

02

Como validar um CPF?

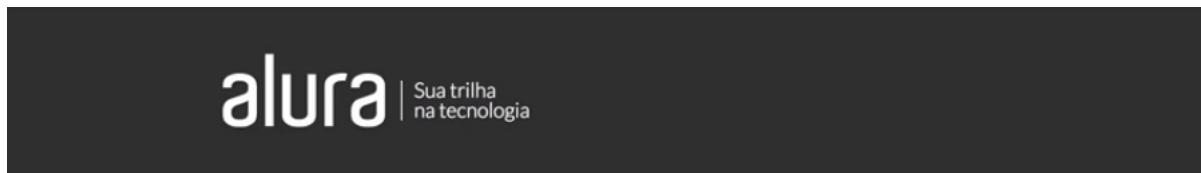
Transcrição

Temos aqui dois CPFs, e precisamos saber se são válidos ou não.

86288366757

98745366797

Vou copiar o primeiro CPF e colocá-lo em um site com formulário que exija CPF, por exemplo, o da Alura.



Informações da sua compra

1. Suas informações

Nome completo

Email

CPF

Telefone com DDD

2. Cartão

Powered by PayPal



Número do cartão

Vencimento

MM

AA

Cód Seg

3 dígitos

Selecionar parcelas para esta compra

Suas informações serão coletadas de acordo com a Política de Privacidade do PayPal.

Eu quero receber informações importantes, ofertas

Esses sites validam os campos preenchidos pelo usuário quando inserirmos os dados. Então copiaremos o número de CPF no campo correspondente, e tiraremos o foco, clicando em outro campo em seguida.

Informações da sua compra

1. Suas informações

Nome completo	Por favor, digite seu nome completo
<input type="text"/>	
Email	Por favor, verifique seu email
<input type="text"/>	
CPF	
86288366757	
Telefone com DDD	
<input type="text"/>	

2. Cartão

 	
Número do cartão	
Vencimento	Cód Seg
MM	AA
3 dígitos	
Selecionar parcelas para esta compra	
Suas informações serão coletadas de acordo com a Política de Privacidade do PayPal .	
<input type="checkbox"/> Eu quero receber informações importantes, ofertas	

O site não emitiu nenhum aviso no campo de CPF, apenas no de nome e email. Assim, podemos presumir que o CPF é válido. Testemos com o campo vazio.

Informações da sua compra

1. Suas informações

Nome completo	Por favor, digite seu nome completo
<input type="text"/>	
Email	Por favor, verifique seu email
<input type="text"/>	
CPF	Por favor, verifique seu CPF
<input type="text"/>	
Telefone com DDD	
<input type="text"/>	

2. Cartão

 	
Número do cartão	
Vencimento	Cód Seg
MM	AA
3 dígitos	
Selecionar parcelas para esta compra	
Suas informações serão coletadas de acordo com a Política de Privacidade do PayPal .	
<input type="checkbox"/> Eu quero receber informações importantes, ofertas	

Agora o site detecta que o CPF é inválido. Se preenchermos novamente e tirarmos o foco, a mensagem some.

Informações da sua compra

1. Suas informações

Nome completo	Por favor, digite seu nome completo
<input type="text"/>	
Email	Por favor, verifique seu email
<input type="text"/>	
CPF	
86288366757	
Telefone com DDD	<input type="text"/>

2. Cartão

E o segundo CPF?

Informações da sua compra

1. Suas informações

Nome completo	Por favor, digite seu nome completo
Email	Por favor, verifique seu email
CPF	Por favor, verifique seu CPF
98745366797	
Telefone com DDD	

2. Cartão

O site pede para verificar o CPF, ou seja, ele é inválido.

Mas como saber se um CPF é válido? Existe um algoritmo para isso, e vamos conhecê-lo. Usaremos o primeiro CPF como exemplo.

CPF: 862.883.667-57

Primeiro multiplica-se os 9 primeiros dígitos pela sequência decrescente de números de 10 à 2 e soma os resultados:

$$8 \times 10 = 80$$

$$8 \times 7 = 56$$

$$6 \times 4 = 24$$

$$6 \times 9 = 54$$

$$8 \times 6 = 48$$

$$6 \times 3 = 18$$

$$2 \times 8 = 16$$

$$3 \times 5 = 15$$

$$7 \times 2 = 14$$

O primeiro passo do algoritmo é multiplicar os nove primeiros algarismos, a começar por 10 até 2. Os resultados dessas multiplicações devem ser somados. Nesse exemplo, a soma é 325.

CPF: 862.883.667-57

Agora pegamos esse total e multiplicamos por 10 = 3250

Com esse total de 3250 e dividimos por 11 = 295

Mas não é o resultado da divisão que nos interessa, mas seu resto.

CPF: 862.883.667-57

O importante é o resto dessa divisão, que é 5. Esse é o primeiro dígito verificador do CPF

Os números verificadores são os que vêm depois do traço. Como o primeiro número verificador do CPF 862.883.667-57 é 5, por enquanto o CPF é válido. Ainda há a segunda etapa, para calcular o outro número verificador.

Nessa etapa, usaremos os dez primeiros algarismos do CPF e a rodada de multiplicação começa em 11.

CPF: 862.883.667-57

Para validar o segundo dígito, agora temos que pegar os 10 primeiros dígitos e multiplicá-lo pela sequência decrescente de 11 à 2 e somar os resultados:

$$8*11 = 88$$

$$8*8 = 64$$

$$6*5 = 30$$

$$6*10 = 60$$

$$8*7 = 56$$

$$6*4 = 24$$

$$2*9 = 18$$

$$3*6 = 18$$

$$7*3 = 21$$

$$5*2 = 10$$

Somando todos os resultados chegamos no valor de 389.

Novamente, usaremos o resultado da soma das multiplicações, e o multiplicaremos por dez.

CPF: 862.883.667-57

Agora pegamos esse total e multiplicamos por 10 = 3890

Com esse total de 3890 e dividimos por 11 = 353

E, novamente, é o resto da divisão por 11 que nos interessa.

CPF: 862.883.667-57

Vamos pegar o resto da divisão que é 7. Então esse CPF é válido

Como os dois dígitos verificadores corretos, podemos afirmar que o CPF é válido. Repetiremos o processo de verificação para o segundo CPF.

CPF: 987.453.667-97

Para validar o segundo dígito, agora temos que pegar os 10 primeiros dígitos e multiplicá-lo pela sequência decrescente de 11 à 2 e somar os resultados:

$$9*10 + 8*9 + 7*8 + 4*7 + 5*6 + 3*5 + 6*4 + 6*3 + 7*2 = 347$$

Com o resultado da soma desses valores, faremos uma nova multiplicação e uma divisão.

CPF: 987.453.667-97

Multiplicar essa soma por 10 = 3470

Dividimos por 11 = 315

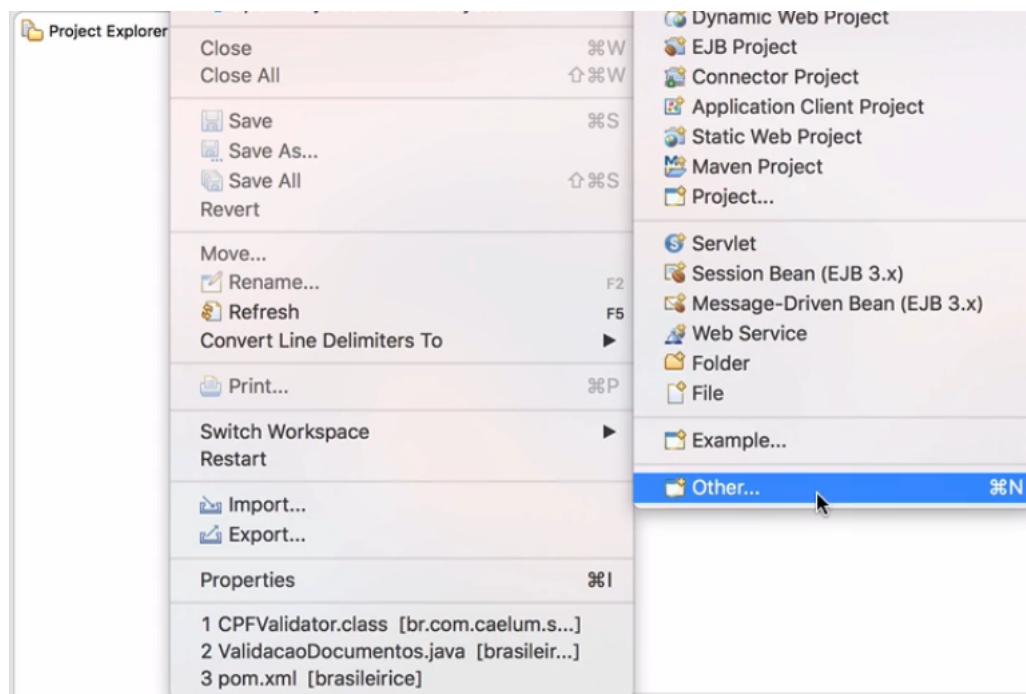
E o resto da divisão é 5.

Então já podemos parar aqui e dizer que esse CPF é inválido

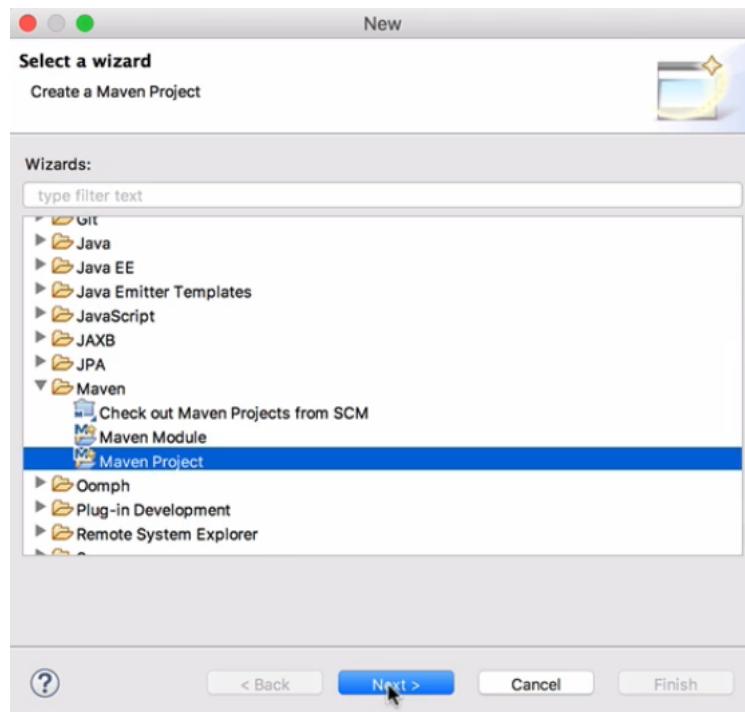
Se o primeiro dígito verificador não corresponde ao que está na sequência do CPF, não é necessário verificar o segundo; o CPF já é inválido.

Aprenderemos a fazer isso no Java. Para isso, abriremos o Eclipse, programa que será usado ao longo do curso todo. Criaremos um projeto novo, mas usaremos o Maven para nos ajudar a gerenciar.

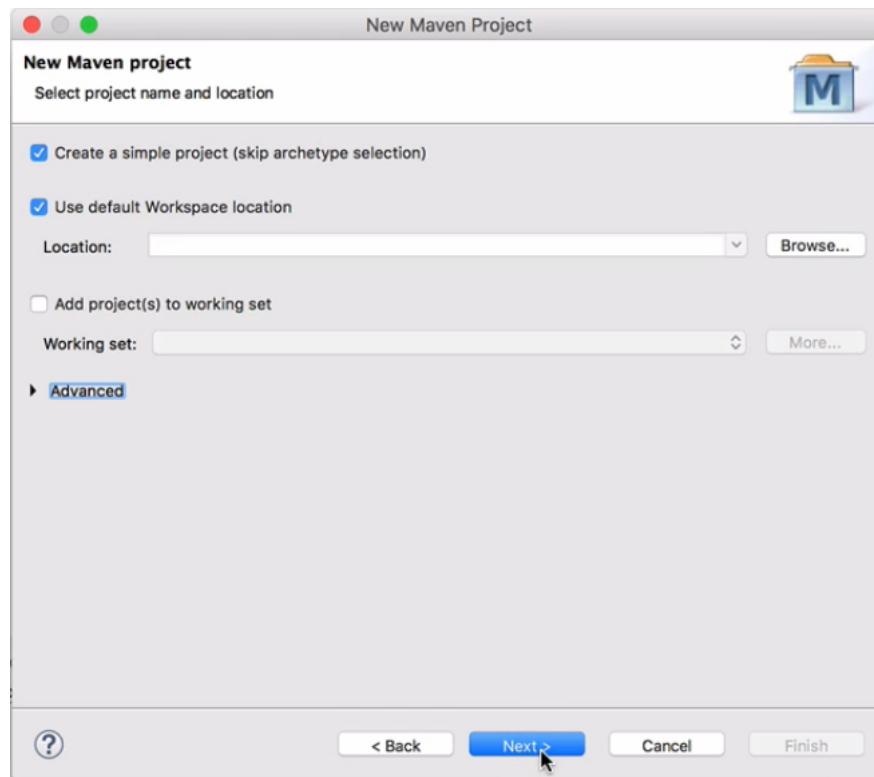
No menu superior, seguiremos o caminho `File > New > Other...` .



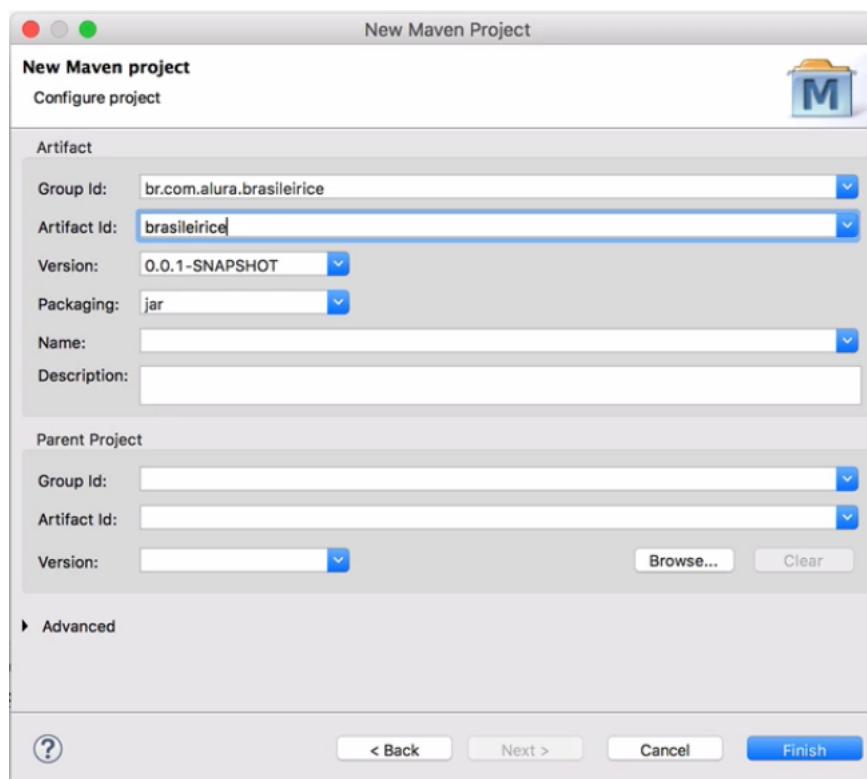
Que abrirá a seguinte janela. Nela, devemos encontrar e selecionar `Maven Project` .



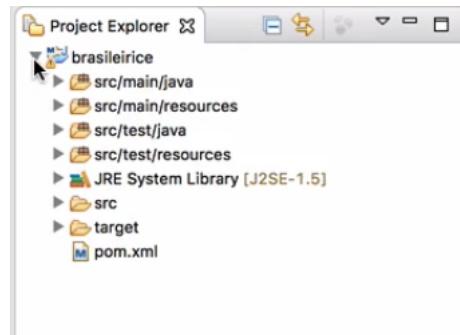
Essa escolha abrirá mais opções. Optaremos por `Simple project` .



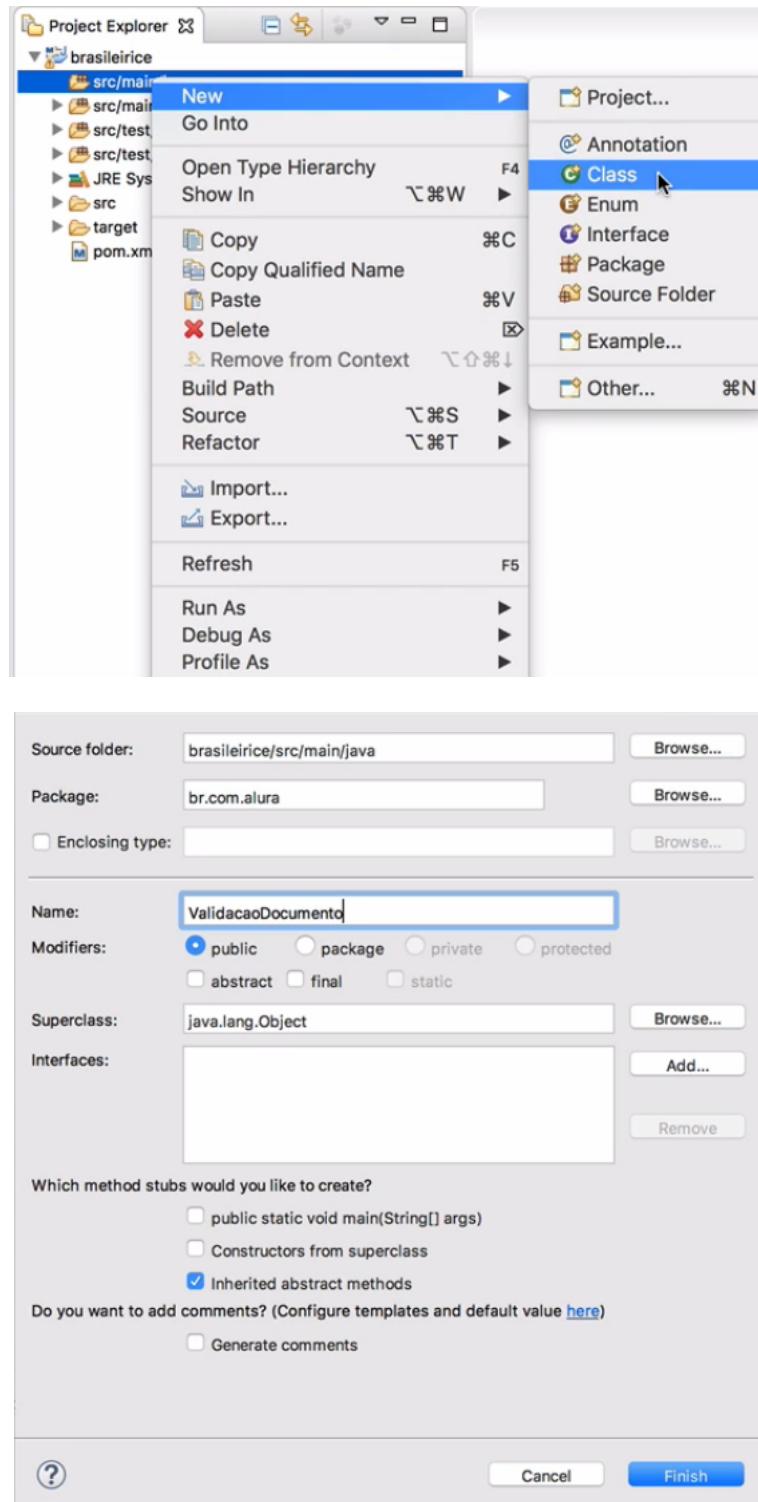
E, em seguida, precisamos preencher os campos `Group Id` e `Artifact Id`. Serão, respectivamente, `br.com.alura.brasileirice` e `brasileirice`.



Ao clicar em `Finish`, o programa baixará o projeto.



Uma das pastas criadas é a de source do próprio Java. Nela, criaremos uma nova classe para fazer todo aquele algoritmo. Basta clicar com o botão direito e em `New > Class`.

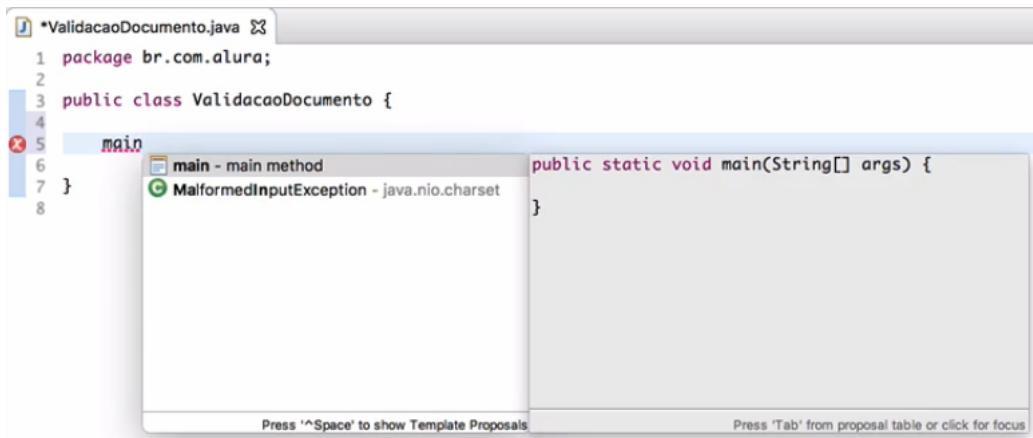


No campo Package colocaremos a raiz mesmo `br.com.alura`. O nome da classe será `ValidacaoDocumento`. Clicando em Finish, a classe estará criada e o programa nos exibe o seguinte código:

```
package br.com.alura;

public class ValidacaoDocumento {
```

Vamos criar o método `main` para validar os CPFs, usando o atalho da minha IDE, com `Ctrl + Espaço`.



O código fica da seguinte maneira.

```
package br.com.alura;

public class ValidacaoDocumento {

    public static void main (String[] args) {

    }
}
```

Agora precisamos pegar cada algarismo do CPF e multiplicar pelo número que corresponde à sua posição (10, 9, 8, 7, 6, 5, 4, 3, 2), depois somar e multiplicar por dez e dividir por onze... É um trabalhão, não é?

Será que não tem nada pronto na internet, nada que possamos reaproveitar? Como o algoritmo é bem repetitivo, é provável que alguém já tenha feito algo parecido. Vamos verificar no framework open source [Stella](http://stella.caelum.com.br/) (<http://stella.caelum.com.br/>), da Caelum. Você pode baixar e usar, além de mandar *requests* para fazer alterações. Ele possui validadores de documentos que só existem no Brasil, como CPF, CNPJ, título de eleitor, entre outros.



Simplificando o desenvolvimento no Brasil.

Bean Validation

```
@Entity
public class Modelo {
    @Id
    @GeneratedValue
    private Long id;
    @CNPJ
    private String cnpj;
    @CPF
    private String cpf;
```

JSF 2

```
<h:outputLabel for="cpfFormatado" value="cpf com formatação:" />
<h:inputText id="cpfFormatado" value="#{usuarioBean.cpfFormatado}">
    <stellaweb:validateCPF formatted="true" />
</h:inputText>
<!-- outra opção -->
<h:outputLabel value="cpf:" />
<h:inputText id="cpf">
    <f:validator validatorId="StellaCPFValidator" />
</h:inputText>
```

Validadores

```
String cpf = "867.554.707-24";
List<ValidationMessage> messages =
    new CPFValidator().invalidMessagesFor(cpf);

for (ValidationMessage error : messages) {
    System.out.println(error.getMessage());
}

new CPFValidator().assertValid("867.554.707-24");
```

Conversor de número por extenso

```
FormatoDeExtenso formato =
    new FormatoDeSegundosComMilesimos();
NumericToWordsConverter converter =
    new NumericToWordsConverter(formato);
String tempo = converter.toWords(9.52);
String message = "Técnico diz que Bolt poderia " +
    "ter feito 100m em " + tempo;

System.out.println(message);
// Técnico diz que Bolt poderia ter feito 100m
// em nove segundos e quinhentos e vinte milésimos de segundo.
```

E como usar? Clicando em "Downloads", na parte superior, veremos a seguinte tela.

Download

mariofts edited this page on 19 Nov 2014 · 13 revisions

Download

Download dos jar individualmente [na página de downloads](#). Lembrando de que o stella-core é necessário por todos os outros.

Configurando como dependência no maven:

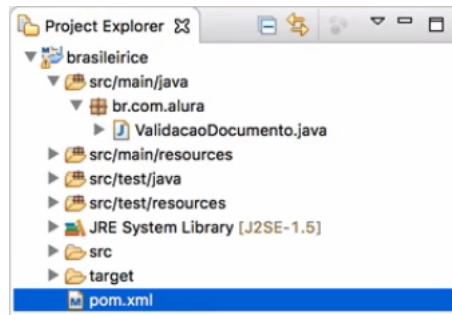
Stella core:

```
<dependency>
    <groupId>br.com.caelum.stella</groupId>
    <artifactId>caelum-stella-core</artifactId>
    <version>2.1.2</version>
</dependency>
```

▼ Pages (26)

- [Home](#)
- [Anotando o modelo](#)
- [Calculando Frete](#)
- [Camada de negócios](#)
- [Customizando o layout dos boletos](#)
- [Download](#)
- [Downloads do caelum stella](#)

Como estamos usando o Maven, podemos aproveitar essa parte de dependência. Vamos copiá-la no arquivo `pom.xml`, que está no Eclipse.



Dentro dele, temos:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-:
<modelVersion>4.0.0</modelversion>
<groupId>br.com.alura.brasileirice</groupId>
<artifactId>Brasileirice</artifactId>
<version>0.0.1-SNAPSHOT</version>
</project>
```

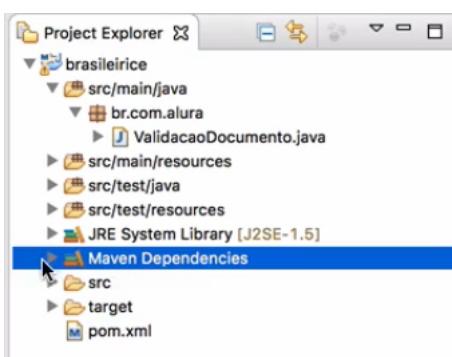
Precisamos criar uma dependência, e, dentro dela inserir o código de dependência copiado da Stella.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-:
<modelVersion>4.0.0</modelversion>
<groupId>br.com.alura.brasileirice</groupId>
<artifactId>Brasileirice</artifactId>
<version>0.0.1-SNAPSHOT</version>

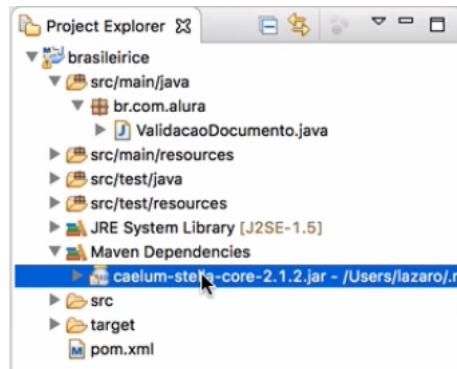
<dependencies>
  <dependency>
    <groupId>br.com.caelum.stella</groupId>
    <artifactId>caelum-stella-core</artifactId>
    <version>2.1.2</version>
  </dependency>
</dependencies>

</project>
```

Com a dependência inserida, o programa baixará para nós a pasta `Maven Dependencies`



E, dentro dela, estará o `.jar` do Stella.



Voltando à classe main, podemos validar o CPF. Para isso, precisamos inserir um CPF, e faremos isso com uma `String`. Será o primeiro CPF com o qual lidamos.

```
package br.com.alura;

public class ValidacaoDocumento {

    public static void main (String[] args) {
        String cpf="86288366757";
    }

}
```

Agora precisamos fazer a validação. Criaremos um `CPFValidator`, que é uma classe do Stella. Criaremos uma instância dessa classe, um objeto, que chamaremos de validador.

```
package br.com.alura;

public class ValidacaoDocumento {

    public static void main (String[] args) {
        String cpf="86288366757";
        CPFValidator validador = new CPFValidator();
    }

}
```

Essa classe tem um método chamado `isValid`, que verifica se o CPF é válido ou não. Chamaremos esse método.

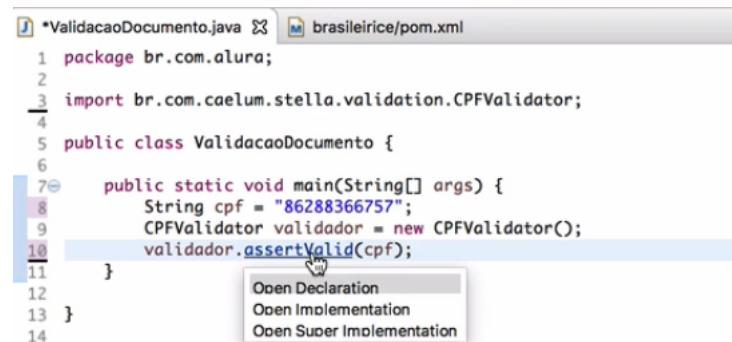
```
package br.com.alura;

public class ValidacaoDocumento {

    public static void main (String[] args) {
        String cpf="86288366757";
        CPFValidator validador = new CPFValidator();
        validador.assertValid(cpf);
    }

}
```

Ainda é preciso mostrar no console a mensagem que avisa se o CPF é válido ou não. Vamos clicar sobre `assertValid` para abrir a implementação em uma nova aba.



```

1 package br.com.alura;
2
3 import br.com.caelum.stella.validation.CPFValidator;
4
5 public class ValidacaoDocumento {
6
7     public static void main(String[] args) {
8         String cpf = "86288366757";
9         CPFValidator validador = new CPFValidator();
10        validador.assertValid(cpf);
11    }
12
13 }
14

```

O que vemos nessa aba é o seguinte código:

```

@Override
public boolean isEligible(String value) {
    if (value == null) {
        return false;
    }
    boolean result;
    if (isFormatted){
        result = FORMATED.matcher(value).matches();
    } else {
        result = UNFORMATED.matcher(value).matches();
    }
    return result;
}

@Override
public void assertValid(String cpf) {
    Lists<ValidationMessage> errors = getInvalidValues(cpf);
    if (!errors.isEmpty()) {
        throw new InvalidStateException(errors);
    }
}

@Override
public List<ValidationMessage> invalidMessagesFor(String cpf){
    return getInvalidValues(cpf);
}

```

Observe que o `assertValid` faz um `InvalidStateException`. Tentaremos capturar essa exceção na nossa validação, usando `try/catch`. Colocaremos uma mensagem que aparecerá no console (`System.out.println`), concatenado com o erro (`e`).

```

package br.com.alura;

public class ValidacaoDocumento {

    public static void main (String[] args) {
        String cpf="86288366757";
        CPFValidator validador = new CPFValidator();

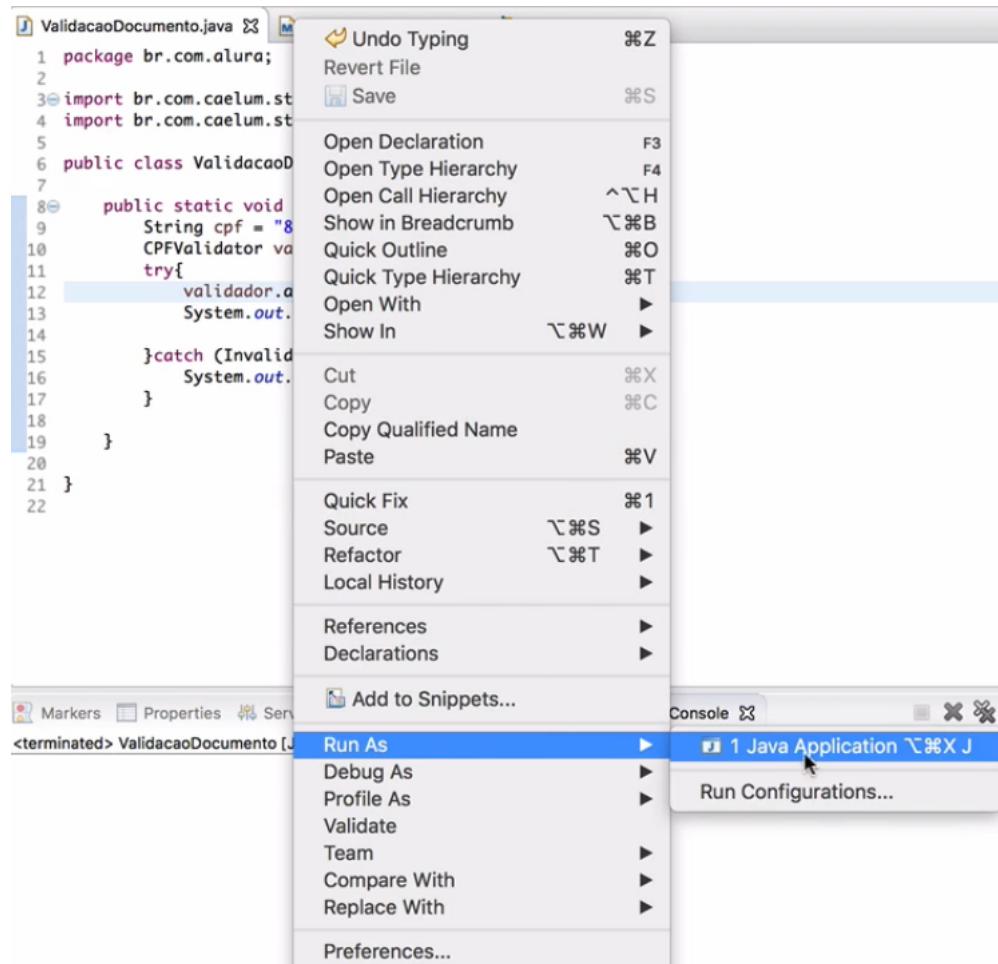
```

```
try{  
  
}catch (InvalidStateException e) {  
    System.out.println("CPF INVÁLIDO : " + e);  
}  
validador.assertValid(cpf);  
}  
  
}
```

Levaremos o validador para dentro do `try`, e acrescentaremos a mensagem de sucesso, para o caso de o CPF passar na validação.

```
package br.com.alura;  
  
public class ValidacaoDocumento {  
  
    public static void main (String[] args) {  
        String cpf="86288366757";  
        CPFValidator validador = new CPFValidator();  
        try{  
            validador.assertValid(cpf);  
            System.out.println("CPF VÁLIDO");  
  
        }catch (InvalidStateException e) {  
            System.out.println("CPF INVÁLIDO : " + e);  
        }  
    }  
}
```

Agora podemos rodar, clicando com o botão direito do mouse e em `Run As > Java Application`.



E o console nos mostra a mensagem CPF VÁLIDO



Vamos testar com um CPF inválido? Usaremos o segundo CPF, que já sabemos que é inválido.

```
package br.com.alura;

public class ValidacaoDocumento {

    public static void main (String[] args) {
        String cpf="98745366797";
        CPFValidator validador = new CPFValidator();
        try{
            validador.assertValid(cpf);
            System.out.println("CPF VÁLIDO");

        }catch (InvalidStateException e) {
            System.out.println("CPF INVÁLIDO : " + e);
        }
    }
}
```

}

E vamos rodar novamente. O console exibe o seguinte:



The screenshot shows an IDE's console window. The title bar says 'ValidacaoDocumento [Java Application]'. The console output shows the error: 'CPF INVÁLIDO : br.com.caelum.stella.validation.InvalidStateException: Validation errors: [CPFError : INVALID CHECK DIGITS]'. The window has various icons for file operations, search, and help.

O erro que ele exibe é `INVALID CHECK DIGITS`, que nos diz que os dígitos verificadores estão errados. Não precisamos fazer todo o algoritmo para descobrir isso, graças ao Stella.

É importante ter em mente que nem sempre um CPF válido pertence a uma pessoa. Às vezes conseguimos gerar números válidos que não correspondem ao cadastro de ninguém, mas que passam no algoritmo. Quando mandamos esse dado para um sistema de pagamento, como PayPal, conseguimos detectar se ele passa no algoritmo, mas não se ele pertence a alguém.

Há vários sites na internet que conseguem gerar CPFs válidos. Se jogarmos "gerador de CPF" no Google, você verá um grande número deles.



The screenshot shows a website titled 'Gerador de CPF'. The top navigation bar has links for 'Gerador de CPF', 'Gerador de Cartão de Crédito', 'Gerador de CNPJ', and 'Gerador de Senha'. The main title 'Gerador de CPF' is in a large, bold, black font. Below the title, a text box contains the placeholder 'CPF'. Under the text box are two buttons: 'Validar CPF' and 'Gerar CPF'. A small checkbox labeled 'Pontuação' is positioned between the buttons. The text 'O gerador de cpf tem como objetivo a geração de CPF para fins de teste de software.' is displayed below the buttons.

Coparemos um número gerado pelo site no nosso validador.

```
package br.com.alura;

public class ValidacaoDocumento {

    public static void main (String[] args) {
        String cpf="28048804948";
        CPFValidator validador = new CPFValidator();
        try{
            validador.assertValid(cpf);
            System.out.println("CPF VÁLIDO");

        }catch (InvalidStateException e) {
            System.out.println("CPF INVÁLIDO : " + e);
        }
    }
}
```

E colocaremos para rodar, com o Run As > Java Application . O console nos retorna a seguinte mensagem:



CPF VÁLIDO

Ou seja, os CPFs gerados pelo site realmente são válidos. O que não quer dizer que seja de uma pessoa.

Também é importante tomar cuidado com os números repetidos. vamos fazer um teste.

```
package br.com.alura;

public class ValidacaoDocumento {

    public static void main (String[] args) {
        String cpf="22222222222";
        CPFValidator validador = new CPFValidator();
        try{
            validador.assertValid(cpf);
            System.out.println("CPF VÁLIDO");

        }catch (InvalidStateException e) {
            System.out.println("CPF INVÁLIDO : " + e);
        }
    }
}
```

Se fizermos pelo algoritmo, veremos que o número 2222222222 passa no teste. Mas, ao rodarmos no verificador, ele nos mostra a seguinte mensagem:



CPF INVÁLIDO : br.com.caelum.stella.validation.InvalidStateException: Validation errors: [CPFError : REPEATED DIGITS]

A biblioteca já é inteligente o bastante para detectar que os números desse CPF são repetidos. Se o site não estiver devidamente preparado, esse número será reconhecido como um CPF válido, por passar no algoritmo.

São esses os cuidados que precisamos tomar na hora de validar CPFs com o algoritmo. Até a próxima!