocorrência de pelo menos dois números em uma expressão
{2,5}	[0-9]{2,5}	Busca a ocorrência de pelo menos dois números e no máximo cinco em uma expressão

Quadro 1 – Quantificadores  
Fonte: Senac EAD (2022)

Para finalizar, você deve compreender os **metacaracteres de fronteira**, o padrão de início e fim de uma determinada variável do tipo *string*. Veja o exemplo:



Metacaractere	Efeito
^	Padrão de início da <i>string</i>
\$	Padrão de fim da <i>string</i>
	Padrão para condição lógica “OU”
.*	Padrão que indica “TUDO”

Quadro 2 – Metacaracteres  
Fonte: Senac EAD (2022)

Para testar os metacaracteres de fronteira, linhas de código em Java serão utilizadas para mostrar na prática como funciona o seu uso:

A seguir, utilize o método **matches()** da classe *string* de Java. Esse método é aplicado sobre uma variável ou valor *string*, recebe por parâmetro uma expressão regular e valida o valor *string* a partir dessa expressão, retornando “verdadeiro”, se houver correspondência com a expressão, ou “falso”, caso contrário.

```
boolean palavra = "Java2022".matches("^Java$");
```

Nesse caso, o retorno será “falso” (*false*), porque a pesquisa está procurando padrões que contenham exatamente a palavra “Java”. Note que antes do “J” está o símbolo “^”, de início de *string*, e no fim da palavra está o símbolo “\$”, de final de *string*, portanto qualquer palavra diferente de “Java”, retornaria o termo “falso” (*false*). Porém, se sua busca fosse por uma palavra que começasse por “Java”, mas que pudesse ter qualquer valor, após isso você teria uma expressão da seguinte forma:

```
boolean palavra = "Java2022".matches("^Java.*");
```

Agora, o “**\***” representa que, após encontrar a palavar “Java”, será possível ter quaisquer caracteres. Por exemplo, palavras como Java2022 (que é a procurada neste teste), Java21, JavaTeste, Java2021 etc., todas retornariam o valor verdadeiro (*true*).

```
boolean time = "Brasill".matches("Brasil|Alemanha");
```

Nesse caso, o retorno seria verdadeiro (*true*), pois está procurando “Brasil” ou “Alemanha”, e o termo que se tem é “Brasil”.

Para simplificar, considere alguns exemplos práticos de validação com expressões regulares para depois testá-los nos formulários do NetBeans.

## A

Considere uma validação de hora em que se precisa ter um formato do tipo “10:45”.

`[0-9]{2}:[0-9]{2}` (dois números de 0 a 9, depois “:”, mais dois números de 0 a 9)

## B

No exemplo de uma validação de data que precisa estar no formato “21/08/2022”, a expressão regular será a seguinte:

`[0-9]{2}/[0-9]{2}/[0-9]{4}` (dois números de 0 a 9, depois “/”, mais dois números de 0 a 9, “/” e, por fim, quatro números de 0 a 9)

## C

Pense agora na validação de uma máscara de CPF, em que se precisa ter um número no formato “001.245.198-45”. Nesse caso, a expressão será:

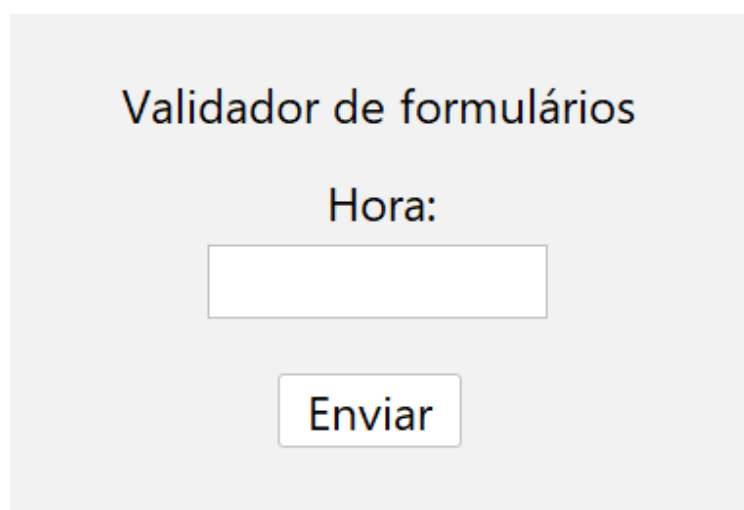
`[0-9]{3}[.][0-9]{3}[.][0-9]{3}[-][0-9]{2}` (três números de 0 a 9, depois “.”, mais três números de 0 a 9, novamente “.”, mais três números de 0 a 9, o símbolo “-” e por fim dois números de 0 a 9)

## D

Considere agora a validação de uma máscara de *e-mail*, que é certamente uma das validações mais importantes na utilização dos formulários. Nela, é preciso ter uma máscara no formato “contato@senacead.com.br”, e a expressão nesse caso será:

`\\w+@\\w+\\.\\w{2,3}\\.\\w{2,3}` (o “\\w” indica qualquer caractere, podendo incluir tanto letra quanto número. Espera-se com essa máscara o seguinte resultado: qualquer informação com número ou texto seguida do arroba “@”, depois novamente qualquer informação seguida de “.”, depois qualquer informação com dois ou três caracteres seguida de “.” e, por fim, mais dois ou três caracteres, formando o formato de padrão que se conhece nos *e-mails*.)

Agora, para colocar em prática a validação de formulários reais com o uso das expressões regulares, utilize o formulário mostrado na seguinte imagem:



Validador de formulários

Hora:

Figura 3 – Formulário validador de hora com Regex  
Fonte: NetBeans (2022)

Com o código apresentado em seguida, pode-se testar se o campo **Hora** tem ou não o formato esperado, que é “hh:mm” (dois números para hora e dois números para minutos, separados pelo sinal de “:”). Caso seja uma condição verdadeira, é enviada uma mensagem de que o formato é válido; caso seja uma condição falsa, é enviada uma mensagem de que o formato é inválido. Para esse teste, utilize a expressão regular **[0-9]{2}:[0-9]{2};**.

```
private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {  
  
    String strHora = txtHora.getText();  
    if(strHora.isEmpty())  
    {  
        JOptionPane.showMessageDialog(null, "O campo data precisa ser preenchido: ");  
    }  
    else {  
        String strTeste = txtHora.getText();  
        boolean palavra = strTeste.matches("[0-9]{2}:[0-9]{2}");  
  
        if(palavra==true) {  
            JOptionPane.showMessageDialog(null, "você inseriu uma hora no formato válido ");  
        }  
        else{  
            JOptionPane.showMessageDialog(null, "Hora precisa ser no formato hh:mm ");  
        }  
    }  
}
```

Esse formulário é uma atividade muito interessante para se testar se o usuário digitou o padrão correto de inserção das horas. Para esse tipo de validação mais complexa, é necessário o uso das expressões regulares. Entenda agora as etapas mais importantes desse código.

As primeiras duas linhas de código que precisam ser analisadas são:

```
String strTeste = txtHora.getText();  
boolean palavra = strTeste.matches("[0-9]{2}:[0-9]{2}");
```

Na primeira linha, cria-se uma variável *string* que receberá o código digitado no campo de texto nomeado de variável “txtHora”. Até aqui não há nada diferente do que já foi feito, porém, a partir da segunda linha, cria-se uma variável lógica (*boolean*), que testará se o valor inserido respeita a expressão regular “[0-9]{2}:[0-9]{2}”. Lembre-se que a função do Java utilizada para fazer essa comparação é chamada de **matches**, e ela é umas das funções mais importantes que você precisa conhecer na parte de validação, pois permite comparar uma *string* recebida pelo formulário com uma expressão regular que foi criada. Isso significa que só será retornado um valor verdadeiro (*true*) se o que está sendo inserido no campo de texto for exatamente o pedido dentro da função **matches**; caso contrário a, função retornará o resultado falso (*false*).

Depois do uso da função **matches**, é possível ter o resultado verdadeiro ou falso, e com ele retornar ao usuário a mensagem que indica se a data estava correta ou não, conforme a condição mostrada a seguir:

```
if(palavra==true)
```

Caso seja uma condição verdadeira (*true*), será mostrada a mensagem: “Você inseriu uma hora no formato válido”; caso seja uma condição falsa, a mensagem será: “Hora precisa ser no formato hh:mm”.

Com essa mesma técnica, é possível fazer um validador para **Telefone** e **CPF**. Nesse caso, pode-se utilizar um mesmo formulário, testando os dois ao mesmo tempo, para demonstrar que, na validação, é possível ter tantos campos de informações quantos forem necessários para o sistema.

Para isso, pode-se utilizar um formulário igual ao exemplo a seguir:



Validador de formulários

Data:

CPF:

Enviar

Figura 4 – Formulário validador de **Data** e **CPF** com Regex

Fonte: NetBeans (2022)

Com o código apresentado em seguida, você pode testar se a data contém ou não o formato esperado, que é “dd/mm/aaaa” (dois números para dia, dois números para mês, quatro números para ano, separados pelo sinal de “/”). Caso seja uma condição verdadeira, haverá a mensagem de que o formato é válido; caso a condição seja falsa, haverá a mensagem de que o formato é inválido. Para esse teste, utilize a expressão (“[0-9]{2}/[0-9]{2}/[0-9]{4}”).

Da mesma forma, você pode testar o CPF, que deve ter três sequências de três números separados por “.” e depois dois números após o sinal “-”. Para isso, utilize a expressão (“[0-9]{3}.[0-9]{3}.[0-9]{3}-[0-9]{2}”).



```
private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {  
  
    String strData = txtData.getText();  
    String strCpf = txtCpf.getText();  
    if(strData.isEmpty() || strCpf.isEmpty())  
    {  
        JOptionPane.showMessageDialog(null, "Todos os campos precisam ser preenchid  
os. ");  
    }  
    else{  
        String strTeste = txtData.getText();  
        boolean palavra = strTeste.matches("[0-9]{2}/[0-9]{2}/[0-9]{4}");  
        String strTeste2 = txtCpf.getText();  
        boolean palavra2 = strTeste2.matches("[0-9]{3}[.][0-9]{3}[.][0-9]{3}-[0-9]{2}");  
  
        if(palavra==true && palavra2==true) {  
            JOptionPane.showMessageDialog(null, "você inseriu a data e o cpf nos fo  
rmatos válidos ");  
        }  
        else if(palavra==false && palavra2==true){  
            JOptionPane.showMessageDialog(null, "Data precisa ser no formato dd/mm/  
aaaa ");  
        }  
        else if(palavra==true && palavra2==false){  
            JOptionPane.showMessageDialog(null, "Cpf precisa se no formato xxx.xxx.  
xxx-xx ");  
        }  
        else{  
            JOptionPane.showMessageDialog(null, "Data precisa ser no formato dd/mm/  
aaaa e Cpf no formato xxx.xxx.xxx-xx ");  
        }  
    }  
}
```

Nessa nova validação, será necessário usar um comando **if** com duas condições para testar se tanto a data quanto o CPF estão preenchidos. Para testar as duas aos mesmo tempo, pode-se utilizar o símbolo “||” (ou), que retorna um valor verdadeiro, se pelo menos uma condição for verdadeira. A seguir, confira a linha de código considerada nessa situação.

```
if(strData.isEmpty() || strCpf.isEmpty())
```

Depois do teste, se os campos foram preenchidos, será preciso criar as máscaras com as expressões regulares para testar se o formato da data e do CPF foram preenchidos corretamente. Para isso, você recebe os valores de data e CPF e cria uma variável lógica (*boolean*) para esse teste, como nos códigos a seguir:

```
String strTeste = txtData.getText();
boolean palavra = strTeste.matches("[0-9]{2}/[0-9]{2}/[0-9]{4}");
String strTeste2 = txtCpf.getText();
boolean palavra2 = strTeste2.matches("[0-9]{3}[.][0-9]{3}[.][0-9]{3}[-][0-9]{2}");
```

Note que foram criadas duas variáveis chamadas “palavra” e “palavra2”, que, por meio das expressões regulares, testarão, com o auxílio da função **matches**, se o texto digitado pelo usuário respeita o formato exigido pelo sistema.

Após esses testes, será utilizado um encadeamento dos condicionais *If-else* para analisar cada uma das quatro situações que podem acontecer, as quais são as seguintes:

- ◆ Se as variáveis “palavra” e “palavra2” retornarem valores verdadeiros (*true*), significa que os formatos das expressões regulares foram respeitados e uma mensagem sobre isso retornará ao usuário, como mostrado neste comando:

```
if(palavra==true && palavra2==true) {
    JOptionPane.showMessageDialog(null, "você inseriu a data e o cpf nos formatos válidos ");
}
```

- ◆ Se a variável “palavra” retornar um valor falso (*false*) e a variável “palavra2” retornar um valor verdadeiro (*true*), significa que somente o formato da data não foi respeitado e uma mensagem sobre isso retornará ao usuário, como mostrado neste comando:

```
else if(palavra==false && palavra2==true){  
    JOptionPane.showMessageDialog(null, "Data precisa ser no formato dd/mm/aaaa ");  
}
```

- ◆ Se a variável “palavra” retornar um valor verdadeiro (*true*) e a variável “palavra2” retornar um valor falso (*false*), significa que somente o formato do CPF não foi respeitado e uma mensagem sobre isso retornará ao usuário, como mostrado neste comando:

```
else if(palavra==true && palavra2==false){  
    JOptionPane.showMessageDialog(null, "Cpf precisa se no formato xxx.xxx.xxx-xx  
");  
}
```

- ◆ Em qualquer outra situação se utiliza o comando **else** (senão), o que significa que os dois formatos não foram respeitados e uma mensagem sobre isso retornará ao usuário, como mostrado neste comando:

```
else{  
    JOptionPane.showMessageDialog(null, "Data precisa ser no formato dd/mm/aaaa e C  
pf no formato xxx.xxx.xxx-xx ");  
}
```

Crie uma interface com um campo de texto para inserção de data no formato “10/05/22 (dd/mm/aa)” e utilizando expressões regulares. O formulário deve apresentar um erro, caso a data seja inserida em qualquer outro formato.

Confira agora outro exemplo. Crie um novo projeto ou novo **JFrame** em um projeto existente e, no editor visual, monte uma tela como a da figura a seguir.

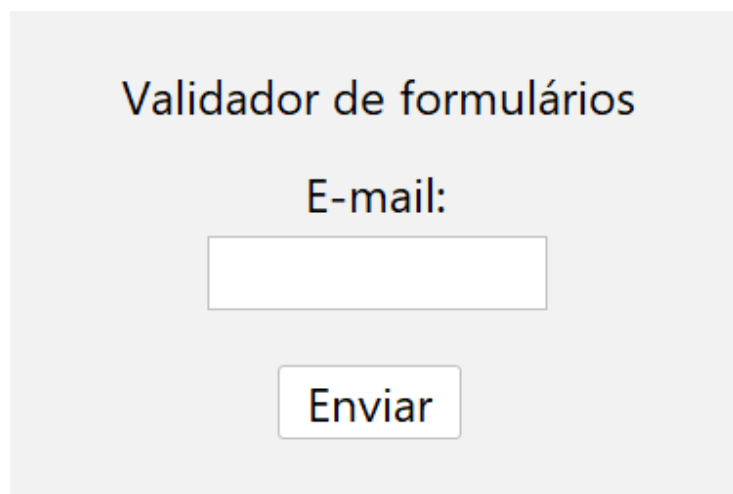
A imagem mostra uma interface gráfica de usuário (GUI) para um validador de e-mail. O título da janela é "Validador de formulários". Abaixo do título, há o rótulo "E-mail:" seguido por um campo de entrada de texto retangular. Abaixo do campo de entrada, há um botão retangular com o texto "Enviar". A interface é simples e funcional, com um fundo cinza claro.

Figura 5 – Formulário validador de *e-mail* com Regex

Fonte: NetBeans 13 (2022)

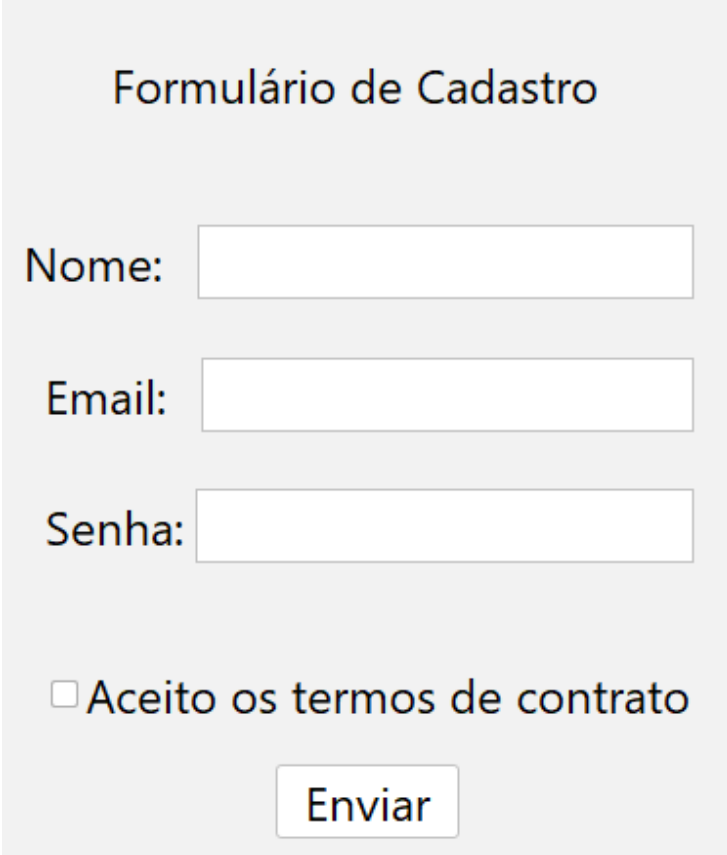
Com o código a seguir, é possível testar se o *e-mail* contém ou não o formato esperado, que é "nome@dominio.com.br". Para esse teste, utilize a expressão regular `\\w+@\\w+\\.\\w{2,3}\\\\.\\w{2,3}`.

```
private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {  
  
    String strHora = txtEmail.getText();  
    if(strHora.isEmpty())  
    {  
        JOptionPane.showMessageDialog(null, "O campo e-mail precisa ser preenchido");  
    }  
    else{  
        String strTeste = txtEmail.getText();  
        boolean palavra = strTeste.matches("\\w+@\\w+\\.\\w{2,3}\\.\\w{2,3}");  
  
        if(palavra==true) {  
            JOptionPane.showMessageDialog(null, "você inseriu um e-mail no formato válido ");  
        }  
        else{  
            JOptionPane.showMessageDialog(null, "E-mail precisa ser no formato nome@dominio.com.br ");  
        }  
    }  
  
}
```

Note que, com as expressões regulares, os códigos ficam bem semelhantes, facilitando muito o poder de validação do programador. Esse último código que foi criado é extremamente semelhante ao já criado para validação de hora, sendo que a expressão regular é diferente e serve para validação de *e-mail*. Esse código é muito importante para você observar que uma pequena diferença pode modificar totalmente o poder de validação de um formulário. Agora você tem uma estrutura que pode ajudá-lo sempre que for necessário validar campos de texto em Java, ou em qualquer outra linguagem, desde que você adapte os comandos utilizados.

## Exemplo final

Para finalizar com um entendimento amplo das validações e funções da linguagem Java nos tratamentos de formulários, você criará um exemplo de validação de um formulário de cadastro com três campos de texto, para inserção de nome, *e-mail* e senha, um campo de *checkbox*, para confirmar se a pessoa leu as regras de cadastro e aceita os termos, e um botão de enviar, como na imagem a seguir:



Formulário de Cadastro

Nome:

Email:

Senha:

☐ Aceito os termos de contrato

Figura 6 – Formulário de cadastro

Fonte: NetBeans (2022)

```
private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {  
  
    String strNome = txtNome.getText();  
    String strEmail = txtEmail.getText();  
    String strSenha = txtSenha.getText();  
  
    if(strNome.isEmpty() || strEmail.isEmpty() || strSenha.isEmpty())  
    {  
        JOptionPane.showMessageDialog(null, "Todos os campos devem ser preenchidos."  
);  
    }  
    else  
    {  
        if (cb_contrato.isSelected()){  
  
            String strTeste = txtEmail.getText();  
            boolean palavra = strTeste.matches("\\w+@\\w+\\.\\w{2,3}|\\w+@\\w+  
\\.\\w{2,3}.\\w{2,3}");  
            if(palavra==true && txtSenha.getText().length()>6) {  
                JOptionPane.showMessageDialog(null, "Cadastro realizado com suc  
cesso ");  
            }  
            else if(palavra==true && txtSenha.getText().length()<6){  
                JOptionPane.showMessageDialog(null, "Senha deve ter no mínimo 6  
caracteres ");  
            }  
            else if(palavra==false && txtSenha.getText().length()>6){  
                JOptionPane.showMessageDialog(null, "Email deve ser no formato  
nome@dominio.com ou nome@dominio.com.br");  
            }  
            else{  
                JOptionPane.showMessageDialog(null, "Email deve ser no formato  
nome@dominio.com.br e a senha deve ter no mínimo 6 caracteres ");  
            }  
        }  
        else  
        {  
            JOptionPane.showMessageDialog(null, "É obrigatório aceitar os termos de  
contrato para cadastro ");  
        }  
    }  
}
```

Nesse exemplo final, há uma visão mais completa da validação de um formulário de cadastro, em que o primeiro passo é testar se todos os campos foram preenchidos por meio do comando:

```
if(strNome.isEmpty() || strEmail.isEmpty() || strSenha.isEmpty())
```

Mas agora, nesse formulário, há também um campo diferente, não utilizado anteriormente, portanto, logo após testar se os campos anteriores foram preenchidos, deve-se testar se o campo novo foi marcado, pois esse é um campo obrigatório e o envio do cadastro só pode ser aceito após a validação de sua marcação com o seguinte comando:

```
if (cb_contrato.isSelected())
```

Esse comando testa se o *checkbox* chamado **cb\_contrato** foi selecionado. Caso tenha sido, o sistema segue captando as informações; caso negativo, ele entra no comando **else** e mostra a seguinte mensagem:

```
else
{
    JOptionPane.showMessageDialog(null, "É obrigatório aceitar os termos de contrato para cadastro ");
}
```

Dentro do **if** estão alguns comandos que precisam ser testados com mais atenção, um deles é a expressão retangular que serve tanto para padrões de **e-mail** do tipo nome@dominio.com quanto para padrões do tipo nome@dominio.com.br, como você pode ver a seguir:



```
String strTeste = txtEmail.getText();  
boolean palavra = strTeste.matches("\\w+@\\w+\\.\\w{2,3}|\\w+@\\w+\\.\\w{2,3}.\\w{2,3}");
```

Observe que no exemplo do *e-mail* anteriormente visto, testava-se apenas o formato “.com” e agora se testa também o “.com.br”, deixando ainda mais completa a validação.

Para que este formulário fique ainda mais detalhado, deve-se observar cada teste após a máscara de expressão regular de validação do *e-mail*:

```
if(palavra==true && txtSenha.getText().length()>=6) {  
    JOptionPane.showMessageDialog(null, "Cadastro realizado com sucesso ");  
}
```

Aqui, testou-se se o campo de *e-mail* obedece ao formato da expressão regular de validação e também se o campo de senha tem seis ou mais letras, se ambas resultarem em “verdadeiro” (*true*), aparecerá a mensagem de “cadastro realizado com sucesso”, e os testes condicionais continuam a seguir:

```
else if(palavra==true && txtSenha.getText().length()<6){  
    JOptionPane.showMessageDialog(null, "Senha deve ter no mínimo 6 caracteres ");  
}
```

Nesse teste, se o formato do *e-mail* estiver verdadeiro (*true*), mas a senha tiver menos que seis caracteres, aparecerá a mensagem “Senha deve ter no mínimo 6 caracteres”.

```
else if(palavra==false && txtSenha.getText().length()>=6){  
    JOptionPane.showMessageDialog(null, "Email deve ser no formato nome@dominio.com  
ou nome@dominio.com.br");  
}
```

Nesse teste, se o formato do *e-mail* resultar em “falso” (*false*), mas a senha estiver com seis ou mais caracteres, aparecerá a mensagem “E-mail deve ser no formato nome@dominio.com ou nome@dominio.com.br”.

```
else{  
    JOptionPane.showMessageDialog(null, "Email deve ser no formato nome@dominio.co  
m.br e a senha deve ter no mínimo 6 caracteres ");  
}
```

Para finalizarem os testes, se ambos resultarem em falso (*false*), a mensagem será a seguinte: “E-mail deve ser no formato nome@dominio.com.br e a senha deve ter no mínimo 6 caracteres”.

## Encerramento

Após a leitura do conteúdo, você pode perceber a importância da validação de formulários. Ela permite que as informações fiquem íntegras e seguras e evita os principais erros que acontecem na comunicação de dados em um *software*, pois limita o que pode ser recebido ao tipo de dado esperado e identifica que o *software* está pronto para ser manipulado.

Neste material, você aprendeu a importância das expressões regulares que auxiliam muitos nas validações de dados com formatos específicos, como datas, horas, CPF etc., e facilitam qualquer tipo de validação, por meio de padrões que agilizam e aumentam a segurança no tratamento das informações.

Além disso, você pôde observar a ligação entre a criação das interfaces com Java Swing e o tratamento das informações que ela recebe, pois um bom *designer* de interação não se preocupa apenas com o visual de sua interface, mas com as sensações que ela transmite ao usuário, portanto, qualquer tipo de erro deve ser evitado e, caso ocorra, ele deve ser identificado e também deve ser mostrada clara e objetivamente uma possível solução.