

# Curso de Java SE



Este material é exclusivo do canal (youtube): [canalfessorbruno](https://www.youtube.com/c/canalfessorbruno) e do site [www.cfbcursos.com.br](http://www.cfbcursos.com.br)  
não pode ser distribuído ou comercializado por outra fonte.



**Apostila: versão 1.0**  
**04/07/2016**

**[www.youtube.com/canalfessorbruno](http://www.youtube.com/canalfessorbruno)**

**[www.cfbcursos.com.br](http://www.cfbcursos.com.br)**

**[canalfessorbruno@gmail.com](mailto:canalfessorbruno@gmail.com)**

**[www.facebook.com/canalfessorbruno](http://www.facebook.com/canalfessorbruno)**

**twitter: @fessorBruno**



## Sumário

Introdução.....	6
NetBeans.....	6
Iniciando um novo projeto Java no NetBeans .....	7
Comandos de impressão em tela “print” e “println” .....	10
Variáveis.....	11
Escopo .....	11
Variáveis primitivas.....	14
Variáveis globais.....	15
Variáveis locais.....	15
static.....	16
Variáveis com mesmo nome.....	17
Operações com variáveis .....	18
Operadores .....	19
Constantes .....	22
Operações de casting.....	23
Comentários.....	24
Comentário de linha única .....	24
Comentário multilinha ou em bloco .....	25
IF.....	25
IF ELSE .....	26
AND(&&), OR(   ), NOT(!).....	28
Ternário.....	30
Switch.....	31
While .....	32
Do while .....	35
For .....	36
Arrays unidimensionais (vetor).....	38
length (constante).....	40
Instanciando e inserindo elementos no array .....	41
Arrays bidimensionais (matriz) .....	41
A classe Arrays .....	43
binarySearch .....	43
copyOf .....	44
copyOfRange.....	45
equals.....	46



fill.....	47
sort .....	48
Lendo valores pelo teclado .....	49
Scanner .....	49
BufferedReader .....	51
Trabalhando com a classe Calendar .....	52
try catch finally – Tratamento de Exceções .....	54
Foreach .....	57
Stack - Pilha .....	58
push – Inserindo elementos na pilha .....	59
firstElement / lastElement.....	59
get .....	60
indexOf.....	61
peek.....	62
size .....	62
isEmpty.....	63
pop .....	64
remove .....	65
search.....	67
add .....	68
clear.....	69
sort .....	70
Queue – Fila .....	71
add e poll .....	72
Lista .....	73
Métodos.....	74
Métodos com parâmetro de entrada .....	75
Métodos com retorno.....	77
Classes.....	79
Pacotes.....	79
Instanciando novos objetos.....	82
Método construtor .....	85
Sobrecarga de métodos .....	86
Public x Private.....	88
Herança .....	89
Protected.....	95



Get e Set.....	95
this.....	98
Passando objetos como parâmetros para métodos.....	99
Classes abstratas.....	100
@Override.....	103
Interfaces.....	106
O método main.....	119
Enum.....	120
super.....	121
Considerações finais.....	124



## Introdução

Java é uma linguagem de programação bastante versátil, hoje em dia vemos java aplicado em diversas plataformas e em diversos dispositivos diferentes. Neste material irei apresentar a linguagem Java SE a você de uma forma bem fácil e progressiva, você vai aprender os principais conceitos da linguagem.

Existem diversas distribuições de Java, SE, EE, FX, ME, neste material iremos aprender o Java SE (Standard Edition) que é a principal distribuição do Java, precisamos partir da SE para entender as outras distribuições.

Então vamos começar que temos muito a aprender.

## NetBeans

Neste curso iremos usar a ferramenta NetBeans, o endereço para download é “netbeans.org”.

Clique no link “Download” e escolha uma das duas opções destacadas em vermelho na ilustração a seguir.



**NetBeans** | NetBeans IDE | NetBeans Platform | Plugins | Docs & Support | Community | Partners | Search

HOME / Download

## Download o NetBeans IDE 8.1

8.0.2 | 8.1 | Desenvolvimento | JDK9 Branch | Arquivo

Endereço de email (opcional):

Inscriver-se na newsletter: ☒ Mensal ☐ Semanal

☒ Permito me contatar neste email

Idioma do IDE: **Português (Brasil)** | Plataforma: **Windows**

Nota: Tecnologias em cinza não são suportadas para esta plataforma.

### Distribuições para baixar do NetBeans IDE

Tecnologias suportadas *	Java SE	Java EE	HTML5/JavaScript	PHP	C/C++	Tudo
① SDK da plataforma NetBeans	•	•				•
① Java SE	•	•				•
① Java FX	•	•				•
① Java EE		•				•
① Java ME						•
① HTML5/JavaScript		•	•	•		•
① PHP			•	•		•
① C/C++					•	•
① Groovy						•
① Java Card(tm) 3 Connected						•
Servidores embutidos						
① GlassFish Server Open Source Edition 4.1.1		•				•
① Apache Tomcat 8.0.27		•				•

**Download** **Download** **Download x86** **Download x86** **Download x86** **Download**

**Download x64** **Download x64** **Download x64**

95 MB livre(s) 192 MB livre(s) 104 - 107 MB livre(s) 104 - 107 MB livre(s) 106 - 110 MB livre(s) 215 MB livre(s)

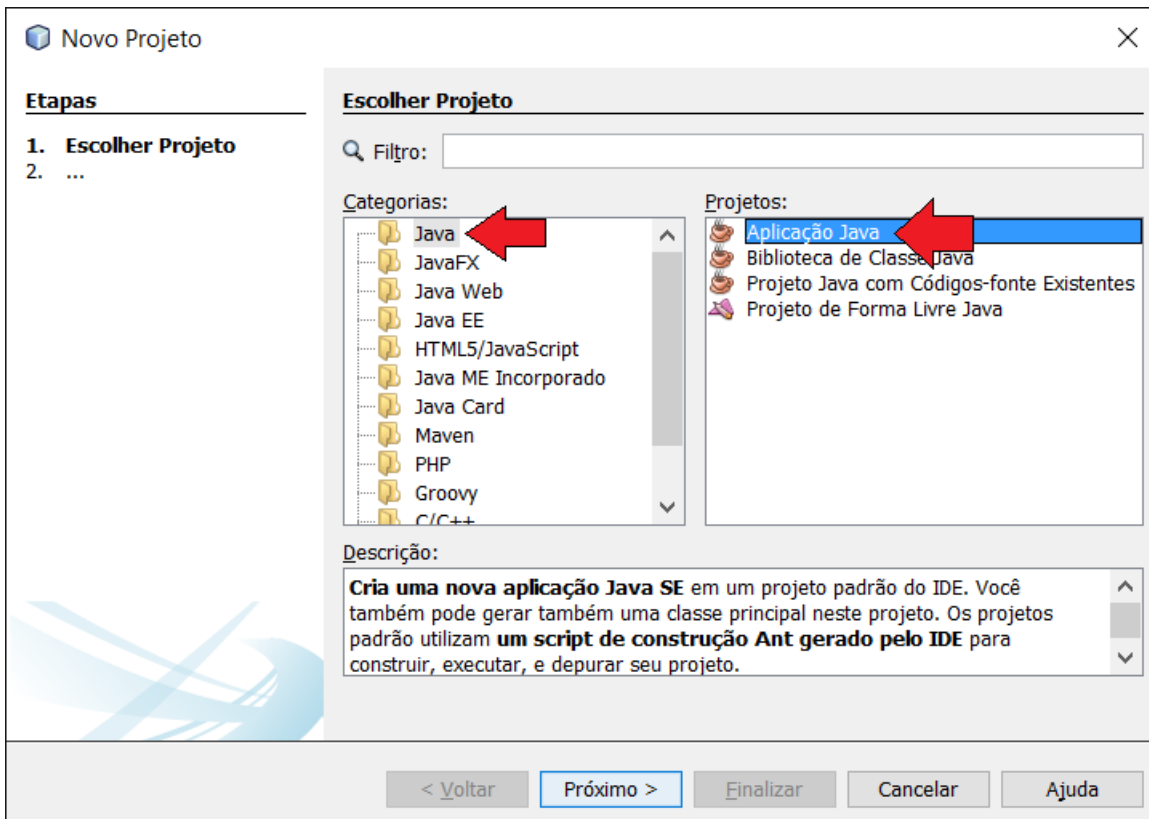
Após baixar inicie a instalação da ferramenta.

## Iniciando um novo projeto Java no NetBeans

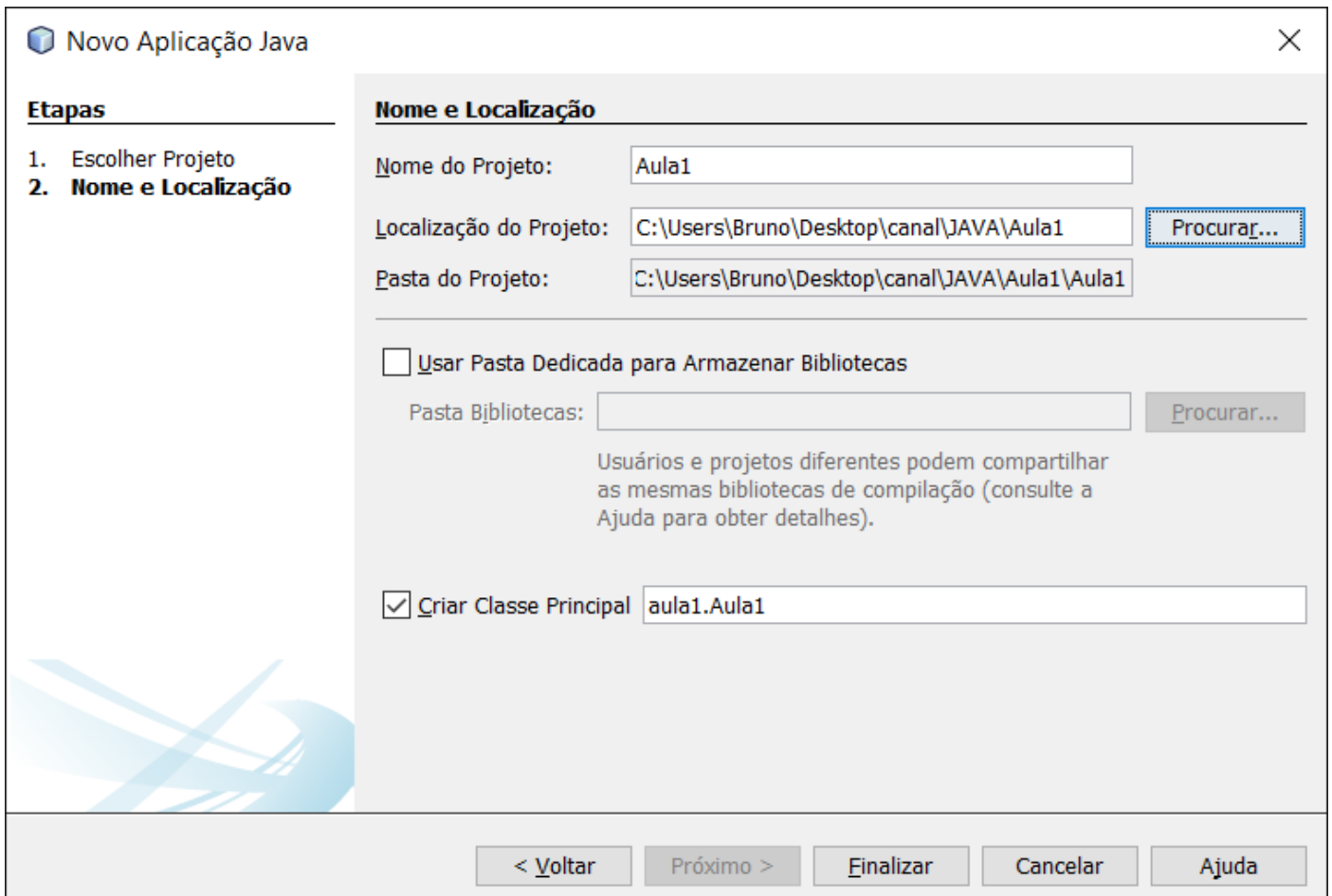


Ao iniciar o NetBeans clique no menu ARQUIVO – NOVO PROJETO ou clique no botão “Novo Projeto”.

Selecione a categoria “Java” e o projeto “Aplicação Java”, clique no botão “Proximo >”.



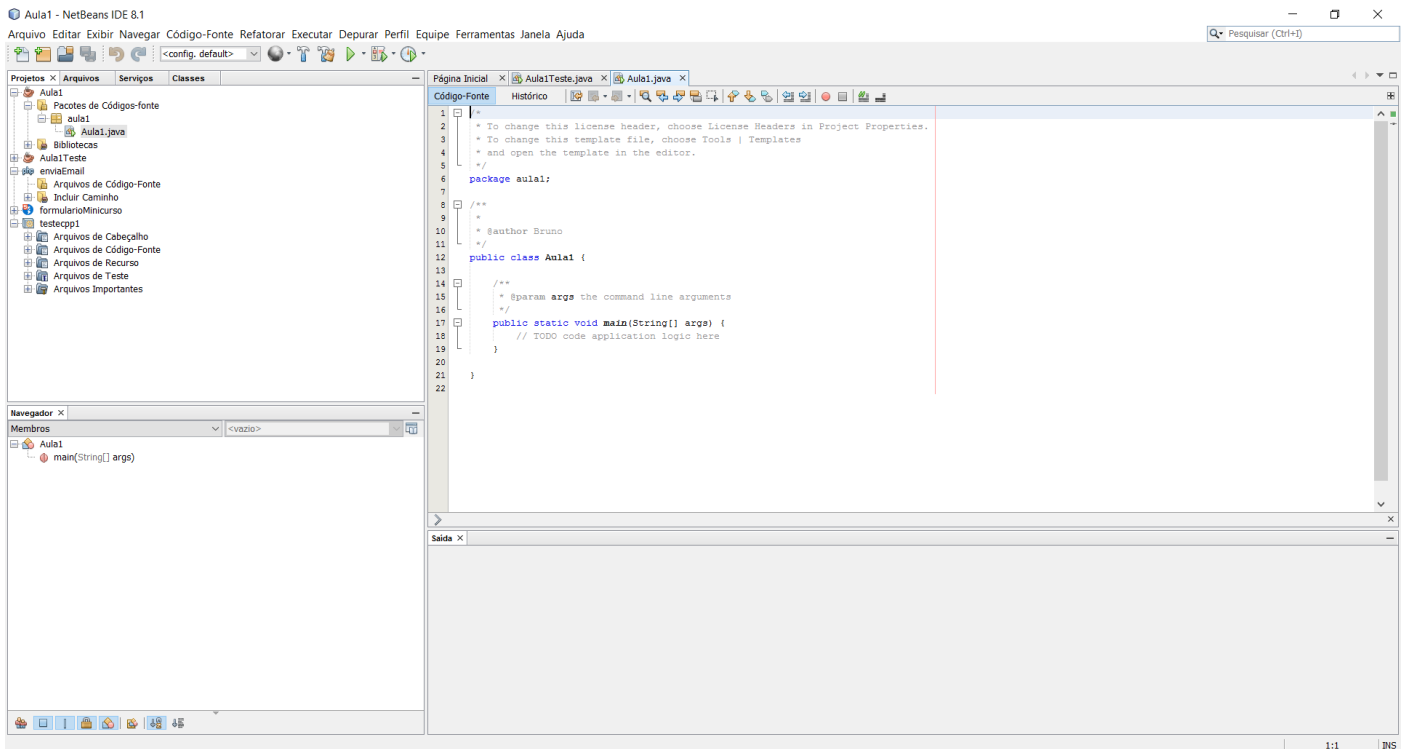
Em “Nome do Projeto” digite “Aula1”, em “Localização do Projeto” clique em “Procurar” e selecione a pasta onde você quer salvar seu projeto. Clique no botão “Finalizar” para gerar o projeto.







Após a conclusão da geração do projeto será aberto o programa com o código fonte básico já inserido.



O código básico é formado pelos comandos que vou explicar a seguir.

```
1  /**
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package aula1;
7
8  /**
9  *
10 * @author Bruno
11 */
12 public class Aula1 {
13
14     /**
15     * @param args the command line arguments
16     */
17     public static void main(String[] args) {
18         // TODO code application logic here
19     }
20
21 }
22
```

Na linha 6 o primeiro comando identifica a pasta onde estamos armazenando nosso projeto.

A linha 12 iniciamos de fato o código principal do nosso arquivo informando que a classe é pública, mais adiante discutiremos sobre isso.

Na linha 17 temos a declaração do método principal “main”, o código fonte principal da classe será inserido aqui dentro deste método a partir da linha 18.



## Comandos de impressão em tela “print” e “println”

Em nosso projeto “Aula1” que acabamos de criar vamos entender como imprimir texto na saída do console. Para tal tarefa podemos usar os métodos “print” ou “println”, a diferença é que “println” faz a quebra de linha após o texto, ambos recebem um string como parâmetro.

Os comandos “print” e “println” estão na classe “System”, veja a seguir como usar estes métodos para gerar saídas no console.

OBS: Vou apagar os comentários para que o código fique mais sucinto.

```
1 package aula1;
2
3 public class Aula1 {
4
5     public static void main(String[] args) {
6
7         System.out.print("Canal Fessor Bruno");
8
9     }
10
11 }
12
```

Para rodar programa vamos usar o botão “Executar Projeto”.



Na tela de saída veremos o resultado do método “print”.

```
Saída - Aula1 (run) X
run:
Canal Fessor BrunoCONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

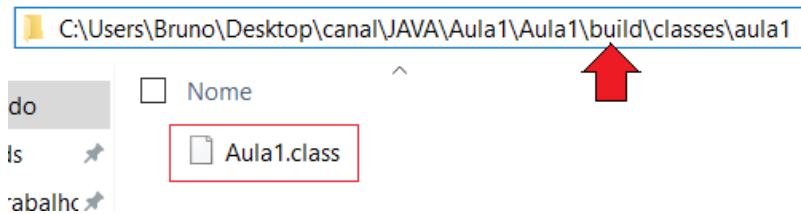
Vamos trocar pelo método “println” e rodar o projeto novamente, observe que agora existe a quebra de linha após a string “Canal Fessor Bruno”.

```
1 package aula1;
2
3 public class Aula1 {
4
5     public static void main(String[] args) {
6
7         System.out.println("Canal Fessor Bruno");
8
9     }
10
11 }
12
```

```
Saída - Aula1 (run) X
run:
Canal Fessor Bruno
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



Ao rodar nosso programa será criado o arquivo “bytecode” que é o resultado da “compilação” do código fonte Java. Este arquivo é do tipo “.class” e vai se encontrar na pasta “build” que foi gerada quando clicamos para rodar o programa.



Este é o nosso programa “compilado” digamos assim. É o arquivo que será executado pela máquina virtual Java.

## Variáveis

Em todos os programas necessitamos de locais para armazenamento temporário de informações/dados, usamos as variáveis para este fim.

É muito importante saber declarar variáveis de forma correta para que não tenhamos problemas, basicamente em Java precisamos informar o tipo e o nome, mas podemos declarar também o tipo de acesso desta variável o que veremos mais adiante.

## Escopo

É extremamente importante observar e saber lidar com o escopo das variáveis, basicamente o escopo de uma variável determina de que parte do código esta variável estará acessível.

Se uma variável foi declarada dentro de uma classe por exemplo, poderá ser acessada por todos os métodos desta classe de forma direta, basicamente dizemos que esta variável tem um escopo global à classe que pertence.

Se uma variável for declarada dentro de um método, esta variável pertence ao método, e pode ser chamada como uma variável local ao método que pertence.

Vou apresentar um exemplo de escopo.

```
12 public class Testes {
13
14     public static void main(String[] args) {
15         int v1=10;
16
17         if(v1==10){
18             int v2=20;
19         }
20         System.out.println(v1);
21         System.out.println(v2);
22     }
23
24 }
```

No código de exemplo acima, vemos o comando de impressão na linha 21 tentando imprimir o valor da variável v2, mas note que v2 foi declarada no escopo do bloco do comando IF, desta maneira v2 só pode ser acessada de dentro do bloco IF, este código então, possui um erro e não será compilado.

Vamos corrigir.



```
12 public class Testes {
13
14     public static void main(String[] args) {
15         int v1=10;
16
17         if(v1==10){
18             int v2=20;
19             System.out.println(v2);
20         }
21         System.out.println(v1);
22     }
23
24 }
```

Note que agora o comando de impressão da variável v2 está dentro do scope do bloco IF na linha 19, desta maneira não temo um erro e o programa será compilado.

Vamos a outra solução.

```
12 public class Testes {
13
14     public static void main(String[] args) {
15         int v1=10;
16         int v2=20;
17
18         if(v1==10){
19             //Bloco sem comandos
20         }
21         System.out.println(v1);
22         System.out.println(v2);
23     }
24 }
```

Na solução acima declaramos a variável v2 como global ao código do método main, também é um código sem erros, inclusive conseguimos acessar as variáveis v1 e v2 perfeitamente de dentro do bloco IF, veja o código a seguir.

```
12 public class Testes {
13
14     public static void main(String[] args) {
15         int v1=10;
16         int v2=20;
17
18         if(v1==10){
19             System.out.println(v1);
20             System.out.println(v2);
21         }
22     }
23 }
```

Entendendo um pouco mais, veja o código a seguir.



```
12 public class Testes {
13
14     public static void main(String[] args) {
15         int v1=10;
16         int v2=20;
17
18         if(v1==10){
19             System.out.println(v1);
20             System.out.println(v2);
21         }
22     }
23
24     public static void cfb() {
25         System.out.println(v1);
26         System.out.println(v2);
27     }
28 }
29
30 }
```

Note que agora temos um método chamado “cfb” que tenta imprimir os valores das variáveis v1 e v2, este código está errado, pois v1 e v2 estão fora do escopo do método “cfb”.

Veja a alteração a seguir.

```
13 public class Testes {
14
15     public static void main(String[] args) {
16         int v1=10;
17         int v2=20;
18
19         if(v1==10){
20             System.out.println(v1);
21             System.out.println(v2);
22         }
23     }
24
25     public static void cfb() {
26         int v1=1;
27         int v2=2;
28         System.out.println(v1);
29         System.out.println(v2);
30     }
31
32 }
```

Note que agora temos declaradas v1 e v2 dentro do método “cfb” e neste caso não existe mais erro, um detalhe muito importante que devemos observar é que v1 e v2 no escopo do método “cfb” são diferentes das variáveis v1 e v2 do método “main”.

Quando temos métodos diferentes, no caso do código acima podemos utilizar variáveis de mesmo nome, mas se forem no mesmo método, não podemos utilizar variáveis com mesmo nome.

Veja o código a seguir.



```
12 public class Testes {
13
14     public static void main(String[] args) {
15         int v1=10;
16         int v2=20;
17
18         if(v1==10){
19             int v1=30;
20             int v2=40;
21             System.out.println(v1);
22             System.out.println(v2);
23         }
24     }
25 }
26
27 }
```

Note que nas linhas 19 e 20 existem dois alertas de erro, mesmo que o escopo seja diferente, quando estamos no mesmo método no caso “main”, não podemos usar variáveis com mesmo nome.

Reescrevendo o programa acima para a forma correta temos o código a seguir.

```
12 public class Testes {
13
14     public static void main(String[] args) {
15         int v1=10;
16         int v2=20;
17
18         if(v1==10){
19             int v3=30;
20             int v4=40;
21             System.out.println(v3);
22             System.out.println(v4);
23         }
24
25         System.out.println(v1);
26         System.out.println(v2);
27     }
28 }
29
30 }
```

## Variáveis primitivas

Confira a tela a seguir com alguns tipos primitivos de dados que podemos usar em Java.

Tipo	Descrição
<b>boolean</b>	Tipo lógico, verdadeiro/true/1 ou falso/false/0
<b>char</b>	Tipo caractere, dados alfanuméricos.
<b>String</b>	Tipo texto.
<b>byte</b>	Inteiro de 8 bits. Valores entre -2 <sup>7</sup> =-128 e 2 <sup>7</sup> -1=127.
<b>short</b>	Inteiro de 16 bits. Valores entre -2 <sup>15</sup> =-32.768 e 2 <sup>15</sup> -1=32.767
<b>int</b>	Inteiro de 32 bits. Valores entre -2 <sup>31</sup> =-2.147.483.648 e 2 <sup>31</sup> -1=2.147.483.647.
<b>long</b>	Inteiro de 64 bits. Valores entre -2 <sup>63</sup> e 2 <sup>63</sup> -1.
<b>float</b>	Números reais (ponto flutuante) de 32 bits. Valores entre 1.40239846e-46 e 3.40282347e+38
<b>double</b>	Números reais (ponto flutuante) de 64 bits. Valores entre 4.94065645841246544e-324 e 1.7976931348623157e+308

Agora vamos aprender como usar variáveis, vamos criar um novo projeto com nome “Aula2”.



Em seguida temos o código fonte após remover os comentários.

```
1 package aula2;
2
3 public class Aula2 {
4
5     public static void main(String[] args) {
6         |
7     }
8
9 }
10
```

## Variáveis globais

É muito importante prestarmos atenção ao escopo das variáveis, basicamente de onde ou até onde elas podem ser acessadas.

O local que declaramos uma variável define seu escopo, vamos observar o exemplo a seguir.

```
1 package aula2;
2
3 public class Aula2 {
4
5     int numG;
6
7     public static void main(String[] args) {
8
9     }
10
11 }
12
```

Desta maneira declaramos uma variável do tipo “int” com nome “numG” como uma variável global, que pode ser acessada por qualquer método da nossa classe.

## Variáveis locais

São variáveis criadas dentro dos métodos, por exemplo, podem ser acessadas somente pelo método onde está declarado, ou seja, seu escopo é somente o método onde foi declarada.

No código fonte a seguir declaramos uma variável local ao método “main” na linha 9 do tipo “int” com nome “numL”.

```
1 package aula2;
2
3 public class Aula2 {
4
5     int numG;
6
7     public static void main(String[] args) {
8
9         int numL;
10
11     }
12
13 }
14
```

Vamos tentar atribuir um valor à variável “numG” de dentro do método “main”.



```
1 package aula2;
2
3 public class Aula2 {
4     int numG;
5
6     public static void main(String[] args) {
7
8         int numL;
9
10        numG=10;
11    }
12
13 }
14
15
16
```

Note que ocorreu um erro, mas, eu disse que variáveis locais podem ser acessadas de qualquer método! Sendo assim, este erro não deveria acontecer.

Vamos aos detalhes, apontando o cursor do mouse para o símbolo vermelho do erro na linha 11, vemos uma mensagem que pode esclarecer o problema.

```
1 package aula2;
2
3 public class Aula2 {
4     int numG;
5
6
7
8
9
10
11    numG=10;
12
13 }
14
15
16
```

non-static variable numG cannot be referenced from a static context  
----  
(Alt-Enter mostra dicas)

“non-static variable numG cannot be referenced from a static contexto” esta mensagem está nos dizendo que a variável “numG” não é “staic” portanto não pode ser acessada a partir de um método “static”.

## **static**

Ai vai sua primeira lição sobre orientação a objetos, classes estáticas “static” só podem acessar variáveis que também sejam estáticas.

Quando declaramos variáveis static em uma classe ela será exatamente a mesma para todos os objetos desta classe, ou seja, não teremos um tipo diferente em cada objeto, todos os objetos, ao acessarem e modificarem essa variável, acessarão a mesma variável, o mesmo espaço na memória, e a mudança valerá para todos os objetos.

Basicamente não serão criadas várias variáveis para todos os objetos e sim somente uma e todos os objetos usarão a mesma variável.

Então para que possamos acessar a variável “numG” pelo método “main” temos duas alternativas:

Alternativa 1: Retiramos a palavra static do método “main”.





```
1 package aula2;
2
3 public class Aula2 {
4
5     int numG;
6
7     public void main(String[] args) {
8
9         int numL;
10
11         numG=10;
12
13     }
14
15 }
16
```

Alternativa 2: Declaramos a variável “numG” como “static”.

```
1 package aula2;
2
3 public class Aula2 {
4
5     static int numG;
6
7     public static void main(String[] args) {
8
9         int numL;
10
11         numG=10;
12
13     }
14
15 }
16
```

## Variáveis com mesmo nome

Um fator importante a ser destacado é que não podemos declarar duas variáveis com mesmo nome no mesmo escopo.

Veja no exemplo a seguir que existem um alerta de erro na linha 8, por causa da duplicidade da variável “num”.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         int num;
8         int num;
9
10    }
11
12 }
```

No exemplo a seguir também temos duas variáveis com mesmo nome “num” mas não existe erro neste código, isso porque as variáveis estão em escopos diferentes.



```
CursoJavaCFB.java x
Código-Fonte  Histórico
1  package cursojavacfb;
2
3  public class CursoJavaCFB {
4
5      public static void main(String[] args){
6
7          int num;
8
9      }
10
11     public void aula(){
12         int num;
13     }
14
15 }
```

No código acima a variável “num” da linha 7 pertence ao método “main” e a variável “num” da linha 12 pertence ao método “aula”.



Aprenda mais sobre variáveis no Canal Fessor Bruno nos links:

<https://goo.gl/Hf1lvx>

<https://goo.gl/2YxMwj>

<https://goo.gl/Dv9rNG>

## Operações com variáveis

Usando o programa anterior “Aula2” vamos entender como realizar operações básicas com variáveis, veja as alterações a seguir.

Na linha 9 inserimos o valor 5 na variável “numL”.

Na linha 10 criamos a variável “soma”.

Na linha 12 adicionamos o valor 10 à variável “numG”.

Na linha 14 realizamos um soma simples das duas variáveis e armazenamos esta soma na variável “soma”.

Na linha 16 imprimimos o resultado da soma no console.



```
1 package aula2;
2
3 public class Aula2 {
4
5     static int numG;
6
7     public static void main(String[] args) {
8
9         int numL=5;
10        int soma;
11
12        numG=10;
13
14        soma=numL+numG;
15
16        System.out.println(soma);
17    }
18 }
19
20 }
21
```

Ao rodarmos o programa o resultado será 15 como na ilustração a seguir.

```
Saída - Aula2 (run) x
run:
15
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## Operadores

Neste capítulo vou mostrar os principais operadores que podemos utilizar em Java.

Operadores matemáticos		
Operador	Descrição	Exemplo
+	Soma, retorna o resultado da soma	A + B
-	Subtração, retorna o resultado da subtração	A – B
*	Multiplicação, retorna o resultado da multiplicação	A * B
/	Divisão, retorna o resultado da divisão	A / B
%	MOD, retorna resto da divisão	A % B

```
int num1=10;
int num2=5;
int res;

res = num1+num2;

System.out.println(res); //Imprime o valor 15 no console
```

```
int num1=10;
int num2=5;
int res;

res = num1*num2;

System.out.println(res); //Imprime o valor 50 no console
```



Operadores de incremento/decremento		
Operador	Descrição	Exemplo
++	Soma um ao valor atual do elemento	A++
--	Subtrai um ao valor atual do elemento	A--
+= X	Soma o valor de X ao elemento	A += 10
-= X	Subtrai o valor de X ao elemento	A -= 10
*= X	Multiplica o valor de X ao elemento	A *= 2

```
int num1=10;

num1++;

System.out.println(num1); //Imprime o valor 11 no console
```

```
int num1=10;

num1+=5;

System.out.println(num1); //Imprime o valor 15 no console
```

```
int num1=10;

num1*=2;

System.out.println(num1); //Imprime o valor 20 no console
```

Operador de atribuição		
Operador	Descrição	Exemplo
=	Operador principal para atribuir/insérer um valor a um elemento	A = 20 ou A = B

```
int num1;

num1=10;

System.out.println(num1); //Imprime o valor 10 no console
```

```
int num1=10;

System.out.println(num1); //Imprime o valor 10 no console
```

Operadores relacionais / comparação		
Operador	Descrição	Exemplo
>	Maior, compara se um elemento tem o valor maior que outro	A > B
<	Menor, compara se um elemento tem o valor menor que outro	A < B
>=	Maior ou igual, compara se um elemento é maior ou igual a outro	A >= B
<=	Menor ou igual, compara se um elemento é menor ou igual a outro	A <= B
==	Igualdade, compara se um elemento tem o valor igual a outro	A == B
!=	Diferença, compara se um elemento tem o valor diferente de outro	A != B
OBS: Estes operadores retornar true (verdadeiro, 1) ou false (falso, 0), por exemplo na comparação A>B, se A for maior que B retorna true/verdadeiro/1, se for menor retorna false/falso/0.		



```
int num1=10;
int num2=5;

System.out.println(num1 > num2); //Imprime o valor TRUE no console
```

```
int num1=10;
int num2=5;

System.out.println(num1 < num2); //Imprime o valor FALSE no console
```

Operadores de shift		
Operador	Descrição	Exemplo
>>	Deslocamento à direita dos bits	A >> 1
<<	Deslocamento à esquerda dos bits	A << 1

```
int num1=10;
int res;

res = num1>>1;

System.out.println(res); //Imprime o valor 5 no console
```

```
int num1=10;
int res;

res = num1<<1;

System.out.println(res); //Imprime o valor 20 no console
```

Operadores AND(&&) e OR(  )		
Operador	Descrição	Exemplo
&&	E, retorna true/verdadeiro/1 se todos as comparações forem verdadeiras	(A > B) && (C < D)
	OU, retorna true/verdadeiro/1 se pelo menos uma das comparações forem verdadeiras	(A > B)    (C < D)

```
int num1=10;
int num2=5;
boolean res;

res = (num1 > num2) && (num2 < num1);

System.out.println(res); //Imprime TRUE no console
```

```
int num1=5;
int num2=10;
boolean res;

res = (num1 > num2) && (num2 < num1);

System.out.println(res); //Imprime FALSE no console
```

## Pré e Pós incremento



Operador	Descrição	Exemplo
X++	Pós incremento, usa o valor de X e só depois faz o incremento	Y = X++
++X	Pré incremento, faz o incremento e então usa o novo valor de X	Y = ++X

```
int num1=10;
int res;


res = num1++;

System.out.println(res); //Imprime o valor 10 no console, mas a variável num1 passa a valer 11
//Isso pq foi atribuído o valor inicial de num1 em res e depois incrementado num1
```

```
int num1=10;
int res;

res = ++num1;

System.out.println(res); //Imprime o valor 11 no console
```

 Aprenda mais sobre variáveis no Canal Fessor Bruno nos links:  
<https://goo.gl/5HcgE4>  
<https://goo.gl/4oATwK>

## Constantes

Constantes têm uma definição um pouco diferente de variáveis, ao contrário das variáveis, as constantes têm seu valor definido de forma fixa e não podem ser alterados. Outro detalhe importante das constantes é que devemos indicar seu valor no momento da declaração, no corpo do código este valor não pode ser alterado.

Para declarar um constante em Java basta usar a palavra chave “final”, vamos a um exemplo.

**OBS:** Vou criar um novo projeto chamado CursoJavaCFB no qual será utilizado para os próximos exemplos ao longo do curso.

Vamos a um programa de exemplo onde iremos declarar três constantes PI, CANAL e VERSAO e no programa iremos utilizar estas constantes.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     static final float PI=3.14159265f;
6     static final String CANAL="Canal Fessor Bruno";
7     static final int VERSAO=1;
8
9     public static void main(String[] args) {
10
11         float raio = 10f;
12         float comp = raio * 2 * PI;
13         float area = (raio * raio) * PI;
14
15         System.out.println("Raio do círculo.....: " + raio);
16         System.out.println("Comprimento do círculo: " + comp);
17         System.out.println("Área do círculo.....: " + area);
18         System.out.println("CFB: " + CANAL);
19         System.out.println("Versão do curso: " + VERSAO);
20
21     }
22
23 }
24
```

Saída - CursoJavaCFB (run) X

```
run:
Raio do círculo.....: 10.0
Comprimento do círculo: 62.831856
Área do círculo.....: 314.15927
CFB: Canal Fessor Bruno
Versão do curso: 1
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Nas linhas 11, 12 e 13 utilizamos a constante PI para calcular o comprimento e a área de um determinado círculo, nas linhas 15 a 19 simplesmente imprimimos no console os valores calculados e as constantes.

Note que na ilustração anterior já está a janela de saída do console.

 Aprenda mais sobre constantes no Canal Fessor Bruno no link:  
<https://goo.gl/DoKQtq>

## Operações de casting

Basicamente podemos entender “casting” em programação, por uma operação de conversão, a tradução direta é “moldagem”, vamos usar uma operação de casting sempre que precisarmos converter um valor em outro.

Para realizar um casting de forma simples basta informar o tipo que deseja converter, veja a sintaxe.

tipo1 = (casting)tipo2;

A tabela a seguir ilustra bem as situações onde são necessárias o uso de casting e onde as conversões são implícitas (autopromoção) onde não há necessidade de casting.

DE PARA	byte	short	char	int	long	float	double
byte	-	(byte)	(byte)	(byte)	(byte)	(byte)	(byte)



<b>short</b>	implícita	-	(short)	(short)	(short)	(short)	(short)
<b>char</b>	(char)	(char)	-	(char)	(char)	(char)	(char)
<b>int</b>	implícita	implícita	implícita	-	(int)	(int)	(int)
<b>long</b>	implícita	implícita	implícita	implícita	-	(long)	(long)
<b>float</b>	implícita	implícita	implícita	implícita	implícita	-	(float)
<b>double</b>	implícita	implícita	implícita	implícita	implícita	implícita	-

Vamos ver um código de exemplo de casting e conversões implícitas.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         byte num_b;
8         short num_s;
9         char num_c;
10        int num_i;
11        long num_l;
12        float num_f;
13        double num_d;
14
15        num_b = 5;
16
17        num_l = num_b;
18        num_s = (short)num_l;
19        num_d = num_s;
20        num_l = num_s;
21        num_f = (float)num_d;
22        num_i = (int)num_f;
23
24        System.out.println("Long: " + num_l + " - Short: " + num_s + " - Double: " + num_d + " - Long: " + num_l + " - Float: " + num_f + " - Int: " + num_i);
25    }
26 }
27
28 }
```

Saida - CursoJavaCFB (run) X

```
run:
Long: 5 - Short: 5 - Double: 5.0 - Long: 5 - Float: 5.0 - Int: 5
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)
```

Veja que nas linhas 18, 21 e 22 foram necessárias as operações de casting.

## Comentários

Comentários servem para inserir textos não compiláveis em nosso código, ou seja um conteúdo em nosso código que esteja em forma de comentário será ignorado no processo de compilação, podemos usar para documentar nosso código, inserir notas ou lembretes.

Existem dois modos de comentários que podemos utilizar em nossos códigos, veja.

### Comentário de linha única

Para comentar somente uma linha em nosso código usamos //, veja o exemplo.

```
1 package exemplo;
2
3 public class Exemplo {
4
5     public static void main(String[] args) {
6         //Esta linha de código é um comentário e não será compilada
7     }
8
9 }
10
```





No código acima a linha 6 é um comentário e não será compilado.

Podemos comentar qualquer linha em nosso código, no exemplo a seguir temos na linha 6 um comentário informativo e na linha 10 temos uma linha de código em comentário, ambas as linhas 6 e 10 não serão compiladas.

```
1 package exemplo;
2
3 public class Exemplo {
4
5     public static void main(String[] args) {
6         //Declaração de Variáveis
7         int num1=10;
8         int num2=20;
9
10        //System.out.println(num1);
11    }
12
13 }
```

### Comentário multilinha ou em bloco

Neste aco usamos /\* para indicar o início do comentário e \*/ para indicar o fim, veja o exemplo.

```
1 package exemplo;
2
3 public class Exemplo {
4
5     public static void main(String[] args) {
6         //Declaração de Variáveis
7         int num1=10;
8         int num2=20;
9         int num3=30;
10        int num4=40;
11
12        /*
13        System.out.println(num1);
14        System.out.println(num2);
15        System.out.println(num3);
16        System.out.println(num4);
17        */
18    }
19
20 }
```

No código anterior além da linha 6 colocamos em comentário também as linhas 13 a 16, mas note que não precisamos comentar linha a linha, simplesmente indicar o início e o fim do comentário.

## IF

O comando IF é um dos comandos mais importantes no controle de execução de ações em um programa. Basicamente é um comando que realizar um teste lógico e se o resultado deste teste for verdadeiro o comando IF executa um determinado bloco de comandos ou não.

Vamos a um exemplo inicial simples onde iremos criar cinco variáveis inteiras, quatro para notas e uma para soma das notas.

Após somar todas as notas iremos verificar se a soma é maior que 60, se for, iremos imprimir uma mensagem no console indicando que o aluno foi aprovado.

Veja o código e o resultado na janela de saída do console na ilustração a seguir.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int n1,n2,n3,n4,soma;
8
9         n1=20;
10        n2=15;
11        n3=25;
12        n4=5;
13
14        soma=n1+n2+n3+n4;
15
16        if(soma > 60){
17            System.out.println("Aluno APROVADO, nota: " + soma);
18        }
19    }
20 }
21
22 }
```

Saida - CursoJavaCFB (run) X

```
run:
Aluno APROVADO, nota: 65
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que neste caso, se a soma das notas for 60 o aluno será reprovado, mas devemos incluir o valor 60 para aprovação, como resolver este problema? Basta mudar o operador para >= (maior ou igual).

```
if(soma >= 60){
    System.out.println("Aluno APROVADO, nota: " + soma);
}
```

**OBS:** Um detalhe importante sobre o bloco de comandos do IF é que, se existir somente um comando para ser executado pelo IF, não é necessário utilizar chaves {}, assim os códigos a seguir resultam no mesmo fim.

```
if(soma >= 60){
    System.out.println("Aluno APROVADO, nota: " + soma);
}
```

```
if(soma >= 60)
    System.out.println("Aluno APROVADO, nota: " + soma);
```

Somente é obrigatório o uso das {} quando o bloco IF contiver mais de uma linha de comando.

Porém, por segurança no aprendizado, irei usar as chaves mesmo que o IF só tenha uma linha de comando.

## IF ELSE

Um complemento ao comando IF é o ELSE, que é o caso contrário do IF, ou seja, se o resultado da comparação for falso.

No exemplo a seguir iremos alterar o código do IF anterior para adicionar uma mensagem caso o aluno seja reprovado.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int n1,n2,n3,n4,soma;
8
9         n1=20;
10        n2=15;
11        n3=25;
12        n4=5;
13
14        soma=n1+n2+n3+n4;
15
16        if(soma >= 60){
17            System.out.println("Aluno APROVADO, nota: " + soma);
18        }else{
19            System.out.println("Aluno REPROVADO, nota: " + soma);
20        }
21    }
22 }
23
24
```

Saída - CursoJavaCFB (run) ×

```
run:
Aluno APROVADO, nota: 65
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que na linha 18 inserimos um ELSE que executa seu bloco de comandos caso o teste do IF seja falso, com os valores atuais a impressão no console irá informar que o aluno foi APROVADO, vamos alterar os valores e ver o novo resultado.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int n1,n2,n3,n4,soma;
8
9         n1=15;
10        n2=10;
11        n3=20;
12        n4=5;
13
14        soma=n1+n2+n3+n4;
15
16        if(soma >= 60){
17            System.out.println("Aluno APROVADO, nota: " + soma);
18        }else{
19            System.out.println("Aluno REPROVADO, nota: " + soma);
20        }
21    }
22 }
23
24
```

Saída - CursoJavaCFB (run) ×

```
run:
Aluno REPROVADO, nota: 50
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que agora o aluno foi reprovado.

Podemos criar várias condições em uma mesma execução IF, por exemplo, se quisermos incluir a condição de aluno em recuperação, vamos às condições.

- Nota menor que 45 o aluno está reprovado;
- Nota entre 45 3 59 o aluno está em recuperação;
- Nota maior ou igual a 60 o aluno está aprovado.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int n1,n2,n3,n4,soma;
8
9         n1=15;
10        n2=10;
11        n3=20;
12        n4=5;
13
14        soma=n1+n2+n3+n4;
15
16        if(soma < 45){
17            System.out.println("Aluno REPROVADO, nota: " + soma);
18        }else if(soma < 60){
19            System.out.println("Aluno em RECUPERAÇÃO, nota: " + soma);
20        }else{
21            System.out.println("Aluno APROVADO, nota: " + soma);
22        }
23    }
24 }
25
26 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Aluno em RECUPERAÇÃO, nota: 50
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Altere os valores das notas e veja os resultados possíveis.



Aprenda mais sobre a condicional IF ELSE no Canal Fessor Bruno nos links:

<https://goo.gl/mb5bYP>

<https://goo.gl/QihDVG>

## AND(&&), OR(||), NOT(!)

Os operadores AND, OR e NOT auxiliam bastante na construção de expressões lógicas para os testes IF e para os loops WHILE e FOR, vamos ver alguns exemplos com estes operadores.

No primeiro exemplo temos duas variáveis relacionadas à temperatura e ao dia da semana, em seguida iremos verificar estas variáveis para decidir se vou ao clube ou fico em casa.

Para ir ao clube a temperatura deve ser maior que 29 e o dia da semana deve ser igual a domingo.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int temp=40;
8         String dia="domingo";
9
10        if((temp >= 30) && (dia == "domingo")){
11            System.out.println("Vou ao clube");
12        }else{
13            System.out.println("Fico em casa");
14        }
15    }
16 }
17
18
19
```

Saída - CursoJavaCFB (run) X

run:  
Vou ao clube  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

Vamos adicionar mais uma condição, se a temperatura for maior que 30 e for sábado eu irei ao parque.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int temp=40;
8         String dia="domingo";
9
10        if((temp >= 30) && (dia == "domingo")){
11            System.out.println("Vou ao clube");
12        }else if((temp >= 30) && (dia == "sábado")){
13            System.out.println("Vou ao parque");
14        }else{
15            System.out.println("Fico em casa");
16        }
17    }
18 }
19
```

Vamos alterar a condição inicial, se a temperatura for maior que 30 e for sábado ou domingo eu vou ao clube.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int temp=40;
8         String dia="domingo";
9
10        if((temp >= 30) && ((dia == "domingo") || (dia == "sábado"))){
11            System.out.println("Vou ao clube");
12        }else{
13            System.out.println("Fico em casa");
14        }
15    }
16 }
17
18
19
```



O operador NOT (!) faz uma negação invertendo o resultado do teste lógico, ou seja, se o teste for verdadeiro o NOT retorna falso e se for falso o NOT retorna verdadeiro.

No exemplo a seguir temos a variável dia com o valor "sol" iremos testar esta variável e SE NÃO for dia de sol iremos ao cinema, caso contrário vou ao clube.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         String dia="sol";
8
9         if(!(dia == "sol")){
10             System.out.println("Vou ao cinema");
11         }else{
12             System.out.println("Vou ao clube");
13         }
14     }
15 }
16
17 }
```

No código acima o teste lógico retorna "true" pois a variável dia tem o valor igual a "sol", porém, o operador NOT inverte seu valor para "false" resultado na seguinte leitura:

IF(!(dia == "sol"))  
SE(NÃO(DIA de SOL))  
SE NÃO for DIA de SOL

Aprenda mais sobre os operadores AND, OR e NOT no Canal Fessor Bruno no link:  
<https://goo.gl/GvyNYQ>

## Ternário

O operador condicional ternário pode substituir o IF ELSE em alguns casos mais simples a sintaxe é a seguinte.

(expressão) ? se\_verdadeiro : se\_falso;

Veja o exemplo a seguir onde testamos a variável dia para saber se vou ao clube ou ao cinema.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         String dia="sol";
8
9         System.out.println((dia == "sol") ? "Vou ao clube" : "Vou ao cinema");
10     }
11 }
12
13 }
14
```

Saída - CursoJavaCFB (run) x

run:  
Vou ao clube  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)



Aprenda mais sobre o operador ternário no Canal Fessor Bruno no link:

<https://goo.gl/DF4iLh>

## Switch

O comando switch é “semelhante” ao comando “IF”, onde dependendo do resultado de um determinado teste será executado um determinado bloco de comandos.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int opc=1;
8
9         switch(opc){
10             case 1:{
11                 System.out.println("Opção UM selecionada");
12                 break;
13             }
14             case 2:{
15                 System.out.println("Opção DOIS selecionada");
16                 break;
17             }
18             case 3:{
19                 System.out.println("Opção TRÊS selecionada");
20                 break;
21             }
22             default:{
23                 System.out.println("Opção inválida");
24                 break;
25             }
26         }
27     }
28 }
29
30 }
```

Saida - CursoJavaCFB (run) X

```
run:
Opção UM selecionada
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```


Podemos aninhar os cases quando mais de uma opção resultar no mesmo valor, veja o exemplo.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int opc=5;
8
9         switch(opc){
10             case 1:
11             case 2:
12             case 3:{
13                 System.out.println("Opção UM, DOIS ou TRÊS selecionada");
14                 break;
15             }
16             case 4:
17             case 5:
18             case 6:{
19                 System.out.println("Opção QUATRO, CINCO ou SEIS selecionada");
20                 break;
21             }
22             default:{
23                 System.out.println("Opção inválida");
24                 break;
25             }
26         }
27     }
28 }
29
30 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Opção QUATRO, CINCO ou SEIS selecionada
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

 Aprenda mais sobre o comando SWITCH no Canal Fessor Bruno no link:  
<https://goo.gl/QVTFby>

## While

O comando while é um comando de loop, isso significa que ele executa um bloco de comandos enquanto uma determinada condição for satisfeita, observe a seguir a sintaxe padrão do comando while.

```
while (condição) {
    //comandos
}
```

A tradução de while é “enquanto”.

Devemos observar ainda alguns detalhes importantes na estrutura do while, precisamos preparar a condição antes da entrada do while e dentro do bloco de comandos devemos trabalhar na condição de parada do loop, pois corre-se o risco do comando entrar em um loop infinito e levar a parada do programa.

```
inicia contador
while(condição){
    //Comandos
    Incrementa/decrementa contador
}
```





Em nosso primeiro exemplo vamos criar um programa que escreva o texto “Canal Fessor Bruno” 10 vezes.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int cont=0;
8
9         while(cont < 10){
10             System.out.println("Canal Fessor Bruno");
11             cont++;
12         }
13     }
14 }
15 }
```

Saída - CursoJavaCFB (run) X

```
run:
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que na linha 7 iniciamos o contador.

Na linha 9 preparamos a condição para execução do while, enquanto o valor da variável cont for menor que 10.

Na linha 11 temos o incremento do contador, já que iniciamos em zero e a condição para parada é quando o cont for maior ou igual a 10, precisamos incrementar, aumentar de um em um o valor de cont a cada iteração do while.

Quando o valor da variável cont foi igual a 10 a condição não foi mais satisfeita e o loop foi encerrado continuando assim a execução normal do programa, como não exista mais nada a ser feito o programa foi finalizado.

Vamos alterar nosso programa para imprimir o valor de cont e não mais a string.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int cont=0;
8
9         while(cont < 10){
10             System.out.println(cont);
11             cont++;
12         }
13     }
14 }
15 }
```

Saída - CursoJavaCFB (run) X

```
run:
0
1
2
3
4
5
6
7
8
9
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



Veja que agora foi impresso uma contagem de 0 a 9. Como podemos alterar esta contagem para que seja de 1 a 10?

Basta somar um ao valor de cont no momento da impressão como vemos no exemplo na linha 10.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int cont=0;
8
9         while(cont < 10){
10             System.out.println(cont+1);
11             cont++;
12         }
13     }
14 }
15 }
```

Podemos usar o loop while quando não sabemos a quantidade exata de execuções/iterações do bloco de comandos, veja o próximo exemplo.

Iniciamos a variável cont com valor zero e no loop while a condição é “enquanto cont for diferente de dez), dentro do while usamos o método random da biblioteca matemática para gerar um número aleatório entre 0 e 10, como não sabemos quando o número 10 será gerado, não temos certeza sobre o número de iterações que este while irá executar.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         long cont=0;
8
9         while(cont != 10){
10             cont=Math.round(Math.random()*10);
11             System.out.println(cont);
12         }
13     }
14 }
15 }
```

Saída - CursoJavaCFB (run) ×

run:  
8  
6  
3  
0  
3  
9  
9  
7  
10  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

A cada vez que rodar este código os números impressos serão diferentes, sempre terminando no 10.

O método round foi chamado simplesmente para arredondar o valor gerado pelo método random.



Aprenda mais sobre o comando while no Canal Fessor Bruno no link:

<https://goo.gl/02LzhE>



## Do while

O comando `do while` é semelhante ao comando `while`, com uma diferença importante, enquanto o comando `while` testa a condição no início o comando `do while` testa a condição no final.

Veja a diferença.

```
while(condição){
    //comandos
}
```

```
do{
    //comandos
}while(condição);
```

Isso é uma diferença significativa entre os comandos, imagine uma sigtação em que a condição já tenha sido satisfeita na entrada do loop `while`, assim, os comandos não seriam executados nenhuma vez, veja o exemplo.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int cont=10;
8
9         while(cont < 10){
10             System.out.println("Canal Fessor Bruno");
11             cont++;
12         }
13     }
14 }
15
```

Saída - CursoJavaCFB (run) X

run:  
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)

Note que na janela de saída não foi impresso nada, pois, a condição já havia sido satisfeita antes de entrar no `while`.

Vamos trocar pelo loop `do while` e ver o resultado.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int cont=10;
8
9         do{
10             System.out.println("Canal Fessor Bruno");
11             cont++;
12         }while(cont < 10);
13     }
14 }
15
```

Saída - CursoJavaCFB (run) X

run:  
Canal Fessor Bruno  
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)



Note que agora existe pelo menos uma impressão, ou seja, os comandos do loop do while foram executados pelo menos uma vez.

Então lembre-se, o loop while faz o teste condicional no início e o loop do while faz o teste condicional no final.



Aprenda mais sobre o comando do while no Canal Fessor Bruno no link:

<https://goo.gl/JgQZMQ>

## For

O loop FOR tem uma sintaxe diferenciada do loop while, usamos o loop for quando podemos prever quantas iterações serão executadas.

Veja a sintaxe do comando for.

```
for(inicia contador; condição de execução; incremento/decremento contador){  
    //Comandos  
}
```

Seguindo esta sintaxe vamos criar um loop for que irá executar 10 iterações.

Vamos iniciar o contador com zero e a condição de execução será enquanto o valor de cont for menor que 10, iremos incrementar a variável cont de um em um.

```
1 package cursojavacfb;  
2  
3 public class CursoJavaCFB {  
4  
5     public static void main(String[] args) {  
6  
7         int cont;  
8  
9         for(cont=0; cont<10; cont++){  
10             System.out.println("Canal Fessor Bruno");  
11         }  
12     }  
13  
14 }
```

Saída - CursoJavaCFB (run) ×

```
run:  
Canal Fessor Bruno  
Canal Fessor Bruno  
Canal Fessor Bruno  
Canal Fessor Bruno  
Canal Fessor Bruno  
Canal Fessor Bruno  
Canal Fessor Bruno  
Canal Fessor Bruno  
Canal Fessor Bruno  
Canal Fessor Bruno  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Vamos fazer uma pequena alteração em nosso programa para imprimir a contagem de 0 a 9, basta imprimir o valor da variável cont.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int cont;
8
9         for(cont=0; cont<10; cont++){
10             System.out.println(cont);
11         }
12     }
13
14 }
```

Saída - CursoJavaCFB (run) ×

```
run:
0
1
2
3
4
5
6
7
8
9
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Mais uma pequena alteração, já usamos o método “random” anteriormente, vamos usá-lo em nosso programa para gerar dez números aleatórios entre zero e dez.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int cont;
8
9         for(cont=0; cont<10; cont++){
10             System.out.println(Math.random()*10);
11         }
12     }
13
14 }
```

Saída - CursoJavaCFB (run) ×

```
run:
7.187244075115685
3.286137187437548
7.586956917593808
7.34829194203768
7.3028910300094445
9.107130353767879
7.985579669932951
5.282183514079193
1.6640248980911065
7.230148170231111
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

OBS: Podemos usar o método “round” como anteriormente para arredondar os números caso queira.



Aprenda mais sobre o comando for no Canal Fessor Bruno no link:

<https://goo.gl/L5ngww>

## *Arrays unidimensionais (vetor)*

Um Array é basicamente um coleção de variáveis do mesmo tipo, todas na mesma estrutura, cada “variável” do array é chamada de elemento e podemos referenciar os elementos com um índice.

Por exemplo, um conjunto de variáveis que representam a pontuação de 5 jogadores.

```
int j1,j2,j3,j4,j5;
```

Ao invés de criarmos 5 variáveis diferentes, podemos criar um array com 5 posições/elementos, cada elemento irá armazenar a pontuação de um determinado jogador.

Para declarar um array em Java basta usar os colchetes [ ] após o tipo de dados, como no exemplo a seguir.

```
int[] pontos;
```

Para alocar memória para nosso array e podermos usar, precisamos usar “new” e informar a quantidade de elementos do nosso array.

```
int[] pontos = new int[5];
```

Com nosso array devidamente declarado e pronto para uso podemos inserir valores nas posições/elementos do nosso array, basta informar a posição que deseja armazenar o valor.

```
pontos[0]=5;  
pontos[1]=7;  
pontos[2]=3;  
pontos[3]=9;  
pontos[4]=4;
```

Note que o primeiro elemento do array é o elemento de índice zero e não um, acostume-se com isso para não ter problemas futuros.

Agora vamos imprimir os valores de cada uma das posições no console.

```
System.out.println(pontos[0]);  
System.out.println(pontos[1]);  
System.out.println(pontos[2]);  
System.out.println(pontos[3]);  
System.out.println(pontos[4]);
```

Viu como é simples trabalhar com arrays, vamos montar o programa com os comandos anteriores e rodar para ver o resultado.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int[] pontos = new int[5];
8
9         pontos[0]=5;
10        pontos[1]=7;
11        pontos[2]=3;
12        pontos[3]=9;
13        pontos[4]=4;
14
15        System.out.println(pontos[0]);
16        System.out.println(pontos[1]);
17        System.out.println(pontos[2]);
18        System.out.println(pontos[3]);
19        System.out.println(pontos[4]);
20
21    }
22
23 }
```

Saída - CursoJavaCFB (run) ×

```
run:
5
7
3
9
4
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Para facilitar podemos usar o loop for para ler ou preencher nosso array. Vamos criar um for para imprimir os elementos do nosso array pontos.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int[] pontos = new int[5];
7         int i;
8
9
10        pontos[0]=5;
11        pontos[1]=7;
12        pontos[2]=3;
13        pontos[3]=9;
14        pontos[4]=4;
15
16        for(i=0; i<5; i++){
17            System.out.println(pontos[i]);
18        }
19
20    }
21
22 }
```

Saída - CursoJavaCFB (run) ×

```
run:
5
7
3
9
4
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



Note que a variável “i” que é o contador do for iniciou em zero, pois é o primeiro elemento do array está na posição zero.

Note também que na linha 17, no comando de impressão usamos a variável “i” como índice para a posição no array.

## *length (constante)*

A constante length retorna o tamanho (quantidade de posições/elementos) do array, bastante útil sempre que for preciso saber a quantidade de elementos de um array.

Veja o exemplo a seguir onde criamos um array de tamanho 10 e em seguida usamos a constante length para obter o tamanho deste array.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         int num[] = new int[10];
8
9         int tam=num.length; //Constante
10
11         System.out.println("Tamanho do Array: " + tam);
12
13     }
14
15 }
```

Saida - CursoJavaCFB (run) ×

```
run:
Tamanho do Array: 10
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Podemos usar em conjunto com o loop for para percorrer um array, veja o exemplo.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         int num[] = new int[10];
8
9         int tam=num.length; //Constante
10
11         System.out.println("Tamanho do Array: " + tam);
12
13         for(int i=0; i<num.length; i++){
14             num[i]=0;
15         }
16
17     }
18
19 }
```

Note que o FOR na linha 13 usa “length” para indicar o tamanho do array, este loop FOR percorre todo array preenchendo todas as posições com o valor zero.





## Instanciando e inserindo elementos no array

Podemos inserir os elementos do array na mesma linha de código em que o instanciamos, no exemplo a seguir que mostro este procedimento note que não informamos o tamanho do array, que será calculado automaticamente pelo número de elementos inserido.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         int[] cfb = new int[]{10,20,30,40,50};
8         int tam=cfb.length;
9
10        for(int i=0;i<tam;i++){
11            System.out.println(cfb[i]);
12        }
13
14    }
15
16 }
17 }
```

Saída - CursoJavaCFB (run) ×

```
run:
10
20
30
40
50
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## Arrays bidimensionais (matriz)

Arrays bidimensionais são arrays basicamente controlados por dois índices, como se tivéssemos um novo array dentro de cada elemento do array anterior. Uma representação simples para um array bidimensional ou matriz é uma tabela, com linhas e colunas.

Array unidimensional, 4 posições/elementos.

--	--	--	--

Array bidimensional, 3 linhas e 4 colunas.


Vamos ao exemplo.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         int num[][] = new int[3][4];
8
9         num[0][0]=0;
10        num[0][1]=0;
11        num[0][2]=0;
12        num[0][3]=0;
13
14        num[1][0]=1;
15        num[1][1]=1;
16        num[1][2]=1;
17        num[1][3]=1;
18
19        num[2][0]=2;
20        num[2][1]=2;
21        num[2][2]=2;
22        num[2][3]=2;
23
24    }
25
26 }
```

No programa anterior criamos uma matriz com 3 linhas e 4 colunas e preenchemos os elementos com alguns valores, graficamente podemos representar esta matriz da seguinte forma.

0	0	0	0
1	1	1	1
2	2	2	2

Vamos inserir um comando loop FOR para imprimir os valores da nossa matriz.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         int num[][] = new int[3][4];
8         int l,c;
9
10        num[0][0]=0;
11        num[0][1]=0;
12        num[0][2]=0;
13        num[0][3]=0;
14
15        num[1][0]=1;
16        num[1][1]=1;
17        num[1][2]=1;
18        num[1][3]=1;
19
20        num[2][0]=2;
21        num[2][1]=2;
22        num[2][2]=2;
23        num[2][3]=2;
24
25        for(l=0;l<3;l++){
26            for(c=0;c<4;c++){
27                System.out.print(num[l][c]+" ");
28            }
29            System.out.println("");
30        }
31    }
32
33 }
```

```
Saída - CursoJavaCFB (run) x
run:
0 0 0 0
1 1 1 1
2 2 2 2
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)
```



Note que como temos um array bidimensional, com dois índices, usamos dois loops FOR, o primeiro para controlar as linhas e o segundo para controlar as colunas.

Para cada iteração do primeiro FOR, ou seja, para cada elemento do primeiro FOR o segundo FOR percorre todas as colunas.

## A classe Arrays

Existe uma classe que se chama Array e nesta classe existem vários métodos prontos para trabalharmos com arrays, vamos aprender alguns deles neste capítulo.

Para usarmos a classe Array e seus métodos precisamos importar a biblioteca Arrays, veja no código básico de exemplo na linha 3.

```
1 package cursojavacfb;
2
3 import java.util.Arrays;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9
10    }
11
12 }
```

## binarySearch

Este método faz uma busca em um array por um valor informado e retorna sua posição.

A sintaxe é simples:

```
Arrays.binarySearch(array, elemento_procurado);
```

```
1 package cursojavacfb;
2
3 import java.util.Arrays;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         int num1[] = new int[5];
10
11         num1[0]=2;
12         num1[1]=5;
13         num1[2]=8;
14         num1[3]=3;
15         num1[4]=9;
16
17         System.out.println(Arrays.binarySearch(num1,8));
18
19     }
20
21 }
```

Saída - CursoJavaCFB (run) ×

```
run:
2
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



No exemplo anterior usamos o método `binarySearch` na linha 17, dentro do método “`println`”, desta maneira, passamos para `binarySearch` o array “`num1`” e o valor que estamos procurando “`8`”, o método retorna a posição 2, indicando que o valor 8 está na posição 2 do vetor `num1`.

Caso o valor que estejamos procurando não exista no array será retornado o valor “`-1`”, veja o exemplo.

```
1 package cursojavacfb;
2
3 import java.util.Arrays;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         int num1[] = new int[5];
10
11         num1[0]=2;
12         num1[1]=5;
13         num1[2]=8;
14         num1[3]=3;
15         num1[4]=9;
16
17         System.out.println(Arrays.binarySearch(num1,1));
18
19     }
20
21 }
```

Saída - CursoJavaCFB (run) ×

```
run:
-1
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## copyOf

Este método copia os elementos de um array para outro array.

A sintaxe é a seguinte:

```
arrayDestino = Arrays.copyOf(array_original, quantidade_de_elementos_a_serem_copiados);
```

Em nosso código de exemplo iremos criar três arrays, preencheremos somente o array “`num1`” e iremos copiar os valores de `num1` para os outros arrays.



```
1 package cursojavacfb;
2
3 import java.util.Arrays;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         int num1[] = new int[10];
10        int num2[] = new int[10];
11        int num3[] = new int[5];
12
13        num1[0]=1;   num1[5]=6;
14        num1[1]=2;   num1[6]=7;
15        num1[2]=3;   num1[7]=8;
16        num1[3]=4;   num1[8]=9;
17        num1[4]=5;   num1[9]=10;
18
19        num2=Arrays.copyOf(num1,10);
20        num3=Arrays.copyOf(num1,5);
21
22        System.out.println("Array num2");
23        for(int i=0; i<num2.length; i++){
24            System.out.print(num2[i] + " ");
25        }
26
27        System.out.println("\nArray num3");
28
29        for(int i=0; i<num3.length; i++){
30            System.out.print(num3[i] + " ");
31        }
32
33    }
34
35 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Array num2
1 2 3 4 5 6 7 8 9 10
Array num3
1 2 3 4 5 CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## copyOfRange

O método “copyOfRange” é semelhante ao método “copyOf” com uma diferença, podemos informar um intervalo, uma faixa de valores a serem copiados, veja a sintaxe.

```
arrayDestino = Arrays.copyOfRange(array_original, valor_inicial, valor_final);
```

Em nosso programa de exemplo iremos copiar os valores que estão da posição 3 até a posição 8.



```
1 package cursojavacfb;
2
3 import java.util.Arrays;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         int num1[] = new int[10];
10        int num2[] = new int[5];
11
12        num1[0]=1;    num1[5]=6;
13        num1[1]=2;    num1[6]=7;
14        num1[2]=3;    num1[7]=8;
15        num1[3]=4;    num1[8]=9;
16        num1[4]=5;    num1[9]=10;
17
18        num2=Arrays.copyOfRange(num1,3,8);
19
20        System.out.println("Array num2");
21        for(int i=0; i<num2.length; i++){
22            System.out.print(num2[i] + " ");
23        }
24
25        System.out.println("\n");
26    }
27
28 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Array num2
4 5 6 7 8
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## equals

Este método faz a comparação de dois arrays, se forem iguais retorna "true" e se não forem retorna "false".

A sintaxe é:

```
Arrays.equals(array_1, array_2);
```

Em nosso código de exemplo iremos copiar os valores do array num1 para o array num2 e realizar o teste com o método "equals" se os arrays forem iguais iremos imprimir a mensagem "Arrays iguais" e se não forem vamos imprimir a mensagem "Arrays diferentes".



```
1 package cursojavacfb;
2
3 import java.util.Arrays;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         int num1[] = new int[10];
10        int num2[] = new int[10];
11
12        num1[0]=1;   num1[5]=6;
13        num1[1]=2;   num1[6]=7;
14        num1[2]=3;   num1[7]=8;
15        num1[3]=4;   num1[8]=9;
16        num1[4]=5;   num1[9]=10;
17
18        num2=Arrays.copyOf(num1,10);
19
20        if(Arrays.equals(num1, num2)){
21            System.out.println("Arrays iguais");
22        }else{
23            System.out.println("Arrays diferentes");
24        }
25
26    }
27
28 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Arrays iguais
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## *fill*

Este método preenche todo o array com o elemento indicado.

Sintaxe:

```
Arrays.fill(array, valor);
```

Em nosso código de exemplo iremos preencher nosso vetor todo com o valor zero.



```
1 package cursojavacfb;
2
3 import java.util.Arrays;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         int num1[] = new int[10];
10
11         Arrays.fill(num1, 0);
12
13         for(int i=0; i<num1.length; i++){
14             System.out.println(num1[i]);
15         }
16
17     }
18
19 }
```

Saida - CursoJavaCFB (run) X

```
run:
0
0
0
0
0
0
0
0
0
0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## sort

Este método ordena todos os elementos do nosso array em ordem crescente.

A sintaxe é bem simples.

```
Arrays.sort(array_a_ser_ordenado);
```

Em nosso código de exemplo iremos criar um programa que irá preencher o array com valores aleatórios e sem seguida na linha 15 irá ordenar estes valores.





```
1 package cursojavacfb;
2
3 import java.util.Arrays;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         int num1[] = new int[10];
10
11         for(int i=0; i<num1.length; i++){
12             num1[i]=(int) Math.round(Math.random()*10);
13         }
14
15         Arrays.sort(num1);
16
17         for(int i=0; i<num1.length; i++){
18             System.out.println(num1[i]);
19         }
20
21     }
22
23 }
```

Saída - CursoJavaCFB (run) x

```
run:
0
0
2
3
3
4
6
8
9
9
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

OBS: como estamos usando o método “random” a cada vez que rodar seu programa os valores serão diferentes.

## *Lendo valores pelo teclado*

Assim como temos o comando “print” para imprimir dados no console, temos também algumas formas de ler dados do teclado e armazenar em variáveis.

Vamos ver neste capítulo duas formas simples.

### *Scanner*

A primeira maneira que iremos aprender é usando a classe scanner, basta criar um objeto desta classe e usar um dos métodos de leitura disponíveis.

Para que possamos utilizar a classe “Scanner” precisamos importar uma biblioteca que está no pacote “java.util”.

Veja o exemplo a seguir com destaque na linha 3 para a importação da biblioteca “Scanner”.



```
1 package cursojavacfb;
2
3 import java.util.Scanner;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Scanner sc=new Scanner(System.in);
10
11         int num;
12
13         num=sc.nextInt();
14
15         System.out.println("O valor digitado foi: " + num);
16
17     }
18
19 }
20
```

Saída - CursoJavaCFB (run) ×

```
run:
10
O valor digitado foi: 10
CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)
```

OBS: Ao rodar o programa, clique na janela de saída do console para poder digitar o valor.

Na linha 13 usamos o método “nextInt()” pois a variável “num” que iremos armazenar o valor é do tipo “int”.

Vamos alterar nosso programa para ler um nome e armazenar em uma variável do tipo String e um valor inteiro armazenando na variável de tipo respectivo.



```
1 package cursojavacfb;
2
3 import java.util.Scanner;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Scanner sc=new Scanner(System.in);
10
11         String nome;
12         int anoNasc;
13
14         System.out.print("Digite seu nome: ");
15         nome=sc.nextLine();
16
17         System.out.print("Informe o ano do seu nascimento: ");
18         anoNasc=sc.nextInt();
19
20         System.out.println("Nome digitado: " + nome);
21         System.out.println("Ano de nascimento: " + anoNasc);
22
23     }
24
25 }
```

Saída - CursoJavaCFB (run) X

```
run:
Digite seu nome: Bruno
Informe o ano do seu nascimento: 1978
Nome digitado: Bruno
Ano de nascimento: 1978
CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)
```

Note que para a leitura do texto/String usamos o método “nextLine” e para a leitura do inteiro usamos o método “nextInt”.

## BufferedReader

A segunda forma que vamos aprender ler dados do teclado é usando o classe “BufferedReader” que é um procedimento um pouco mais trabalhoso.



```
1 package cursojavacfb;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.InputStreamReader;
6
7 public class CursoJavaCFB {
8
9     public static void main(String[] args){
10
11         BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
12
13         String nome = null;
14         int anoNasc = 0;
15
16         System.out.print("Digite seu nome: ");
17         try {
18             nome=br.readLine();
19         } catch (IOException erro) {
20             System.out.print("Erro: " + erro);
21         }
22
23         System.out.print("Informe o ano do seu nascimento: ");
24         try {
25             anoNasc=Integer.parseInt(br.readLine());
26         } catch (IOException erro) {
27             System.out.print("Erro: " + erro);
28         }
29
30         System.out.println("Nome digitado: " + nome);
31         System.out.println("Ano de nascimento: " + anoNasc);
32
33     }
34
35 }
```

Saída - CursoJavaCFB (run) X

```
run:
Digite seu nome: Bruno
Informe o ano do seu nascimento: 1978
Nome digitado: Bruno
Ano de nascimento: 1978
CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)
```

Veja que para este procedimento precisamos importar três bibliotecas nas linhas 3,4 e 5.

Na linha 11 criamos o objeto “br” do tipo “BufferedReader” para usarmos o método para leitura do teclado.

Na linha 13 precisamos inicializar a variável com null devido ao bloco “try catch” onde usamos a variável.

Sobre “try catch” irei dar mais detalhes sobre isso mais adiante.

Na linha 25 precisamos de uma conversão de tipos (typecast) para converter de String para int, já que estamos lendo uma String do teclado e armazenando em uma variável do tipo int.

## Trabalhando com a classe Calendar

A classe Calendar substitui a classe Date, esta classe possui métodos interessantes para trabalharmos com datas e horas em nossos programas.

Precisaremos importar a biblioteca “Calendar” que está no pacote “java.util”, em seguida temos de declarar e instanciar um objeto do tipo Calendar, assim podemos usar os métodos para manipulação de datas e horas.



Vamos a um exemplo simples.

```
1 package cursojavacfb;
2
3 import java.util.Calendar;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Calendar data = Calendar.getInstance();
10
11         System.out.println("Hoje é dia: " + data.get(Calendar.DAY_OF_MONTH));
12         System.out.println("Mês: " + data.get(Calendar.MONTH));
13         System.out.println("O ano atual é: " + data.get(Calendar.YEAR));
14         System.out.println("O dia da semana é: " + data.get(Calendar.DAY_OF_WEEK));
15         System.out.println("Dia do ano: " + data.get(Calendar.DAY_OF_YEAR));
16
17         System.out.println("Hora: " + data.get(Calendar.HOUR) + " " + data.get(Calendar.AM_PM));
18         System.out.println("Hora: " + data.get(Calendar.HOUR_OF_DAY));
19         System.out.println("Minuto: " + data.get(Calendar.MINUTE));
20         System.out.println("Segundo: " + data.get(Calendar.SECOND));
21
22     }
23
24 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Hoje é dia: 30
Mês: 3
O ano atual é: 2016
O dia da semana é: 7
Dia do ano: 121
Hora: 8 1
Hora: 20
Minuto: 52
Segundo: 12
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

A primeira coisa que fizemos foi importar a biblioteca Calendar na linha 3.

Na linha 9 declaramos e instanciamos um objeto do tipo Calendar com nome “data”.

Nas linhas 11 a 20 usamos algumas constantes básicas da classe Calendar, um detalhe importante da constante “MONTH” é que o primeiro mês do ano “Janeiro” é representado pelo valor ZERO, então o valor 3 mostrado no exemplo anterior representa o mês 4 (Abril).

Outro detalhe importante é sobre a constante “AM\_PM”, para o valor “AM” é retornado o valor ZERO e para o valor “PM” é retornado o valor 1.

Vamos escrever a data completa, vamos utilizar vetores para nos auxiliar no momento da impressão.



```
1 package cursojavacfb;
2
3 import java.util.Calendar;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Calendar data = Calendar.getInstance();
10
11         String[] semana = new String[8];
12         String[] meses = new String[12];
13
14         semana[0]=null; semana[1]="Domingo"; semana[2]="Segunda-Feira"; semana[3]="Terça-Feira";
15         semana[4]="Quarta-Feira"; semana[5]="Quinta-Feira"; semana[6]="Sexta-Feira"; semana[7]="Sábado";
16
17         meses[0]="Janeiro"; meses[1]="Fevereiro"; meses[2]="Março"; meses[3]="Abril"; meses[4]="Maio"; meses[5]="Junho";
18         meses[6]="Julho"; meses[7]="Agosto"; meses[8]="Setembro"; meses[9]="Outubro"; meses[10]="Novembro"; meses[11]="Dezembro";
19
20         System.out.print("Belo Horizonte, " + semana[data.get(Calendar.DAY_OF_WEEK)]);
21         System.out.print(" " + data.get(Calendar.DAY_OF_MONTH));
22         System.out.print(" de " + meses[data.get(Calendar.MONTH)]);
23         System.out.println(" de " + data.get(Calendar.YEAR));
24
25         System.out.print("Horário local: " + data.get(Calendar.HOUR_OF_DAY));
26         System.out.print(": " + data.get(Calendar.MINUTE));
27         System.out.println(": " + data.get(Calendar.SECOND));
28
29     }
30
31 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Belo Horizonte, Sábado 30 de Abril de 2016
Horário local: 21:19:7
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## try catch finally – Tratamento de Exceções

A rotina try catch finally de tratamento de exceções é uma rotina basicamente para tratamento de erros, praticamente é uma rotina básica em todas as linguagens orientadas a objetos, iremos usar sempre que uma rotina for passível de erro, veja a sintaxe completa a seguir.

```
try{
    //Comandos a serem executados
}catch(exceção){
    //Rotina a ser executada em caso de erro
}finally{
    //Rotina opcional a ser executada so final de todo o procedimento
}
```

Basicamente dentro do bloco “try” iremos inserir os comandos a serem executados que podem causar algum erro/exceção.

Caso haja algum problema com o código do bloco “try” a execução é passada para o bloco “catch” e seus comandos serão executados.

O bloco “finally” não é obrigatório, caso esteja presente, seus comandos serão executado ao final de toda sequência “try catch”.

Vamos a um exemplo.



```
1 package cursojavacfb;
2
3 import java.util.Scanner;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Scanner sc = new Scanner(System.in);
10
11         int num=0;
12
13         try{
14             num=sc.nextInt();
15             System.out.println("Valor válido: " + num);
16         }catch(Exception erro){
17             System.out.println("Valor Inválido");
18             System.out.println("Erro: " + erro);
19         }
20
21     }
22
23 }
```

Saída - CursoJavaCFB (run) X

```
run:
b
Valor Inválido
Erro: java.util.InputMismatchException
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Em nosso programa criamos uma variável do tipo “int” com nome “num”.

A execução normal do nosso programa seria ler um valor do teclado (inteiro) e imprimir este valor no console.

Mas note na janela de console que o valor que digitei foi a letra “b”, gerando assim um erro, pois, a variável num é do tipo “int”.

Neste caso a execução é interrompida no bloco “try” e passa ao bloco “catch” que executa seus dois comandos de impressão.

Se inserirmos um valor inteiro válido na janela de saída teremos simplesmente a execução do bloco “try” como podemos ver na ilustração a seguir com o valor 10.

Saída - CursoJavaCFB (run) X

```
run:
10
Valor válido: 10
CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)
```

Vamos adiciona o blobo “finally”.



```
1 package cursojavacfb;
2
3 import java.util.Scanner;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Scanner sc = new Scanner(System.in);
10
11         int num=0;
12
13         try{
14             num=sc.nextInt();
15             System.out.println("Valor válido: " + num);
16         }catch(Exception erro){
17             System.out.println("Valor Inválido");
18             System.out.println("Erro: " + erro);
19         }finally{
20             System.out.println("Procedimento finalizado.");
21         }
22     }
23 }
24
25 }
```

Agora quando o procedimento terminar tendo ocorrido uma exceção ou não o bloco “finally” será executado, veja a ilustração a seguir.

Com exceção.

Saída - CursoJavaCFB (run) X

```
run:
b
Valor Inválido
Erro: java.util.InputMismatchException
Procedimento finalizado.
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Sem exceção.

Saída - CursoJavaCFB (run) X

```
run:
10
Valor válido: 10
Procedimento finalizado.
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Vamos adicionar uma variável simples para controle de erro e usar no “finally” para saber se houve erro ou não.





```
1 package cursojavacfb;
2
3 import java.util.Scanner;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Scanner sc = new Scanner(System.in);
10
11         int num=0;
12         int er=0;
13
14         try{
15             num=sc.nextInt();
16             System.out.println("Valor válido: " + num);
17         }catch(Exception erro){
18             System.out.println("Valor Inválido");
19             System.out.println("Erro: " + erro);
20             er=1;
21         }finally{
22             if(er==1){
23                 System.out.println("Procedimento finalizado COM erro.");
24             }else{
25                 System.out.println("Procedimento finalizado SEM erro.");
26             }
27         }
28     }
29 }
30
31 }
```

Saída - CursoJavaCFB (run) X

```
run:
10
Valor válido: 10
Procedimento finalizado SEM erro.
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Caso haja alguma exceção a saída será a seguinte.

Saída - CursoJavaCFB (run) X

```
run:
b
Valor Inválido
Erro: java.util.InputMismatchException
Procedimento finalizado COM erro.
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

## Foreach

O loop FOREACH é bem interessante quando queremos percorrer nosso array e trabalhar elemento por elemento deste array de forma simples, pelo menos um pouco mais simples que o loop FOR tradicional.

Em java não existe um comando específico chamado FOREACH, usamos o comando for com uma sintaxe diferente, veja a sintaxe.

```
for(variável_para_o_elemento_do_array : array)
```

O loop FOREACH percorre todo array, cada elemento do array é inserido na variável e dentro do FOR podemos usar esta variável, um detalhe importante é que não manipulamos os elementos diretamente no array.



Em nosso código de exemplo vamos percorrer o array “transportes”, veja.

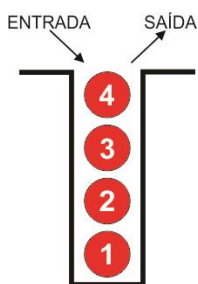
```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         String transportes[] = new String[5];
8
9         transportes[0]="carro";
10        transportes[1]="moto";
11        transportes[2]="navio";
12        transportes[3]="barco";
13        transportes[4]="bicicleta";
14
15        for(String tr:transportes){
16            if(tr=="barco"){
17                System.out.println("Barco está na lista de transportes");
18            }
19        }
20    }
21 }
22
23 }
```

Saida - CursoJavaCFB (run) X

```
run:
Barco está na lista de transportes
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## Stack - Pilha

Pilha é uma coleção de dados com uma particularidade muito importante, o primeiro elemento a entrar é o último elemento a sair.



Na ilustração ao lado podemos entender bem o funcionamento de uma pilha, note que os elementos (bolas vermelhas) que adicionamos nesta pilha são numerados por ordem de entrada, ou seja, a bola de número 1 foi a primeira a ser adicionada na pilha, a de número 2 foi a segunda e assim por diante até a bola 4 que foi a última, neste processo, ao retirarmos as bolas da pilha, qual será a primeira e a última bolas a saírem da pilha?

A bola 4 que foi a última a ser adicionada será a primeira a ser retirada e a bola 1 que foi a primeira a ser adicionada será a última a sair.

Então quando falar de pilha não esqueça “Primeiro a entrar é o último a sair e o último a entrar é o primeiro a sair”.

Para usar pilhas em nosso programa precisamos importar a biblioteca “Stack”.

```
import java.util.Stack;
```

Vamos ao programa de exemplo.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11     }
12
13 }
```

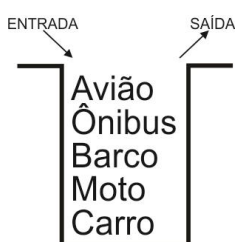
Na linha 9 declaramos e instanciamos uma nova pilha chamada “transp”.

### *push – Inserindo elementos na pilha*

Para adicionar elementos a uma pilha podemos usar o método push, vamos entender como funciona.

```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Onibus");
15         transp.push("Avião");
16
17     }
18
19 }
```

Veja que inserimos 5 elementos em nossa pilha, o primeiro elemento a ser inserido foi o “Carro” e o último foi o “Avião”, veja uma ilustração representando este pilha.



### *firstElement / lastElement*

Estes dois métodos retornam o primeiro e último elementos da pilha, veja o exemplo.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Onibus");
15         transp.push("Avião");
16
17         System.out.println("Primeiro elemento: " + transp.firstElement());
18         System.out.println("Último elemento..: " + transp.lastElement());
19
20     }
21
22 }
```

Saída - CursoJavaCFB (run) X

```
run:
Primeiro elemento: Carro
Último elemento..: Avião
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## get

O método get retorna o elemento indicado pelo parâmetro, veja o exemplo onde obtivemos o elemento da posição 2, lembrando que o primeiro elemento é o de índice zero.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Onibus");
15         transp.push("Avião");
16
17         System.out.println(transp.get(2));
18
19     }
20
21 }
```

Saída - CursoJavaCFB (run) X

```
run:
Barco
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## indexOf

Este método retorna a posição (índice) do elemento indicado como parâmetro, no exemplo obtivemos o índice do elemento "Avião" que está na posição 4, lembrando que inicia em zero.

```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Onibus");
15         transp.push("Avião");
16
17         System.out.println(transp.indexOf("Avião"));
18
19     }
20
21 }
```

Saída - CursoJavaCFB (run) X

```
run:
4
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



## peek

Este método retorna último elemento da pilha, o elemento que está por cima, sem removê-lo da pilha.

```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Onibus");
15         transp.push("Avião");
16
17         System.out.println(transp.peek());
18
19     }
20
21 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Avião
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## size

Este método retorna o tamanho da pilha, o número de elementos.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Onibus");
15         transp.push("Avião");
16
17         System.out.println(transp.size());
18     }
19 }
20
21 }
```

Saída - CursoJavaCFB (run) ×

```
run:
5
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## *isEmpty*

Este método verifica se a pilha está vazia ou não, se estiver retorna true e se não estiver retorna false.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Onibus");
15         transp.push("Avião");
16
17         if(transp.isEmpty()){
18             System.out.println("Pilha Vazia");
19         }else{
20             System.out.println("A Pilha contém elementos");
21         }
22     }
23 }
24
25 }
```

Saída - CursoJavaCFB (run) ×

```
run:
A Pilha contém elementos
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## pop

Este método remove e retorna o elemento que está no topo da pilha, em nosso programa vamos usar um loop “while” que irá executar o método “pop” enquanto a pilha não for vazia e o método pop irá retornar ao println o elemento removido do topo, ao final vamos imprimir a quantidade de elementos da pilha para confirmar que está vazia.





```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Onibus");
15         transp.push("Avião");
16
17         while(!transp.isEmpty()){
18             System.out.println(transp.pop());
19         }
20
21         System.out.println("Quantidade de elementos na pilha: " + transp.size());
22
23     }
24
25 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Avião
Onibus
Barco
Moto
Carro
Quantidade de elementos na pilha: 0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## remove

Este método remove um elemento qualquer da pilha, indicando o elemento ou o índice.

Primeiro vamos remover pelo elemento.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Ônibus");
15         transp.push("Avião");
16
17         transp.remove("Ônibus");
18
19         while(!transp.isEmpty()){
20             System.out.println(transp.pop());
21         }
22     }
23 }
24
25 }
```

Saída - CursoJavaCFB (run) X

```
run:
Avião
Barco
Moto
Carro
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Agora vamos remover pelo índice.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Ônibus");
15         transp.push("Avião");
16
17         transp.remove(3);
18
19         while(!transp.isEmpty()) {
20             System.out.println(transp.pop());
21         }
22     }
23 }
24
25 }
```

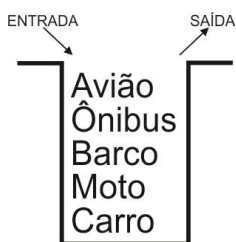
Saída - CursoJavaCFB (run) X

run:  
Avião  
Barco  
Moto  
Carro  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

Note que nos dois exemplos o elemento “Ônibus” não é impresso.

## search

Procura um elemento e retorna sua posição no empilhamento, em nosso exemplo procuramos o elemento “Moto” que está na posição 4, aqui o elemento do topo é o 1.





```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Ônibus");
15         transp.push("Avião");
16
17         System.out.println(transp.search("Moto"));
18
19     }
20
21 }
```

Saída - CursoJavaCFB (run) X

```
run:
4
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## *add*

Adicione um elemento no final na pilha.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args){
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Ônibus");
15         transp.push("Avião");
16
17         System.out.println(transp.add("Bicicleta"));
18
19         while(!transp.isEmpty()){
20             System.out.println(transp.pop());
21         }
22     }
23 }
24
25 }
```

Saída - CursoJavaCFB (run) X

```
run:
true
Bicicleta
Avião
Ônibus
Barco
Moto
Carro
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## clear

Este método remove todos os elementos da pilha.



```
1 package cursojavacfb;
2
3 import java.util.Stack;
4
5 public class CursoJavaCFB {
6
7     public static void main(String[] args) {
8
9         Stack transp = new Stack();
10
11         transp.push("Carro");
12         transp.push("Moto");
13         transp.push("Barco");
14         transp.push("Ônibus");
15         transp.push("Avião");
16
17         transp.clear();
18
19         System.out.println("Quantidade de elementos: " + transp.size());
20
21     }
22
23 }
```

Saída - CursoJavaCFB (run) X

run:  
Quantidade de elementos: 0  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

## sort

Este método ordena de forma crescente os elementos da pilha.



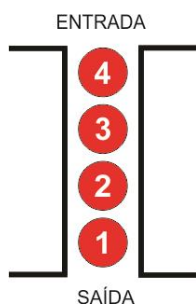
```
1 package cursojavacfb;
2
3 import java.util.Comparator;
4 import java.util.Stack;
5
6 public class CursoJavaCFB {
7
8     public static void main(String[] args) {
9
10         Stack transp = new Stack();
11         Comparator c=null;
12
13         transp.push("Carro");
14         transp.push("Moto");
15         transp.push("Barco");
16         transp.push("Ônibus");
17         transp.push("Avião");
18
19         transp.sort(c);
20
21         while(!transp.isEmpty()){
22             System.out.println(transp.pop());
23         }
24     }
25 }
26
27 }
```

Saída - CursoJavaCFB (run) X

```
run:
Ônibus
Moto
Carro
Barco
Avião
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## Queue – Fila

A estrutura de fila é semelhante a de pilha com uma diferença básica, o primeiro elemento a entrar é o primeiro elemento a sair e o último elemento a entrar é o último elemento a sair.



Para trabalhar com filas precisamos importar duas bibliotecas “Queue” e “LinkedList”.

A declaração de uma fila se difere da pilha por não poder ser instanciada, pois a classe “Queue” é abstrada e não tem necessidade de ser instanciada.



```
1 package cursojavacfb;
2
3 import java.util.LinkedList;
4 import java.util.Queue;
5
6 public class CursoJavaCFB {
7
8     public static void main(String[] args){
9
10         Queue fila=new LinkedList();
11
12     }
13
14 }
```

## add e poll

Em filas usamos os métodos “add” para adicionar os elementos e “poll” para remover, veja no exemplo e note que o primeiro elemento a entrar foi o primeiro a sair.

```
1 package cursojavacfb;
2
3 import java.util.LinkedList;
4 import java.util.Queue;
5
6 public class CursoJavaCFB {
7
8     public static void main(String[] args){
9
10         Queue fila=new LinkedList();
11
12         fila.add("Carro");
13         fila.add("Moto");
14         fila.add("Barco");
15         fila.add("Ônibus");
16         fila.add("Avião");
17
18         while(!fila.isEmpty()){
19             System.out.println(fila.poll());
20         }
21
22     }
23
24 }
```

```
>
Saída - CursoJavaCFB (run) x
run:
Carro
Moto
Barco
Ônibus
Avião
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Assim como na pilha temos vários métodos para adicionar e remover os elementos, veja a tabela a seguir.

Método	Descrição	Exemplo
peek	Retorna sem remover o elemento no topo da fila	fila.peek();
remove	Remove o elemento do topo da fila	fila.remove("Moto");
clear	Remove todos os elementos da fila	fila.clear();
size	Retorna o tamanho da fila	file.size();





## Lista

Lista é outra coleção de dados que temos disponível em Java, basicamente é uma coleção de dados em sequência. O usuário desta interface tem um controle preciso sobre onde será inserido um elemento na lista, podemos acessar os elementos pelo seu índice inteiro (posição na lista), e procurar elementos na lista.

Vamos ao código de exemplo e já usando os métodos add, isEmpty e remove.

```
1 package cursojavacfb;
2
3 import java.util.LinkedList;
4 import java.util.List;
5
6 public class CursoJavaCFB {
7
8     public static void main(String[] args){
9
10         List lista = new LinkedList();
11
12         lista.add("Carro");
13         lista.add("Moto");
14         lista.add("Barco");
15         lista.add("Ônibus");
16         lista.add("Avião");
17
18         while(!lista.isEmpty()){
19             System.out.println(lista.remove(0));
20         }
21     }
22 }
23
24 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Carro
Moto
Barco
Ônibus
Avião
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Além dos métodos add, isEmpty e remove ainda temos alguns métodos interessantes, veja.

Método	Descrição	Exemplo
remove(int)	Remove o elemento da lista pelo seu índice	lista.remove(3);
remove(elemento)	Remove o elemento da lista	lista.remove("Moto");
clear	Remove todos os elementos da lista	lista.clear();
size	Retorna o tamanho da lista	lista.size();
contains	Retorna true se a lista contém o elemento pesquisado	if(lista.contains("Moto"))
get	Retorna o elemento indicado pelo índice	lista.get(3);
set	Redefine o elemento indicado como índice	lista.set(3,"Bicicleta");
indexOf	Retorna o índice indicado pelo elemento	lista.indexOf("Moto");



## Métodos

Métodos são blocos de códigos das classes com alguma funcionalidade específica pertinente à classe que pertence. Basicamente são funcionalidades inseridas nas classes.

Um método pode realizar qualquer rotina e pode ter várias características, como possuir ou não parâmetro de entrada, tipo de retorno, tipo de visualização/acesso, etc.

Programas em Java são escritos juntando métodos e classes que são. Em síntese métodos são pequenas partes de código, dentro de uma classe.

Os métodos são simplesmente funções ou procedimentos, são separados em “blocos” de código e são passíveis de reutilização é claro, podendo ser aproveitados métodos já existentes para a construção rotinas novas.

Em nosso código padrão já existe um método declaro o método “main” (principal).

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7
8     }
9
10 }
```

A anatomia geral de um método é a seguinte.

```
modificador_de_acesso  tipo_de_retorno  nome_do_método  (  parâmetros_de_entrada  ){
    //Comandos
}
```

No caso do método “main” temos a seguinte comparação com a anatomia descrita acima.

```
public static  void  main  (  String[] args  ){
}
```

Vamos criar um método simples que imprime uma mensagem de texto.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7     }
8
9     public static void msg(){
10         System.out.println("\nCanal Fessor Bruno");
11     }
12
13 }
```

Nosso método tem o modificador de acesso “public”, “static” porque o método main é static, o tipo de retorno “void”, o nome “msg” e não tem parâmetros de entrada.

OBS: Mais adiante iremos discutir mais sobre “modificadores de acesso” public e private.

O tipo de retorno representa o tipo de dado que o método retorna pra quem chamou o método.



Ao rodar nosso programa não aparecerá nada no console de saída, isso porque não chamamos nosso método, para que um método seja executado precisamos chamá-lo, veja.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6         msg();
7     }
8
9     public static void msg(){
10         System.out.println("\nCanal Fessor Bruno");
11     }
12 }
13
14
15
```

Saída - CursoJavaCFB (run) X

run:  
Canal Fessor Bruno  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

Agora podemos ver a mensagem no console impressa pelo método “msg”.

## Métodos com parâmetro de entrada

Podemos passar dados de forma fácil para serem trabalhados dentro do método.

Para definir parâmetros de entrada precisamos indicar o tipo e o nome do parâmetro, como se fosse uma variável comum, podemos indicar quantos parâmetros forem necessários, basta separar por vírgula indicando sempre o tipo de dados e o nome do parâmetro.

```
método( tipo_do_parâmetro1 nome_do_parâmetro1, tipo_do_parâmetro2 nome_do_parâmetro2 )
```

É um trabalho bem simples, vamos alterar o método “msg” para receber o texto que desejamos que seja impresso.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         msg("Canal Fessor Bruno");
8         msg("Curso de Java SE");
9         msg("cfbcursos.com.br");
10        msg("www.youtube.com/canalfessorbruno");
11    }
12
13
14    public static void msg(String txt){
15        System.out.println("\n" + txt);
16    }
17
18 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Canal Fessor Bruno
Curso de Java SE
cfbcursos.com.br
www.youtube.com/canalfessorbruno
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que agora definimos nosso método com um parâmetro de entrada do tipo “String” com nome “txt”, agora para chamar o método “msg” precisamos passar a mensagem a ser impressa.

Vamos alterar um pouco mais nosso método e adicionar outro parâmetro para indicar quantas vezes o texto será impresso.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         msg("Canal Fessor Bruno", 4);
8     }
9
10
11     public static void msg(String txt, int qtde) {
12         for(int i=0; i<qtde; i++) {
13             System.out.println("\n" + txt);
14         }
15     }
16 }
17
```

Saída - CursoJavaCFB (run) ×

```
run:
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
Canal Fessor Bruno
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Veja que no loop FOR usamos o parâmetro qtde para indicar a condição de parada.

## Métodos com retorno

Os métodos podem retornar valores para quem fez a chamada, veja um exemplo de um método que faz a soma de dois valores e retorna o resultado desta soma.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         int num1,num2,res;
8
9         num1=10;
10        num2=5;
11
12        res=soma(num1,num2);
13
14        System.out.println("A soma é: " + res);
15    }
16
17    public static int soma(int n1, int n2){
18        int sum=n1+n2;
19        return sum;
20    }
21 }
22
23 }
```

Saída - CursoJavaCFB (run) ×

```
run:
A soma é: 15
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que na definição do nosso método na linha 18 o tipo de retorno não é mais “void”, agora como nosso método precisa retornar um valor inteiro o tipo de retorno é “int”.

OBS: Sempre que o método não for retornar nada definimos como VOID.

Na linha 20 indicamos o retorno no método, no caso a variável “sum” que contem as soma dos valores dos parâmetros de entrada.

O código acima foi um procedimento menos direto, podemos resumir para diminuir o código, veja o programa acima refeito.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         int num1,num2;
8
9         num1=10;
10        num2=5;
11
12        System.out.println("A soma é: " + soma(num1,num2));
13    }
14
15    public static int soma(int n1, int n2){
16        return n1+n2;
17    }
18 }
19
20 }
```



Veja que reduzimos uma variável “res”.

Vamos reduzir todas as variáveis.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         System.out.println("A soma é: " + soma(10,5));
8
9     }
10
11     public static int soma(int n1, int n2){
12         return n1+n2;
13     }
14
15 }
```

O resultado dos três códigos anteriores é exatamente o mesmo.

## Classes

Em uma linguagem de programação orientada a objetos como é o caso do Java é muito importante entender o conceito de classes, diria que até obrigatório saber trabalhar com classes.

Vamos dedicar este capítulo a este entendimento.

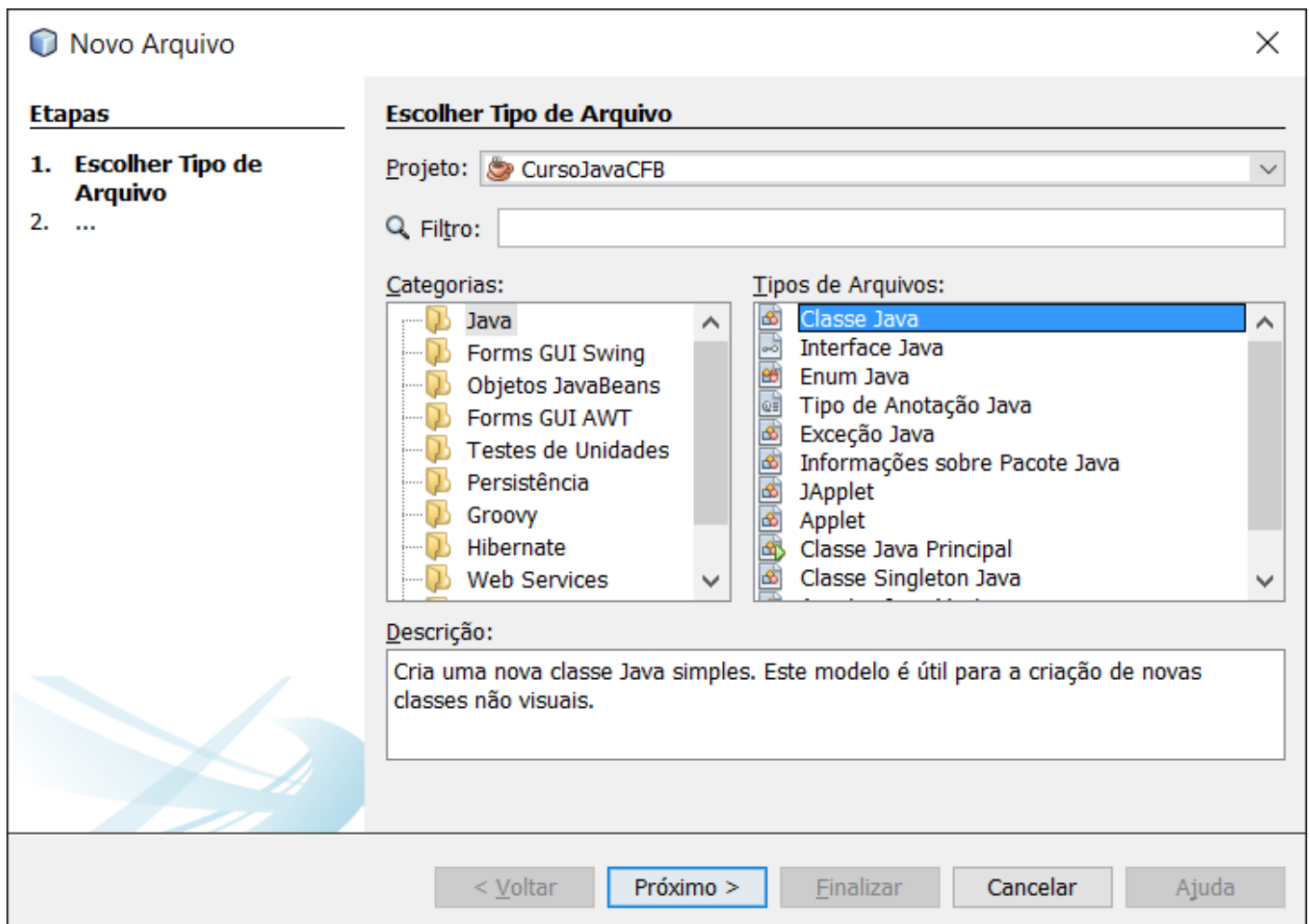
Uma classe é um tipo de dados composto por membros do qual podemos criar objetos, os objetos são instâncias das classes. Podemos criar uma classe e instanciar vários objetos desta classe.

Cada arquivo em java é uma classe, vamos entender a estrutura de uma classe.

## Pacotes

Os pacotes são uma estrutura de pastas para organizar as diversas classes de nosso programa.

Vamos criar uma classe chamada carro. Clique no menu ARQUIVO – NOVO ARQUIVO, selecione “Classe Java”.



Clique em “Próximo” e digite o nome da classe “Carro”.





New Classe Java

Etapas

1. Escolher Tipo de Arquivo

2. Nome e Localização

Nome e Localização

Nome da Classe:

Carro

Projeto:

CursoJavaCFB

Localização:

Pacotes de Códigos-fonte

Paquete:

cursojavacfb

Arquivo Criado:

desktop\Meus Cursos\Java\CursoJavaCFB\src\cursojavacfb\Carro.java

< Voltar

Próximo >

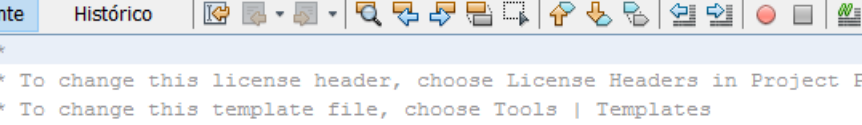
Finalizar

Cancelar

Ajuda

Clique em “Finalizar”.

Nossa classe será criada com seu código básico.



```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package cursojavacfb;
7
8   /**
9    *
10   * @author Bruno
11   */
12   public class Carro {
13
14   }
15
```

Vou apagar todos os comentários.



```
1 package cursojavacfb;
2
3 public class Carro {
4
5 }
6
```

Veja que a classe foi criada no pacote “cursojavacfb”, ou seja, em nosso projeto “CursoJavaCFB” existe uma pasta com nome “cursojavacfb” onde serão armazenadas todas as classes deste projeto.

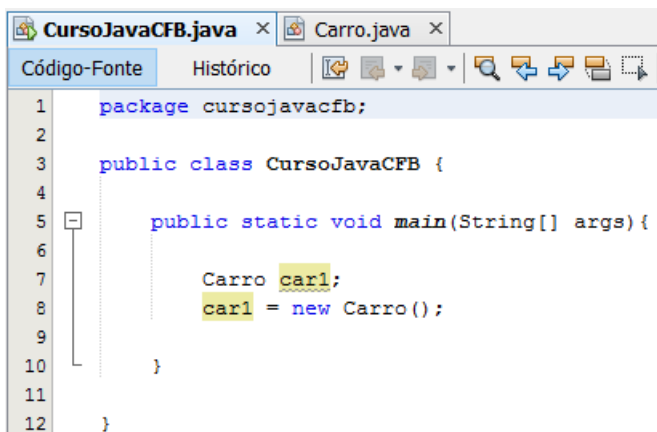
C:\Users\Bruno\Desktop\Meus Cursos\Java\CursoJavaCFB\src\cursojavacfb

Pronto, já criamos uma classe básica chamada Carro e já podemos instanciar objetos desta nossa classe por nosso programa afora.

### *Instanciando novos objetos*

Instanciar novos objetos de uma classe é uma tarefa bem simples, basta usar a palavra reservada “new”, vamos ver a seguir.

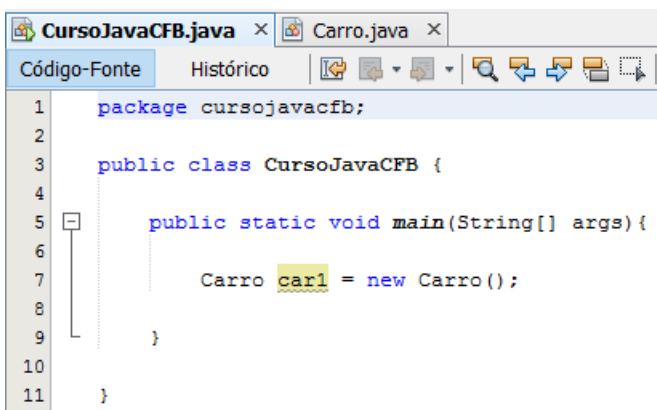
Vamos voltar para o arquivo “CursoJavaCFB” e instanciar um novo objeto da nossa classe “Carro”.



```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Carro car1;
8         car1 = new Carro();
9     }
10
11 }
12
```

No código acima declaramos um elemento do tipo “Carro” chamado “car1” na linha 7 e na linha 8 instanciamos um novo objeto.

Podemos declarar e instanciar na mesma linha, vamos alterar o código anterior.



```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Carro car1 = new Carro();
8     }
9
10 }
11
```

A partir de agora podemos usar o objeto car1 em nosso programa, porém, não criamos nenhum método ou propriedade na classe Carro.



Vamos criar algumas propriedades para a classe Carro. Veja o código a seguir onde criamos quatro parâmetros na classe Carro.

```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Carro {
4     public String nome;
5     public String cor;
6     public int velMax;
7     public int pot;
8
9 }
10
```

Vamos usar os parâmetros em nossos objetos instanciados da classe “Carro”.

```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Carro car1 = new Carro();
8         Carro car2 = new Carro();
9
10        car1.nome="Rapidão";
11        car1.cor="Vermelho";
12        car1.pot=450;
13        car1.velMax=320;
14
15        car2.nome="Familião";
16        car2.cor="Prata";
17        car2.pot=100;
18        car2.velMax=180;
19
20    }
21
22 }
```

Note que instanciamos dois objetos “car1” e “car2” da classe “Carro” e atribuímos valores aos parâmetros nome, cor, pot e velMax de cada um dos objetos.

Agora vamos criar um método em nossa classe para facilitar a impressão das informações dos carros.



```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Carro {
4     public String nome;
5     public String cor;
6     public int velMax;
7     public int pot;
8
9     public void imprime() {
10         System.out.println("\nNome.....: " + this.nome);
11         System.out.println("Cor.....: " + this.cor);
12         System.out.println("Potência.....: " + this.pot);
13         System.out.println("Velocidade Máxima: " + this.velMax);
14     }
15 }
16
```

Agora vamos chamar o método “imprime” nos objetos instanciados.

```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         Carro car1 = new Carro();
8         Carro car2 = new Carro();
9
10        car1.nome="Rapidão";
11        car1.cor="Vermelho";
12        car1.pot=450;
13        car1.velMax=320;
14
15        car2.nome="Familião";
16        car2.cor="Prata";
17        car2.pot=100;
18        car2.velMax=180;
19
20        car1.imprime();
21        car2.imprime();
22    }
23 }
24
25
```

```
Saída - CursoJavaCFB (run) x
run:
Nome.....: Rapidão
Cor.....: Vermelho
Potência.....: 450
Velocidade Máxima: 320

Nome.....: Familião
Cor.....: Prata
Potência.....: 100
Velocidade Máxima: 180
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



## Método construtor

O método construtor é o método que será executado automaticamente quando for criada uma nova instância da classe, ou seja, ao se criar uma nova instância, um novo objeto da classe, o método construtor será automaticamente executado para este novo objeto.

Para se definir um método como construtor basta dar o mesmo nome da classe ao método.

Por exemplo, se a classe se chama “Carro” o nome do método construtor será “Carro”.

Vamos adicionar o método construtor em nossa classe Carro com quatro parâmetros de entrada, veja o exemplo da linha 9 até 14.

```
1  package cursojavacfb;
2
3  public class Carro {
4      public String nome;
5      public String cor;
6      public int velMax;
7      public int pot;
8
9      public Carro(String no, String cr, int vm, int pt){
10         this.nome=no;
11         this.cor=cr;
12         this.velMax=vm;
13         this.pot=pt;
14     }
15
16     public void imprime(){
17         System.out.println("\nNome.....: " + this.nome);
18         System.out.println("Cor.....: " + this.cor);
19         System.out.println("Potência.....: " + this.pot);
20         System.out.println("Velocidade Máxima: " + this.velMax);
21     }
22 }
```

Como definimos um método construtor com quatro parâmetros de entrada será obrigatório neste caso passar os valores para os parâmetros no momento da chamada, veja o exemplo.



```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Carro car1 = new Carro("Rapidão","Vermelho",450,320);
8         Carro car2 = new Carro("Familião","Prata",100,180);
9
10        car1.imprime();
11        car2.imprime();
12
13    }
14
15 }
```

```
Saída - CursoJavaCFB (run) x
run:
Nome.....: Rapidão
Cor.....: Vermelho
Potência.....: 320
Velocidade Máxima: 450

Nome.....: Familião
Cor.....: Prata
Potência.....: 180
Velocidade Máxima: 100
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## Sobrecarga de métodos

Sobrecarga de métodos é um recurso em orientação a objetos interessante que nos permite criar mais de um método com mesmo nome, desde que a definição dos parâmetros de entrada sejam diferentes.

Em nosso programa de exemplo vamos definir um novo método construtor, sem uso de parâmetros, vamos ver o exemplo.



```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Carro {
4     public String nome;
5     public String cor;
6     public int velMax;
7     public int pot;
8
9     public Carro(String no, String cr, int vm, int pt){
10         this.nome=no;
11         this.cor=cr;
12         this.velMax=vm;
13         this.pot=pt;
14     }
15
16     public Carro(){
17         this.nome="Padrão";
18         this.cor="Branco";
19         this.velMax=120;
20         this.pot=80;
21     }
22
23     public void imprime(){
24         System.out.println("\nNome.....: " + this.nome);
25         System.out.println("Cor.....: " + this.cor);
26         System.out.println("Potência.....: " + this.pot);
27         System.out.println("Velocidade Máxima: " + this.velMax);
28     }
29 }
```

Note que criamos dois métodos construtores o primeiro na linha 9 com quatro parâmetros de entrada e o segundo na linha 16 sem nenhum parâmetro.

Vamos instanciar um novo objeto em nosso código principal e observar as diferenças.



```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Carro car1 = new Carro("Rapidão", "Vermelho", 450, 320);
8         Carro car2 = new Carro("Familião", "Prata", 100, 180);
9         Carro car3 = new Carro();
10
11         car1.imprime();
12         car2.imprime();
13         car3.imprime();
14     }
15 }
16
17 }

Saída - CursoJavaCFB (run) x
run:
Nome.....: Rapidão
Cor.....: Vermelho
Potência.....: 320
Velocidade Máxima: 450

Nome.....: Familião
Cor.....: Prata
Potência.....: 180
Velocidade Máxima: 100

Nome.....: Padrão
Cor.....: Branco
Potência.....: 80
Velocidade Máxima: 120
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que declaramos e instanciamos um novo objeto “car3” e não passamos parâmetros de entrada para ele, desta maneira o método construtor executado será o segundo, na linha 16.

Veja que ao imprimir os valores chamando o método “imprime” do objeto “car3” ele imprime os valores atribuídos no método construtor sem parâmetros de entrada.

## Public x Private

Os modificadores de acesso “public” e “private” definem como a variável ou método será “visível/acessível” dentro do nosso programa.

Se não for definido pelo programador, automaticamente será definido como “public”.

Os métodos ou propriedades definidas como “public” tem um tipo de acesso público, ou seja, podem ser usadas ou acessadas de fora da classe.

Já os métodos ou propriedades definidas como “private” são provadas e só podem ser acessadas de dentro da própria classe.

Vamos ao exemplo, veja que agora definimos a propriedade “velMax” como private e a retiramos do método construtor principal.





```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Carro {
4     public String nome;
5     public String cor;
6     private int velMax;
7     public int pot;
8
9     //Método construtor principal
10    public Carro(String no, String cr, int pt){
11        this.nome=no;
12        this.cor=cr;
13        this.pot=pt;
14    }
15
16    //Método construtor sobrecarregado
17    public Carro () {
18        this.nome="Padrão";
19        this.cor="Branco";
20        this.velMax=120;
21        this.pot=80;
22    }
23
24    public void imprime() {
25        System.out.println("\nNome.....: " + this.nome);
26        System.out.println("Cor.....: " + this.cor);
27        System.out.println("Potência.....: " + this.pot);
28        System.out.println("Velocidade Máxima: " + this.velMax);
29    }
30 }
```

Isso porque os valores atribuídos por parâmetros são externos ao método, como a variável agora é “private” só poderá ser acessada pelo próprio método.

Vamos definir a velocidade máxima de acordo com a potência, como existem dois métodos construtores que precisam calcular a velocidade máxima, vamos criar um método para este cálculo, assim evitamos criar duas rotinas semelhantes.

## Herança

O conceito de herança deve ser bem entendido. É bem simples, basicamente é quando criamos uma nova classe que vai herdar características de outra classe, definimos isto através da palavra “extends”.

Vamos criar uma nova classe chamada “Caminhao” que herdará a classe carro.

Clique no menu “ARQUIVO – NOVO ARQUIVO”, selecione “Classe Java”, nossa nova classe terá o nome “Caminhao”.



New Classe Java

Etapas

1. Escolher Tipo de Arquivo

2. Nome e Localização

Nome e Localização

Nome da Classe: Caminhao

Projeto: CursoJavaCFB

Localização: Pacotes de Códigos-fonte

Paquete: cursojavacfb

Arquivo Criado: top\Meus Cursos\Java\CursoJavaCFB\src\cursojavacfb\Caminhao.java

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

Após apagar todos os comentários, vamos herdar a classe “Carro” como mostra o exemplo a seguir.

CursoJavaCFB.java x Carro.java x Caminhao.java x

Código-Fonte Histórico

1 package cursojavacfb;

2

3 public class Caminhao extends Carro{

4

5 }

6

Isso fará com que a classe “Caminhao” automaticamente herde todas as propriedades e métodos da classe “Carro”, por isso vamos realizar algumas alterações na classe carro e criar uma nova propriedade na classe “Carro” para identificar o tipo, se for um carro simples iremos definir como tipo 1 e se for um caminhão iremos definir como tipo 2.



```
CursoJavaCFB.java x Carro.java x Caminhao.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3
4 public class Carro {
5     public String nome;
6     public String cor;
7     public int velMax;
8     public int pot;
9     public int tipo;
10
11 //Método construtor principal
12 public Carro(String no, String cr, int pt){
13     this.tipo=1;
14     this.nome=no;
15     this.cor=cr;
16     this.pot=pt;
17     this.velMax=calcVelMax(pot);
18 }
19
20 //Método construtor sobrecarregado
21 public Carro() {
22     this.tipo=1;
23     this.nome="Padrão";
24     this.cor="Branco";
25     this.pot=80;
26     this.velMax=calcVelMax(pot);
27 }
28
29 public int calcVelMax(int pt){
30     int vm=0;
31     if(pt <=80){
32         vm=120;
33     }else if(pt <=200){
34         vm=260;
35     }else if(pt > 201){
36         vm=360;
37     }
38     return vm;
39 }
40
41 public void imprime(){
42     System.out.println("\nNome.....: " + this.nome);
43     System.out.println("Cor.....: " + this.cor);
44     System.out.println("Potência.....: " + this.pot);
45     System.out.println("Velocidade Máxima: " + this.velMax);
46 }
```

Agora vamos criar o construtor da classe “Caminhao”.

OBS: Se não for criado um construtor para a classe “Caminhao” será usado o construtor da classe pai que é a classe “Carro”.



```
CursoJavaCFB.java x Carro.java x Caminhao.java x
Código-Fonte  Histórico
1  package cursojavacfb;
2
3  public class Caminhao extends Carro{
4
5      public Caminhao(String no, String cr){
6          this.tipo=2;
7          this.nome=no;
8          this.cor=cr;
9          this.pot=500;
10         this.velMax=180;
11     }
12
13 }
14
```

Note que as variáveis `tipo`, `nome`, `cor`, `pot` e `velMax` não estão definidas “visivelmente” dentro da classe “Caminhao”, mas elas existem para a classe “Caminhao” simplesmente por esta classe herdar a classe “Carro” que tem as definições destas propriedades.

Vamos criar um objeto do tipo “Caminhao” em nosso código principal.



```
CursoJavaCFB.java x Carro.java x Caminhao.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Carro car1 = new Carro("Rapidão", "Vermelho", 450);
8         Carro car2 = new Carro("Familião", "Prata", 150);
9         Carro car3 = new Carro();
10        Caminhao car4 = new Caminhao("Pesadão", "Cinza");
11
12        car1.imprime();
13        car2.imprime();
14        car3.imprime();
15        car4.imprime();
16
17    }
18
19 }
```

Saída - CursoJavaCFB (run) x

```
run:
Nome.....: Rapidão
Cor.....: Vermelho
Potência.....: 450
Velocidade Máxima: 360

Nome.....: Familião
Cor.....: Prata
Potência.....: 150
Velocidade Máxima: 260

Nome.....: Padrão
Cor.....: Branco
Potência.....: 80
Velocidade Máxima: 120

Nome.....: Pesadão
Cor.....: Cinza
Potência.....: 500
Velocidade Máxima: 180
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Carro {
4     public String nome;
5     public String cor;
6     private int velMax;
7     public int pot;
8
9     //Método construtor principal
10    public Carro(String no, String cr, int pt){
11        this.nome=no;
12        this.cor=cr;
13        this.pot=pt;
14        this.velMax=calcVelMax(pot);
15    }
16
17    //Método construtor sobrecarregado
18    public Carro(){
19        this.nome="Padrão";
20        this.cor="Branco";
21        this.pot=80;
22        this.velMax=calcVelMax(pot);
23    }
24
25    private int calcVelMax(int pt){
26        int vm=0;
27        if(pt <=80){
28            vm=120;
29        }else if(pt <=200){
30            vm=260;
31        }else if(pt > 201){
32            vm=360;
33        }
34        return vm;
35    }
36
37    public void imprime(){
38        System.out.println("\nNome.....: " + this.nome);
39        System.out.println("Cor.....: " + this.cor);
40        System.out.println("Potência.....: " + this.pot);
41        System.out.println("Velocidade Máxima: " + this.velMax);
42    }
43 }
```

Veja que agora criamos um método chamado “calcVelMax” que deve retornar um valor inteiro e tem seu acesso privado somente pela classe, na linha 25. Este método vai receber um valor relacionado à potência e irá retornar a velocidade máxima do carro em relação a esta potência.

Agora, nas linhas 14 e 22 para definir a velocidade máxima simplesmente chamamos o método “calcVelMax” passando o valor de retorno para a variável privada “velMax”.

Agora precisamos alterar em nosso código principal.



```
CursoJavaCFB.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Carro car1 = new Carro("Rapidão","Vermelho",450);
8         Carro car2 = new Carro("Familião","Prata",150);
9         Carro car3 = new Carro();
10
11         car1.imprime();
12         car2.imprime();
13         car3.imprime();
14     }
15 }
16
17 }
```

```
Saída - CursoJavaCFB (run) x
run:
Nome.....: Rapidão
Cor.....: Vermelho
Potência.....: 450
Velocidade Máxima: 360
Nome.....: Familião
Cor.....: Prata
Potência.....: 150
Velocidade Máxima: 260
Nome.....: Padrão
Cor.....: Branco
Potência.....: 80
Velocidade Máxima: 120
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)
```

Note que não passamos o valor para o parâmetro “velMax” e mesmo assim as velocidades máximas foram calculadas em função da potência.

## Protected

O modificador protected deixa o atributo visível para todas as outras classes e subclasses que estão no mesmo pacote, semelhante ao public. A principal diferença para public é que apenas as classes do mesmo pacote tem acesso ao elemento.

Veja a tabela informativa sobre o tipo/local de acesso/visibilidade dos modificadores.


Modificador	Classe	Pacote	Subclasse	Global
Public	Sim	Sim	Sim	Sim
Protected	Sim	Sim	Sim	Não
Sem modificador	Sim	Sim	Não	Não
Private	Sim	Não	Não	Não

## Get e Set

Os métodos “get” e “set” servem para obter ou alterar, respectivamente, determinadas propriedades de uma classe.

Vamos criar uma nova classe chamada “Colaborador”.



 New Classe Java ✕

**Etapas**

- Escolher Tipo de Arquivo
- Nome e Localização**

**Nome e Localização**

Nome da Classe:

Colaborador

Projeto:

CursoJavaCFB

Localização:

Pacotes de Códigos-fonte

Paquete:

cursojavacfb

Arquivo Criado:

\\Meus Cursos\Java\CursoJavaCFB\src\cursojavacfb\Colaborador.java

< Voltar

Próximo >

**Finalizar**

Cancelar

Ajuda

Na classe “Colaborador” vamos declarar três parâmetros private, três métodos “set” e três métodos “get”.





```
1 package cursojavacfb;
2
3 public class Colaborador {
4
5     private String nome;
6     private int cargo;
7     private double salario;
8
9     public void setNome(String no) {
10         this.nome=no;
11     }
12
13     public void setCargo(int ca) {
14         this.cargo=ca;
15     }
16
17     public void setSalario(double sa) {
18         this.salario=sa;
19     }
20
21     public String getNome() {
22         return this.nome;
23     }
24
25     public int geCargo() {
26         return this.cargo;
27     }
28
29     public double getSalario() {
30         return this.salario;
31     }
32 }
```

OBS: Na linha 25 corrija o nome do método para “getCargo”.

Os métodos “set” destacados em vermelho são métodos que vão “setar” configurar/definir o valor de uma propriedade por exemplo, o padrão de construção é que permita receber um parâmetro de entrada para que possa definir o valor de uma propriedade.

Note que mesmo que os parâmetros/variáveis sejam “private” nós conseguiremos, usando os métodos “get” e “set” manipular estes elementos.

Em nosso arquivo principal, vamos instanciar um objeto da classe “Colaborador” e usar os métodos “get” e “set”.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Colaborador c1=new Colaborador();
8
9         c1.setNome("Bruno");
10        c1.setCargo(1);
11        c1.setSalario(5000.0);
12
13        System.out.println("Nome...: " + c1.getNome());
14        System.out.println("Cargo...: " + c1.geCargo());
15        System.out.println("Salário: " + c1.getSalario());
16
17    }
18
19 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Nome...: Bruno
Cargo...: 1
Salário: 5000.0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## this

O operador “this” indica auto-referência, basicamente serve para indicar que estamos operando um elemento pertencente à própria classe, ou seja, faz uma referência à raiz da classe.

Vamos modificar a classe “Colaborador”.



```
1 package cursojavacfb;
2
3 public class Colaborador {
4
5     private String nome;
6     private int cargo;
7     private double salario;
8
9     public void setNome(String nome) {
10         this.nome=nome;
11     }
12
13     public void setCargo(int cargo){
14         this.cargo=cargo;
15     }
16
17     public void setSalario(double salario){
18         this.salario=salario;
19     }
20
21     public String getNome(){
22         return this.nome;
23     }
24
25     public int geCargo(){
26         return this.cargo;
27     }
28
29     public double getSalario(){
30         return this.salario;
31     }
32 }
```

OBS: Na linha 25 corrija o nome do método para “getCargo”.

Note que estamos usando como parâmetros de entrada variáveis com os mesmos nomes das variáveis da classe, mas como isso é possível? Porque não o compilador não confunde as variáveis? Justamente por causa do uso do operador “this”! Quando usamos o “this” na frente das variáveis nas linhas 10, 14 e 18, indicamos que estamos nos referenciando às variáveis da classe e não aos parâmetros.

### *Passando objetos como parâmetros para métodos*

Só como intuito didático vou criar um método no código principal da classe “CursoJavaCFB” que receberá um objeto do tipo colaborador e irá retornar seu salário.



```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Colaborador c1=new Colaborador();
8
9         c1.setNome("Bruno");
10        c1.setCargo(1);
11        c1.setSalario(5000.0);
12
13        System.out.println("Nome...: " + c1.getNome());
14        System.out.println("Cargo...: " + c1.geCargo());
15        System.out.println("Salário: " + obtemSalario(c1));
16
17    }
18
19    public static double obtemSalario(Colaborador c){
20        return c.getSalario();
21    }
22
23 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Nome...: Bruno
Cargo...: 1
Salário: 5000.0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

OBS: Na linha 14 corrija o nome do método para “getCargo”.

O método “obtemSalario” precisou ser “static” simplesmente pelo fato da classe “main”, onde estamos chamando o método, ser “static”. Se retirar “static” do método “main” também poderá retirar “static” do método “obtemSalario”.

## Classes abstratas

Basicamente classes do tipo “abstract” não podem ser instanciadas. Classes abstratas servem somente como modelo de definição para outras classes, por exemplo, em nossa empresa vamos criar dois tipos de colaboradores “Gerente” e “Auxiliar”, não teremos um tipo de colaborador genérico, portanto, não iremos instanciar objetos do tipo “Colaborador” e sim objetos do tipo “Gerente” e ou “Auxiliar”.

Teremos Colaboradores do tipo Gerente ou Auxiliar.

Vamos entender como isso funciona.

Em nossa classe “Colaborador” vamos alterar para “abstract” como vemos na linha 3 do exemplo a seguir.



```
CursoJavaCFB.java x Colaborador.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 abstract class Colaborador {
4
5     private String nome;
6     private int cargo;
7     private double salario;
8
9     public void setNome(String nome) {
10         this.nome=nome;
11     }
12
13     public void setCargo(int cargo) {
14         this.cargo=cargo;
15     }
16
17     public void setSalario(double salario) {
18         this.salario=salario;
19     }
20
21     public String getNome() {
22         return this.nome;
23     }
24
25     public int getCargo() {
26         return this.cargo;
27     }
28
29     public double getSalario() {
30         return this.salario;
31     }
32 }
```

Agora vamos criar a classes “Gerente” e “Auxiliar” com os códigos a seguir.

```
CursoJavaCFB.java x Colaborador.java x Gerente.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Gerente extends Colaborador{
4
5     public Gerente() {
6
7     }
8
9 }
10
```

```
CursoJavaCFB.java x Colaborador.java x Gerente.java x Auxiliar.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Auxiliar extends Colaborador{
4
5     public Auxiliar() {
6
7     }
8
9 }
10
```



Note que as classes “Gerente” e “Auxiliar” herdam a classe “Colaborador”.

Agora no construtor das classes “Gerente” e “Auxiliar” vamos chamar os métodos “setCargo” e “setSalario”.

```
CursoJavaCFB.java x Colaborador.java x Gerente.java x Auxiliar.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Gerente extends Colaborador{
4
5     public Gerente(){
6         super.setCargo(1);
7         super.setSalario(5000.00);
8     }
9
10 }
11
```

```
CursoJavaCFB.java x Colaborador.java x Gerente.java x Auxiliar.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Auxiliar extends Colaborador{
4
5     public Auxiliar(){
6         super.setCargo(2);
7         super.setSalario(1000.00);
8     }
9
10 }
11
```

Note que usamos “super” antes da chamada dos métodos, não é obrigatório, mas evita alguns problemas no construtor porque, no momento em que o método substituído é chamado, o objeto não está completamente inicializado.

Usando “super” estamos nos referindo diretamente aos métodos da classe pai, que no caso é “Colaborador”.

Agora vamos alterar o código da classe principal “CursoJavaCFB”.



```
CursoJavaCFB.java x Colaborador.java x Gerente.java x Auxiliar.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Colaborador c1=new Gerente();
8         Colaborador c2=new Auxiliar();
9
10        c1.setNome("Bruno");
11        c2.setNome("Chuck");
12
13        System.out.println("Nome...: " + c1.getNome());
14        System.out.println("Cargo...: " + c1.getCargo());
15        System.out.println("Salário: " + obterSalario(c1) + "\n");
16
17        System.out.println("Nome...: " + c2.getNome());
18        System.out.println("Cargo...: " + c2.getCargo());
19        System.out.println("Salário: " + obterSalario(c2) + "\n");
20
21    }
22
23    public static double obterSalario(Colaborador c){
24        return c.getSalario();
25    }
26
27 }
```

```
Saída - CursoJavaCFB (run) x
run:
Nome...: Bruno
Cargo...: 1
Salário: 5000.0

Nome...: Chuck
Cargo...: 2
Salário: 1000.0

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que na declaração dos elementos “c1” e “c2” instanciamos objetos do tipo “Gerente” e “Auxiliar”.

Como a classe “Colaborador” é abstrata o comando a seguir não é possível.

```
Colaborador c1 = new Colaborador();
```

## @Override

A palavra reservada “@Override” deverá ser usada sempre que formos sobrescrever um método da classe pai.

Vamos ao exemplo.

Na classe “Colaborador” vamos criar o método “getNomeCargo”.

```
public String getNomeCargo(){
    return "";
}
```

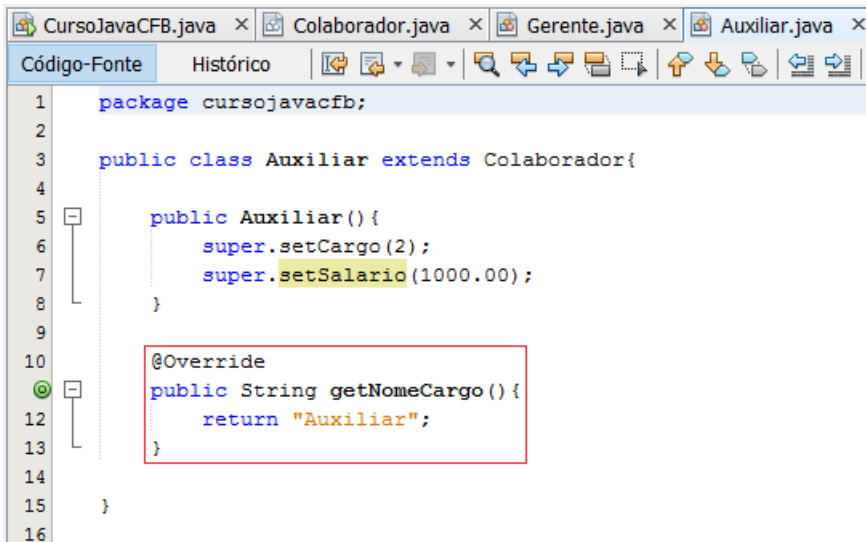


```
CursoJavaCFB.java x Colaborador.java x Gerente.java x Auxiliar.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 abstract class Colaborador {
4
5     private String nome;
6     private int cargo;
7     private double salario;
8
9     public void setNome(String nome) {
10         this.nome=nome;
11     }
12
13     public void setCargo(int cargo) {
14         this.cargo=cargo;
15     }
16
17     public void setSalario(double salario) {
18         this.salario=salario;
19     }
20
21     public String getNome() {
22         return this.nome;
23     }
24
25     public int getCargo() {
26         return this.cargo;
27     }
28
29     public double getSalario() {
30         return this.salario;
31     }
32
33     public String getNomeCargo() {
34         return "";
35     }
36 }
```

Agora nas classes “Gerente” e “Auxiliar” vamos sobrescrever este método.

```
CursoJavaCFB.java x Colaborador.java x Gerente.java x Auxiliar.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Gerente extends Colaborador{
4
5     public Gerente() {
6         super.setCargo(1);
7         super.setSalario(5000.00);
8     }
9
10     @Override
11     public String getNomeCargo() {
12         return "Gerente";
13     }
14 }
15
```





```
1 package cursojavacfb;
2
3 public class Auxiliar extends Colaborador{
4
5     public Auxiliar(){
6         super.setCargo(2);
7         super.setSalario(1000.00);
8     }
9
10    @Override
11    public String getNomeCargo(){
12        return "Auxiliar";
13    }
14
15 }
16
```

Agora na classe principal “CursoJavaCFB” vamos chamar adicionar as chamadas do método “getNomeCargo”.



```
CursoJavaCFB.java x Colaborador.java x Gerente.java x Auxiliar.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Colaborador c1=new Gerente();
8         Colaborador c2=new Auxiliar();
9
10        c1.setNome("Bruno");
11        c2.setNome("Chuck");
12
13        System.out.println("Nome...: " + c1.getNome());
14        System.out.println("Cargo...: " + c1.getCargo());
15        System.out.println("Cargo...: " + c1.getNomeCargo());
16        System.out.println("Salário: " + obtemSalario(c1) + "\n");
17
18        System.out.println("Nome...: " + c2.getNome());
19        System.out.println("Cargo...: " + c2.getCargo());
20        System.out.println("Cargo...: " + c2.getNomeCargo());
21        System.out.println("Salário: " + obtemSalario(c2) + "\n");
22
23    }
24
25    public static double obtemSalario(Colaborador c){
26        return c.getSalario();
27    }
28
29 }
```

Saída - CursoJavaCFB (run) x

```
run:
Nome...: Bruno
Cargo...: 1
Cargo...: Gerente
Salário: 5000.0

Nome...: Chuck
Cargo...: 2
Cargo...: Auxiliar
Salário: 1000.0

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## Interfaces

Uma interface define ações que devem ser obrigatoriamente executadas, mas que podem ser executadas de formas diferentes em cada classe.

Interfaces possuem elementos como valores constantes ou assinaturas de métodos que devem ser implementados dentro da classe que herde a interface.

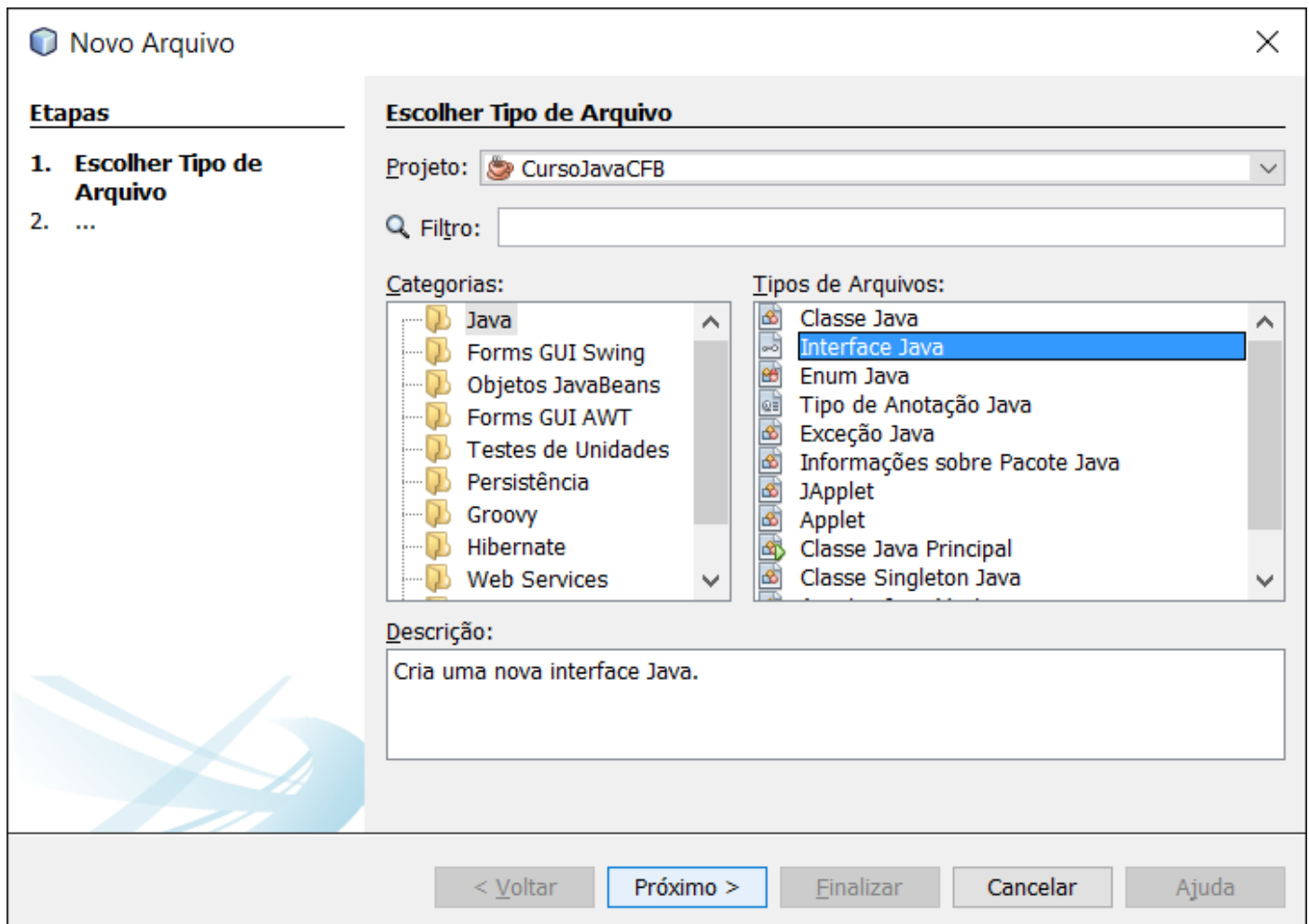
Qual o objetivo? Muitas classes, as vezes, possuem os mesmos métodos, porém, podem executá-lo de forma diferente.

Por exemplo, uma interface chamada "rua" que possui a assinatura do método "dirigir()", desta forma, toda classe que implementar "rua" deve dizer como "dirigir()". Assim, se tivermos duas classes chamadas "carro" e "caminhão" ambas implementando a interface "rua", nas duas classes devemos codificar a forma como cada um irá "dirigir()".

Interfaces são criadas da mesma forma classes, mas utilizando a palavra-chave interface ao invés de class.



Vamos a um programa de exemplo, para criar uma interface clique no menu “ARQUIVO – NOVO ARQUIVO”.



Selecione a opção “Interface Java”.

Nossa interface irá se chamar “Aereo”.



New Interface Java

1. Escolher Tipo de Arquivo

2. Nome e Localização

Nome e Localização

Nome da Classe: Aereo

Projeto: CursoJavaCFB

Localização: Pacotes de Códigos-fonte

Paquete: cursojavacfb

Arquivo Criado: desktop\Meus Cursos\Java\CursoJavaCFB\src\cursojavacfb\Aereo.java

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

Nesta interface, vamos declarar as assinaturas dos métodos que serão obrigatórios a implementação para os elementos que implementarem esta interface.

```
CursoJavaCFB.java x Aereo.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public interface Aereo {
4     void maisVelocidade();
5     void menosVelocidade();
6     void subir();
7     void descer();
8     void ligar();
9     void desligar();
10
11 }
12
```

Veja que na linha três indicamos que estamos criando uma interface. Note também o padrão de declaração, como precisamos declarar somente a assinatura dos métodos, terminamos com ponto e vírgula, aqui não iremos implementar a rotina do método, somente a assinatura.

Agora vamos criar uma nova classe chamada “Aviao”.



Novo Arquivo

✕

1. Escolher Tipo de Arquivo

2. ...

Escolher Tipo de Arquivo

Projeto: CursoJavaCFB

Filtro:

Categorias:

Java

Forms GUI Swing

Objetos JavaBeans

Forms GUI AWT

Testes de Unidades

Persistência

Groovy

Hibernate

Web Services

Tipos de Arquivos:

Classe Java

Interface Java

Enum Java

Tipo de Anotação Java

Exceção Java

Informações sobre Pacote Java

JApplet

Applet

Classe Java Principal

Classe Singleton Java

Descrição:

Cria uma nova classe Java simples. Este modelo é útil para a criação de novas classes não visuais.

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

109



New Classe Java ✕

**Etapas**

- Escolher Tipo de Arquivo
- Nome e Localização**

**Nome e Localização**

Nome da Classe:

Aviao

Projeto:

CursoJavaCFB

Localização:

Pacotes de Códigos-fonte

Paquete:

cursojavacfb

Arquivo Criado:

Desktop\Meus Cursos\Java\CursoJavaCFB\src\cursojavacfb\Aviao.java

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda



```
CursoJavaCFB.java x Aereo.java x Aviao.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 abstract class Aviao {
4     public int tipo;
5     public String nome;
6     public boolean ligado;
7     public int velMax;
8     public int vel;
9     public int altitude;
10    public int altMax;
11    public int fatorVel;
12
13    public Aviao(int tipo){
14        this.ligado=false;
15        this.vel=0;
16        this.altitude=0;
17        this.tipo=tipo;
18        if(this.tipo == 1){
19            this.nome="Monomotor";
20            this.velMax=200;
21            fatorVel=2;
22            altMax=5000;
23        }else if(this.tipo == 2){
24            this.nome="Jato";
25            this.velMax=600;
26            fatorVel=10;
27            altMax=30000;
28        }else if(this.tipo == 3){
29            this.nome="Caça";
30            this.velMax=1000;
31            fatorVel=50;
32            altMax=50000;
33        }else{
34            this.nome="Experimental";
35            this.velMax=400;
36            fatorVel=5;
37            altMax=10000;
38        }
39    }
40
41 }
```

A classe “Aviao” será nossa classe base, do qual iremos criar os tipos de aviões, não será permitido declarar um novo objeto diretamente do tipo “Aviao”, usaremos esta classe para ser herdada por outras classes de tipos de aviões.

Note que no construtor precisaremos informar um valor para o tipo do avião e por este tipo iremos configurar as propriedades nome, velMax, fatorVel, altMax, repare que fatorVel e altMax estão sem o operador “this”, para mostrar que, neste caso, não faz diferença.

Agora vamos criar uma classe que irá se chamar “Monomotor”.

Nossa classe “Monomotor” irá herdar a classe “Aviao” e irá implementar a interface “Aereo”.



```
CursoJavaCFB.java x Aereo.java x Aviao.java x Monomotor.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Monomotor extends Aviao implements Aereo{
4
5     public Monomotor(int tipo) {
6         super(tipo);
7     }
8
9     @Override
10    public void maisVelocidade() {
11
12    }
13
14    @Override
15    public void menosVelocidade() {
16
17    }
18
19    @Override
20    public void subir() {
21
22    }
23
24    @Override
25    public void descer() {
26
27    }
28
29    @Override
30    public void virar(int dir) {
31
32    }
33
34    @Override
35    public void ligar() {
36
37    }
38
39    @Override
40    public void desligar() {
41
42    }
43
44 }
45
```

Uma observação muito importante, como nossa classe “Monomotor” está implementando a interface “Aereo” é obrigatório que implementemos “todos” os métodos assinados na interface “Aereo”. Note ainda que usamos “@Override” na implementação dos métodos.

Note que na classe construtor foi chamado o método “super” passando como parâmetro o tipo, esta rotina nada mais faz além de chamar o construtor da super classe de “Monomotor”, sua classe pai, que é a classe “Aviao”, lembra-se do construtor da classe “Aviao”? que necessita do parâmetro “tipo”!

Vamos implementar as funcionalidades dos métodos.





```
5 public Monomotor(int tipo) {
6     super(tipo);
7 }
8
9 @Override
10 public void maisVelocidade() {
11     if((super.ligado)&&(super.vel < super.velMax)){
12         super.vel+=super.fatorVel;
13     }
14 }
15
16 @Override
17 public void menosVelocidade() {
18     if(super.vel > 0){
19         super.vel-=10;
20     }else{
21         super.vel=0;
22     }
23 }
24
25 @Override
26 public void subir() {
27     if((super.ligado)&&(super.altitude < super.altMax)){
28         super.altitude+=super.fatorVel;
29     }
30 }
31
32 @Override
33 public void descer() {
34     if(super.altitude > 0){
35         super.altitude-=10;
36     }else{
37         super.altitude=0;
38     }
39 }
40
41 @Override
42 public void ligar() {
43     super.ligado=true;
44 }
45
46 @Override
47 public void desligar() {
48     super.ligado=false;
49 }
```

Vamos criar mais uma classe chamada “Jato”.



New Classe Java ✕

**Etapas**

- Escolher Tipo de Arquivo
- Nome e Localização**

**Nome e Localização**

Nome da Classe:

Jato

Projeto:

CursoJavaCFB

Localização:

Pacotes de Códigos-fonte

Paquete:

cursojavacfb

Arquivo Criado:

\\Desktop\\Meus Cursos\\Java\\CursoJavaCFB\\src\\cursojavacfb\\Jato.java

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

Que também herdará a classe “Aviao” e irá implementar a interface “Aereo”.



```
CursoJavaCFB.java x Aereo.java x Aviao.java x Monomotor.java x Jato.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Jato extends Aviao implements Aereo{
4
5     public Jato(int tipo) {
6         super(tipo);
7     }
8
9     @Override
10    public void maisVelocidade() {
11        if ((super.ligado) && (super.vel < super.velMax)) {
12            super.vel += super.fatorVel;
13        }
14    }
15
16    @Override
17    public void menosVelocidade() {
18        if (super.vel > 0) {
19            super.vel -= 10;
20        } else {
21            super.vel = 0;
22        }
23    }
24
25    @Override
26    public void subir() {
27        if ((super.ligado) && (super.altitude < super.altMax)) {
28            super.altitude += super.fatorVel;
29        }
30    }
31
32    @Override
33    public void descer() {
34        if (super.altitude > 0) {
35            super.altitude -= 10;
36        } else {
37            super.altitude = 0;
38        }
39    }
40
41    @Override
42    public void ligar() {
43        super.ligado = true;
44    }
45
46    @Override
47    public void desligar() {
48        super.ligado = false;
49    }
50
51 }
```

Note que estes métodos são os mesmos nas classes “Monomotor” e “Jato”, já que os métodos são os mesmos podemos alterar nosso programa implementando a interface na classe “Aviao”.

Vamos alterar.

Classe “Monomotor”.



```
CursoJavaCFB.java x Aereo.java x Aviao.java x Monomotor.java x Jato.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Monomotor extends Aviao{
4
5     public Monomotor(int tipo) {
6         super(tipo);
7     }
8
9 }
10
```

Classe “Jato”.

```
CursoJavaCFB.java x Aereo.java x Aviao.java x Monomotor.java x Jato.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Jato extends Aviao{
4
5     public Jato(int tipo) {
6         super(tipo);
7     }
8
9 }
10
```

Veja que nas classes “Monomotor” e “Jato” iremos implementar somente métodos futuros que serão exclusivos de cada classe, se existirem métodos exclusivos para “Monomotor” e para “Jato”.

Classe “Aviao”.



```
CursoJavaCFB.java x Aereo.java x Aviao.java x Monomotor.java x Jato.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3
4 abstract class Aviao implements Aereo{
5     public int tipo;
6     public String nome;
7     public boolean ligado;
8     public int velMax;
9     public int vel;
10    public int altitude;
11    public int altMax;
12    public int fatorVel;
13
14    public Aviao(int tipo){
15        this.ligado=false;
16        this.vel=0;
17        this.altitude=0;
18        this.tipo=tipo;
19        if(this.tipo == 1){
20            this.nome="Monomotor";
21            this.velMax=200;
22            fatorVel=2;
23            altMax=5000;
24        }else if(this.tipo == 2){
25            this.nome="Jato";
26            this.velMax=600;
27            fatorVel=10;
28            altMax=30000;
29        }else if(this.tipo == 3){
30            this.nome="Caça";
31            this.velMax=1000;
32            fatorVel=50;
33            altMax=50000;
34        }else{
35            this.nome="Experimental";
36            this.velMax=400;
37            fatorVel=5;
38            altMax=10000;
39        }
40    }
41
42    @Override
43    public void maisVelocidade() {
44        if((ligado)&&(vel < velMax)){
45            vel+=fatorVel;
46        }
47    }
48
49    @Override
50    public void menosVelocidade() {
51        if(vel > 0){
52            vel-=10;
53        }else{
54            vel=0;
55        }
56    }
57
58    @Override
59    public void subir() {
60        if((ligado)&&(altitude < altMax)){
61            altitude+=fatorVel;
62        }
63    }
64
65    @Override
66    public void descer() {
67        if(altitude > 0){
68            altitude-=10;
69        }else{
70            altitude=0;
71        }
72    }
73
74    @Override
75    public void ligar() {
76        ligado=true;
77    }
78
79    @Override
80    public void desligar() {
81        ligado=false;
82    }
83 }
```



Podemos também trocar o comando IF por SWITCH.

```
CursoJavaCFB.java x Aereo.java x Aviao.java x
Código-Fonte Histórico
17         this.tipo=tipo;
18
19         switch(this.tipo) {
20             case 1:
21                 this.nome="Monomotor";
22                 this.velMax=200;
23                 this.fatorVel=2;
24                 this.altMax=5000;
25                 break;
26             case 2:
27                 this.nome="Jato";
28                 this.velMax=600;
29                 this.fatorVel=10;
30                 this.altMax=30000;
31                 break;
32             case 3:
33                 this.nome="Caça";
34                 this.velMax=1000;
35                 this.fatorVel=50;
36                 this.altMax=50000;
37                 break;
38             default:
39                 this.nome="Experimental";
40                 this.velMax=400;
41                 this.fatorVel=5;
42                 this.altMax=10000;
43                 break;
44         }
45     }
46 }
```

Com as classes prontas agora podemos declarar e instanciar os objetos no código principal.



```
CursoJavaCFB.java x Aereo.java x Aviao.java x Monomotor.java x Jato.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Aviao av1 = new Monomotor(1);
8         Aviao av2 = new Jato(2);
9
10        System.out.println("Avião.....: " + av1.nome);
11        System.out.println("Altitude máxima..: " + av1.altMax);
12        System.out.println("Altitude atual...: " + av1.altitude);
13        System.out.println("Velocidade máxima: " + av1.velMax);
14        System.out.println("Velocidade atual.: " + av1.vel);
15        System.out.println("Ligado.....: " + av1.ligado);
16
17        av1.ligar();
18        while(av1.vel < 150){
19            av1.maisVelocidade();
20        }
21        while(av1.altitude < 2000){
22            av1.subir();
23        }
24
25        System.out.println("\nNovos valores");
26        System.out.println("Altitude atual...: " + av1.altitude);
27        System.out.println("Velocidade atual.: " + av1.vel);
28
29    }
30
31 }
```

```
Saída - CursoJavaCFB (run) x
run:
Avião.....: Monomotor
Altitude máxima..: 5000
Altitude atual...: 0
Velocidade máxima: 200
Velocidade atual.: 0
Ligado.....: false

Novos valores
Altitude atual...: 2000
Velocidade atual.: 150
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## O método main

Em java além do construtor que é executado sempre que um novo objeto da classe é criado, também existem o método “main” que tem uma importância muito grande.

Um sistema em Java pode ser formado por diversas classes, mas como identificar qual é a classe que será a primeira a ser executada, a classe de carregamento do programa, a classe inicial ou classe de entrada? A resposta é simples! A classe que contém o método de entrada “main”.

Na verdade todas as classes que não possuem o método “main” não podem ser executada diretamente, por linha de comando por exemplo, para isto é necessário que a classe possua o método “main”.

**OBS:** A classe que possuir o método de entrada “main” não necessita de uma classe construtora, afinal o método a ser executado nesta classe não é o construtor e sim o método de entrada “main”.



Então, no código a seguir note que o método “CursoJavaCFB()” não é automaticamente executado como o construtor e sim o método “main”.

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args) {
6
7         System.out.println("Método main");
8
9     }
10
11     public static void CursoJavaCFB() {
12         System.out.println("Construtor padrão");
13     }
14
15 }
```

Saída - CursoJavaCFB (run) ×

```
run:
Método main
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

O método “main” deve seguir algumas regras importantes a serem observadas.

- 1 – Obrigatoriamente deve ter o nome main
- 2 – Deve ser público (public)
- 3 – Deve ser estático (static)
- 4 – Não pode retornar nada, portanto deve ter o retorno como void (void)
- 5 – Deve receber como parâmetro um array de strings chamado “args”

## Enum

Enum basicamente é uma estrutura de enumeração, onde podemos associar elementos e valores enumerados.

Em nosso código de exemplo iremos criar uma estrutura “enum” com tipos de aviões.





```
CursoJavaCFB.java x
Código-Fonte  Histórico
1  package cursojavacfb;
2
3  public class CursoJavaCFB {
4
5      public enum tiposAvioes{
6          Monomotor, Bimotor, Jato, Caça, Experimental;
7      }
8
9      public static void main(String[] args){
10
11          int tpAv;
12
13          tpAv=tiposAvioes.Jato.ordinal();
14
15          System.out.println(tpAv);
16
17      }
18
19  }
```

```
Saída - CursoJavaCFB (run) x
run:
2
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Na linha 13 veja que obtivemos a posição do avião indicado, no caso “Jato”, que está na posição 2, lembrando que o primeiro está na posição zero.

## super

O comando “super” faz uma chamada ao construtor padrão da classe pai, ou uma chamada a algo da super classe (classe pai), que é a classe herdada. Em comparação, “THIS” faz referência a algo da classe atual e “SUPER” faz referência a algo da classe pai.

Vamos ao nosso exemplo.

Vamos criar uma classe chamada “Veiculo” conforme o modelo a seguir.



```
CursoJavaCFB.java x Veiculo.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Veiculo {
4
5     private boolean ligado;
6     private int rodas;
7     private String cor;
8
9     public Veiculo(int rodas, String cor){
10         this.ligado=false;
11         this.rodas=rodas;
12         this.cor=cor;
13     }
14
15     public void ligar(){
16         this.ligado=true;
17     }
18
19     public void desligar(){
20         this.ligado=false;
21     }
22
23     public void imprime(){
24         if(this.rodas == 2){
25             System.out.println("Tipo... Moto");
26         }else if(this.rodas == 4){
27             System.out.println("Tipo... Carro");
28         }
29         System.out.println("Ligado: " + this.ligado);
30         System.out.println("Rodas: " + this.rodas);
31         System.out.println("Cor...: " + this.cor);
32         System.out.println("\n");
33     }
34 }
35
```

Em seguida vamos alterar a classe “Carro”, caso você não tenha criado esta classe é hora de cria-la conforme o código a seguir.

```
CursoJavaCFB.java x Veiculo.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Carro extends Veiculo{
4
5     public Carro(String cor){
6         super(4, cor);
7     }
8
9 }
10
```

Note que a classe “Carro” herda a classe “Veiculo”. No construtor da classe “Carro” iremos informar a cor que será passada ao construtor da classe pai.

Na linha 6 usamos “super” que neste caso faz uma chamada ao construtor da classe pai que é a classe “Veiculo”. Veja que na classe “Veiculo” o construtor recebe dois parâmetros “rodas” e “cor”.

Então lembre-se, sempre que precisar se referir a algo da super classe, use “super”.



OBS: Neste momento será necessário deletar a classe “Caminhao” que criamos anteriormente, pois ela herda a classe “Carro” antiga, então para não termos problemas com esta classe, vamos deletá-la.

Para que possamos entender um pouco mais sobre o uso de “super”, vamos criar mais um método da classe pai “Veiculo” chamado “cfb” como destacado no exemplo a seguir.

```
19 public void desligar(){
20     this.ligado=false;
21 }
22
23 public void imprime(){
24     if(this.rodas == 2){
25         System.out.println("Tipo..: Moto");
26     }else if(this.rodas == 4){
27         System.out.println("Tipo..: Carro");
28     }
29     System.out.println("Ligado: " + this.ligado);
30     System.out.println("Rodas.: " + this.rodas);
31     System.out.println("Cor...: " + this.cor);
32     System.out.println("\n");
33 }
34
35 public void cfb(){
36     System.out.println("Canal Fessor Bruno");
37 }
38
39 }
```

E na classe “Carro” vamos chamar este método direto da classe pai.

```
CursoJavaCFB.java x Veiculo.java x Carro.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Carro extends Veiculo{
4
5     public Carro(String cor){
6         super(4, cor);
7         super.cfb();
8     }
9
10 }
11
```

Vamos criar uma nova classe chamada “Moto”, nos mesmos padrões da classe “Carro”, conforme o código a seguir.

```
CursoJavaCFB.java x Veiculo.java x Carro.java x Moto.java x
Código-Fonte Histórico
1 package cursojavacfb;
2
3 public class Moto extends Veiculo{
4
5     public Moto(String cor){
6         super(2, cor);
7         super.cfb();
8     }
9
10 }
11
```

Vamos agora voltar ao arquivo do código principal “CursoJavaCFB” e instanciar objetos do tipo “Carro” e “Moto”.



The screenshot shows an IDE with four tabs: CursoJavaCFB.java, Veiculo.java, Carro.java, and Moto.java. The 'CursoJavaCFB.java' tab is active, displaying the following code:

```
1 package cursojavacfb;
2
3 public class CursoJavaCFB {
4
5     public static void main(String[] args){
6
7         Veiculo v1 = new Carro("Vermelho");
8         Veiculo v2 = new Moto("Preto");
9
10        v1.ligar();
11
12        v1.imprime();
13        v2.imprime();
14    }
15 }
16
17 }
```

Below the code editor, the 'Saída - CursoJavaCFB (run)' window shows the output of the program:

```
run:
Canal Fessor Bruno
Canal Fessor Bruno
Tipo.: Carro
Ligado: true
Rodas.: 4
Cor...: Vermelho

Tipo.: Moto
Ligado: false
Rodas.: 2
Cor...: Preto
```

Note que as primeiras impressões vem do construtor das classes, que chamam o método “cfb” na classe pai.

Note também que o objeto “v1” está ligado (true) e “v2” está desligado (false), isso porque chamamos o método “ligar” do objeto “v1”.

## Considerações finais

Neste material você teve acesso ao conteúdo principal básico para se aprender Java, este foi seu impulso inicial na linguagem, ainda há um longo caminho a seguir, em um próximo material sobre Java iremos abordar sobre a parte gráfica da programação Java.

Até mais...