

# Kotlin



Prof. Dr. João Paulo Lemos Escola

Copyright© 2022

# Conteúdo

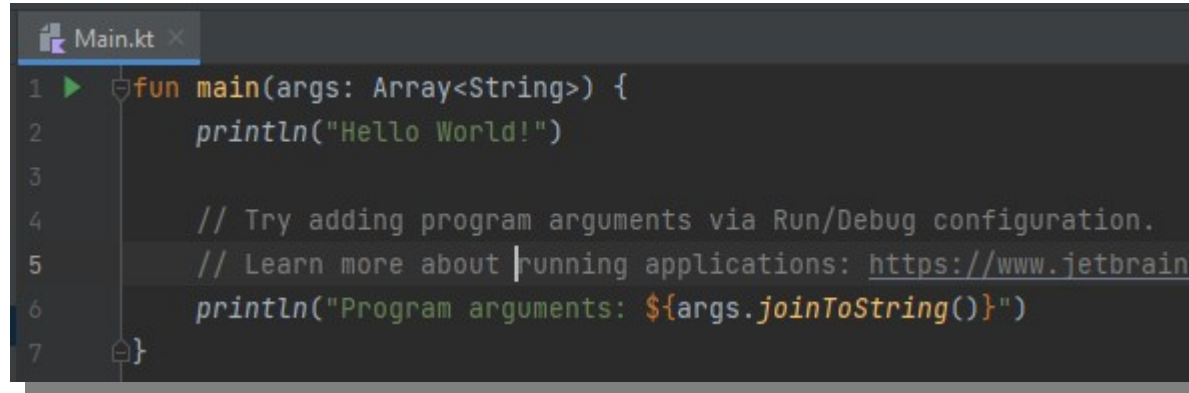
- Histórico da linguagem;
- Instalação da IDE;
- Sintaxe básica;
- Iterações (laços de repetição);
- Classes / métodos / herança / pacotes / sobrecarga;
- Conversões de tipos;
- Exceções

# Histórico

- 2005: Google lançou a plataforma Android;
- 2010: Máquina virtual Dalvik;
- 2010: Oracle processou a Google;
- Criada em 2011 pela JetBrains;
- Tornou-se OpenSource em 2012;
- Em 2017 o Kotlin foi incluído no Android;
- Em 2019 se tornou substituta da linguagem Java na plataforma Android.

# Arquivo .kt

- Os arquivos Kotlin tem a extensão “.kt”;



```
1  fun main(args: Array<String>) {  
2      println("Hello World!")  
3  
4      // Try adding program arguments via Run/Debug configuration.  
5      // Learn more about running applications: https://www.jetbrains  
6      println("Program arguments: ${args.joinToString()}")  
7  }
```

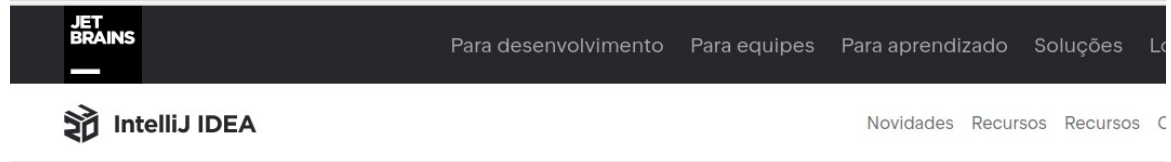
# Características da linguagem

- Multiplataforma e OpenSource;
- Interoperável com Java;
- Utiliza a JVM Java;
- Busca simplificação do código;
- Tipagem dinâmica;
- Paradigmas: Orientação a Objetos e Funcional;
- Não necessita ponto-e-vírgula ao final do comando.

# Instalação da IDE

- [jetbrains.com/pt-br/idea/download/](https://jetbrains.com/pt-br/idea/download/)
- Baixe a versão zip:

<https://www.jetbrains.com/pt-br/idea/download/#section=windows>



Versão: 2021.3  
Build: 213.5744.223  
29 de novembro de 2021  
[Notas de lançamento](#)

## Baixar IntelliJ IDEA

[Windows](#) [macOS](#) [Linux](#)

### Ultimate

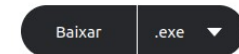
Para desenvolvimento Web e corporativo



Avaliação gratuita por 30 dias

### Community

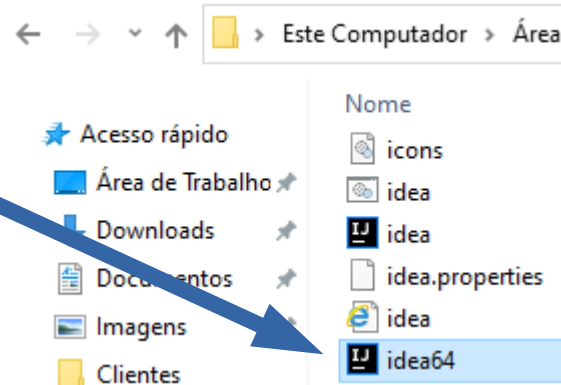
Para desenvolvimento JVM e Android



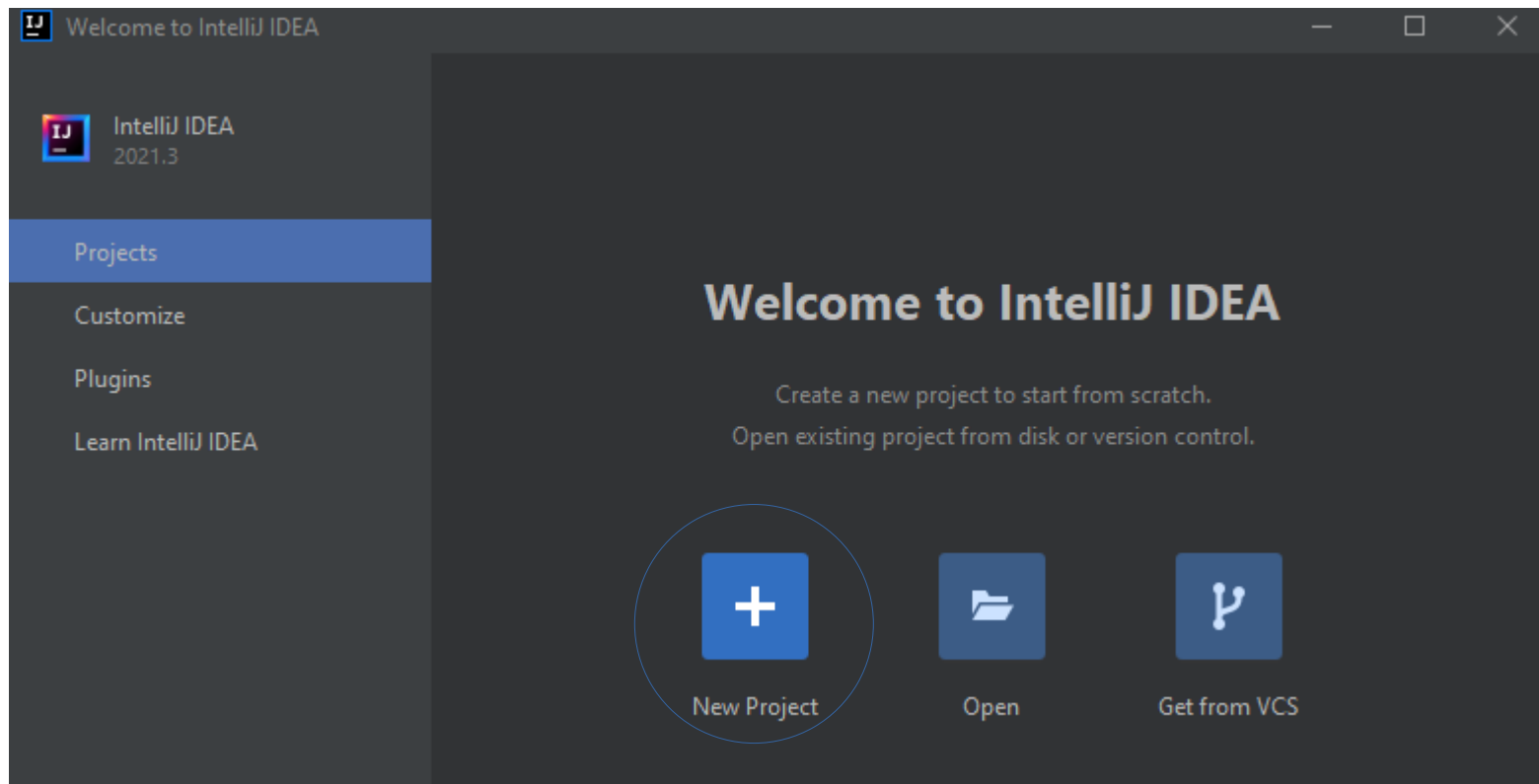
Gratuito, com base em open source

# Executando a IDE

- Execute o aplicativo pelo arquivo executável “idea64” da pasta “bin”.



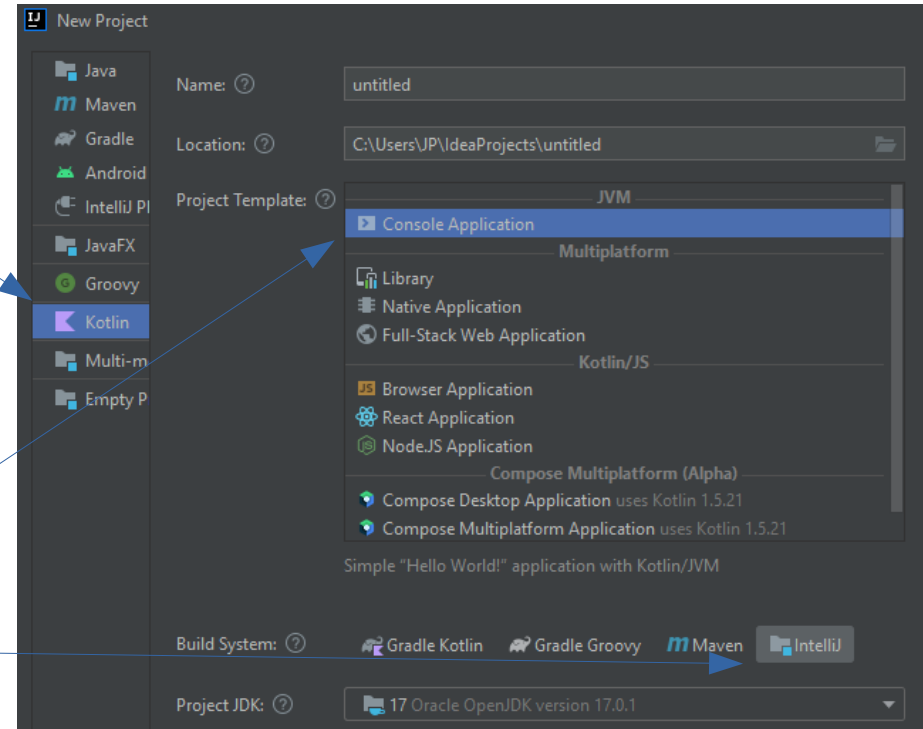
# Crie um novo projeto





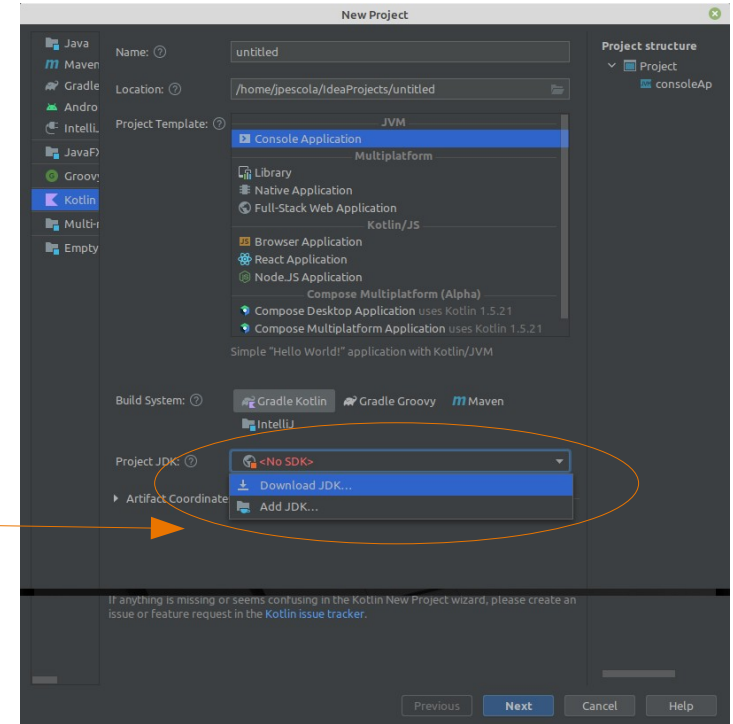
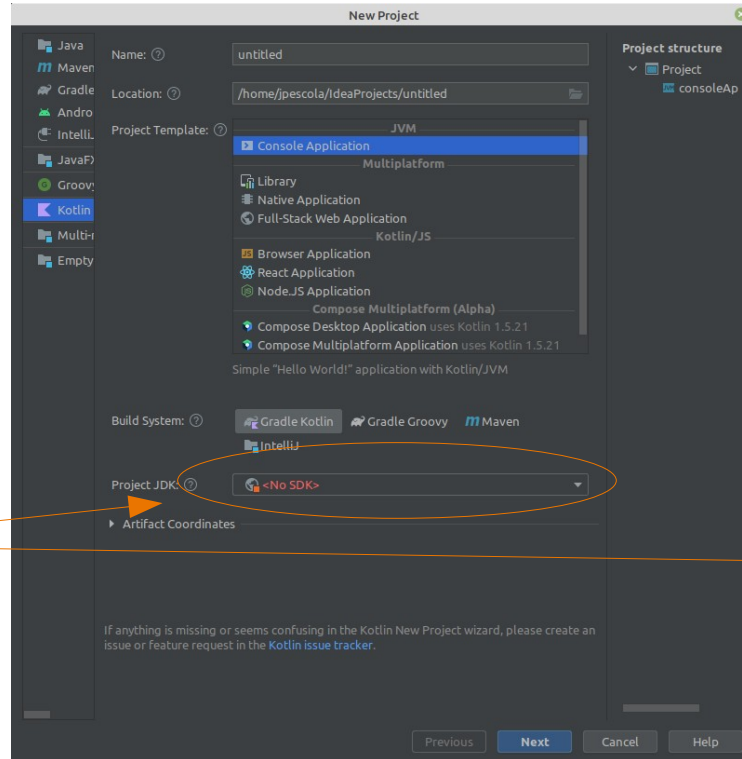
# Novo projeto Kotlin

- Selecione 'Kotlin';
- Aplicação para console;
- Construtor local.



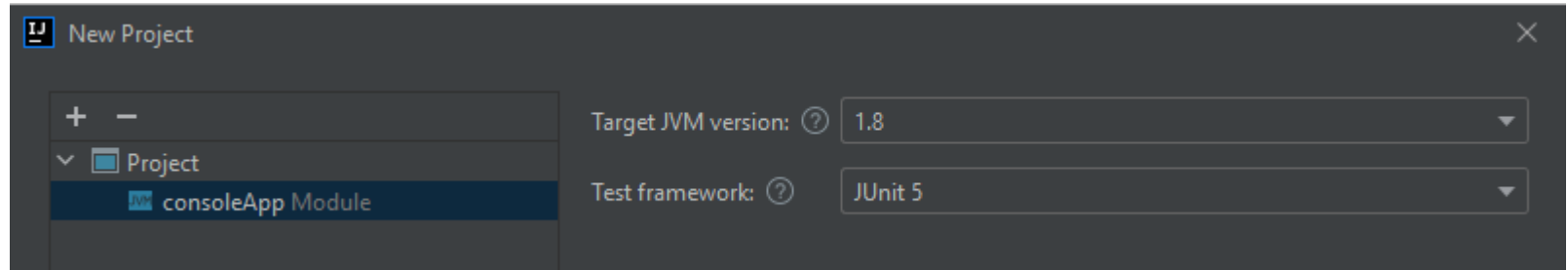
# JDK no IntelliJ

Baixe o JDK  
caso necessário



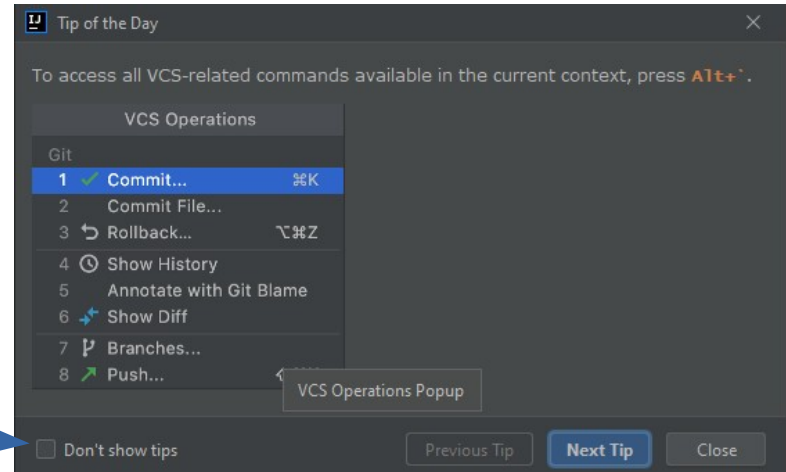
# Versão JVM e JUnit

- JUnit é um framework de testes para Java;
- Utilizado em testes unitários;
- Deixe as opções padrão e escolha “Finish”.



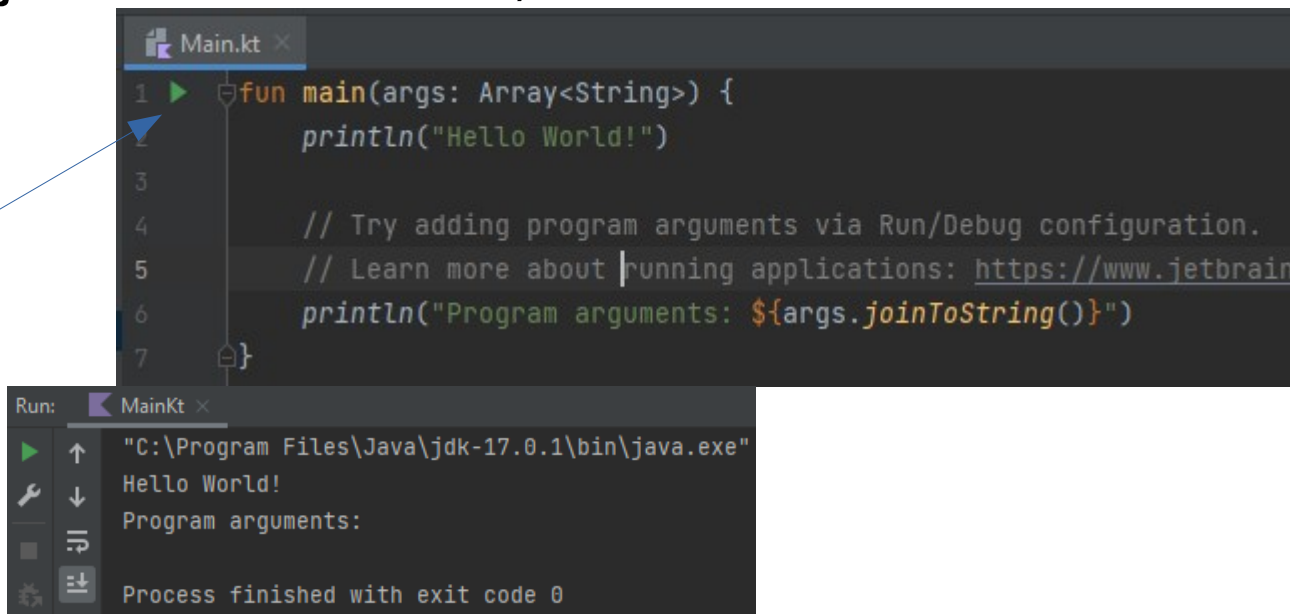
# Dicas do dia

- Essa janela mostra dicas diferentes todos os dias;
- Ideal para aprender sempre recursos novos da IDE;
- Desabilite se desejar:



# Executando a aplicação

- Veja que a IDE criou uma função chamada 'main' (principal);
- Essa é a primeira função executada em um programa Kotlin;
- A IDE criou um projeto 'Hello World!';
- Execute o projeto clicando no play e escolhendo 'run';



The screenshot displays an IDE window with a file named 'Main.kt'. The code defines a `main` function that prints 'Hello World!' and then 'Program arguments: ' followed by the joined string of command-line arguments. A blue arrow points from the text 'clicando no play' in the list to the green play button icon on the first line of the code. Below the code editor, the 'Run' console shows the execution path, the output 'Hello World!', the prompt for arguments, and a confirmation that the process finished with exit code 0.

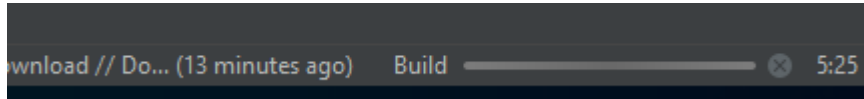
```
1 ▶ fun main(args: Array<String>) {  
2     println("Hello World!")  
3  
4     // Try adding program arguments via Run/Debug configuration.  
5     // Learn more about running applications: https://www.jetbrains  
6     println("Program arguments: ${args.joinToString()}")  
7 }
```

Run: MainKt ×

↑ "C:\Program Files\Java\jdk-17.0.1\bin\java.exe"  
↓ Hello World!  
Program arguments:  
Process finished with exit code 0

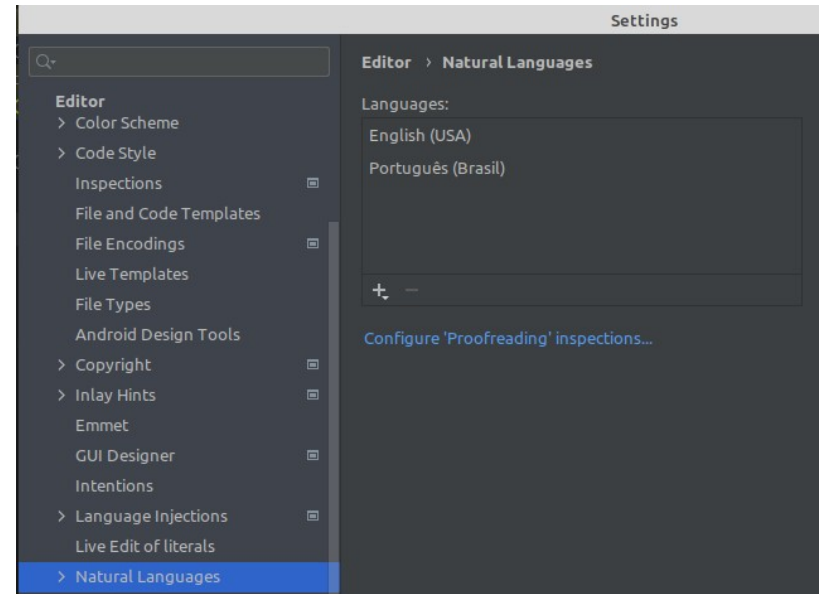
# Demorando muito?

- Caso o Build esteja demorando muito, reinicie o computador;



# Dicionários

- Vamos incluir o dicionário Português (Brasil);
- Clique em File > Settings > Editor > Natural Languages;
- Clique em “+” e adicione Português (Brasil).



# Estrutura da função main

- fun: declaração de uma função
- main: nome da função
- args: nome do vetor de argumentos
- Array<String>: tipo do vetor de argumentos
- println: imprime e quebra linha ao final.

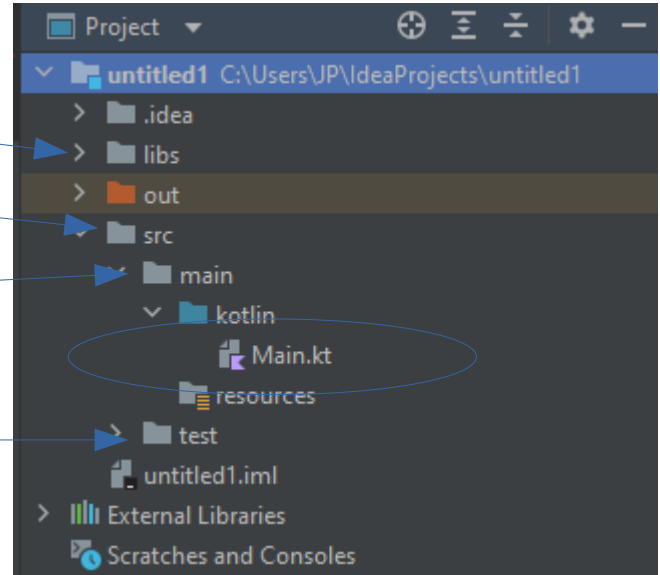
A screenshot of a code editor window titled 'Main.kt'. The code is written in Kotlin and defines a 'main' function. The function signature is 'fun main(args: Array<String>) {'. The first line inside the function is 'println("Hello World!")'. There are two comment lines: '// Try adding program arguments via Run/Debug configuration.' and '// Learn more about running applications: https://www.jetbrains.com/idea/running-applications/'. The final line inside the function is 'println("Program arguments: \${args.joinToString()}")'. The function is closed with a closing curly brace '}' on line 7. The code is color-coded: 'fun' is orange, 'main' is orange, 'args' is orange, 'Array' is orange, '<' is orange, 'String' is orange, '>' is orange, '{' is orange, 'println' is green, 'Hello World!' is green, '//' is grey, 'Try adding program arguments via Run/Debug configuration.' is grey, 'Learn more about running applications:' is grey, 'https://www.jetbrains.com/idea/running-applications/' is blue, 'println' is green, 'Program arguments:' is green, '\${args.joinToString()}' is green, and '}' is orange.

```
1  fun main(args: Array<String>) {  
2      println("Hello World!")  
3  
4      // Try adding program arguments via Run/Debug configuration.  
5      // Learn more about running applications: https://www.jetbrains.com/idea/running-applications/  
6      println("Program arguments: ${args.joinToString()}")  
7  }
```



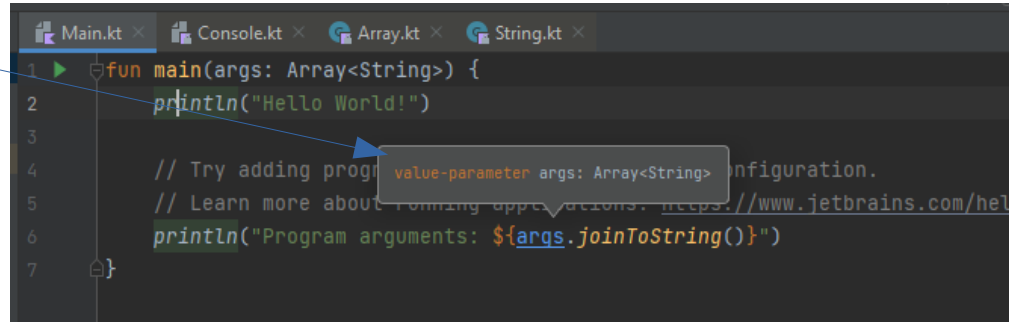
# Estrutura do projeto

- Bibliotecas;
- Source (arquivos-fonte);
- Principal;
- Testes.



# Funcionalidades da IDE

- CTRL+Click: direciona para a declaração;
- CTRL+cursor: mostra um rótulo de detalhes.

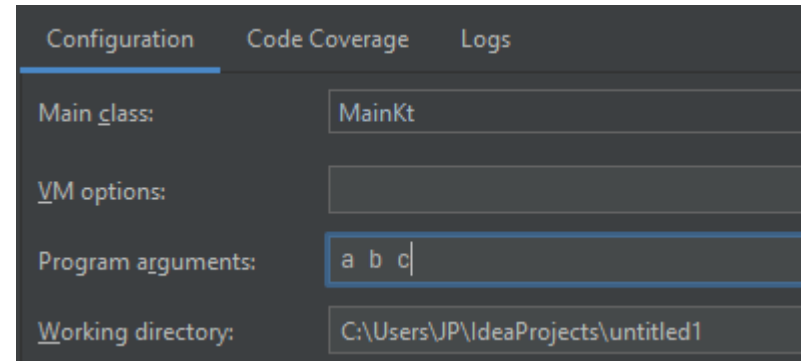


# Teclas de atalho

- Ctrl+D: duplica a linha atual;
- Ctrl+X: recorta (exclui) a linha atual;
- Ctrl+/: adiciona um comentário;
- Ctrl+Alt+L: formatar o código;
- Shift+Alt+seta: move a linha atual;
- Shift+F10: compila e executa o projeto atual.

# Argumentos

- Podemos configurar os argumentos que serão passados na execução para o programa;
- Simulando os argumentos da linha de comandos;
- Menu:  
Run / Edit configurations /  
Program arguments



# Hello World simplificado

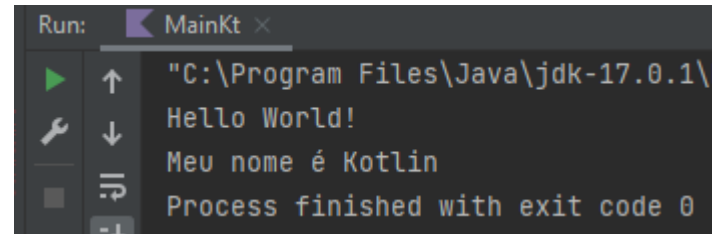
- A função main pode ser utilizada sem declaração de parâmetros.

```
1  ▶ fun main() {  
2      println("Hello World!")  
3  }
```

# Imprimindo

- `print`: imprime sem quebra de linha;
- `println`: imprime com quebra de linha;

```
fun main() {  
    println("Hello World!")  
    print("Meu nome é ")  
    print("Kotlin")  
}
```



# Comentários

- Comentário de uma linha:

`//` utilize duas barras

- Comentário de múltiplas linhas:

`/*` começa com barra-asterisco e  
termina com asterisco-barra `*/`



```
fun main() {  
    // comentário de uma linha  
    println("Hello World!")  
    /* comentário  
    de várias  
    linhas */  
    print("Meu nome é ")  
    print("Kotlin")  
}
```

@InlineOnly

# Variáveis e constantes

- Variáveis podem ser alteradas em outras partes do código:

```
var a = 1
```

```
a = 2
```

- Constantes são definidas uma única vez, não podendo ser alteradas:

```
val a = 1
```

```
// a = 2 // não permitido
```

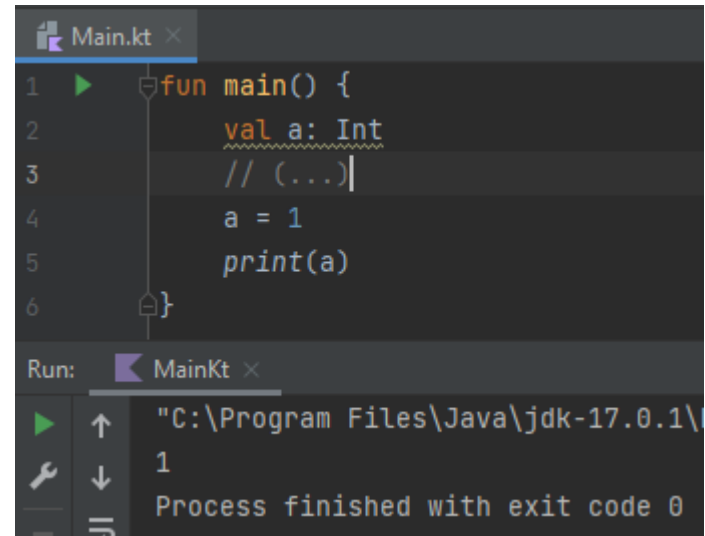


# Você viu?

- Percebeu que não foi necessário definir o tipo para a variável 'a'?
- Motivo:
  - Em Kotlin não é necessário definir os tipos de dados;
  - Tipagem dinâmica.

# Declaração de tipos

- Declaramos o tipo da variável após o nome da variável, separando por 'dois pontos';
- Quando queremos declarar variáveis vazias somos obrigados a definir o tipo:



The screenshot shows an IDE window titled 'Main.kt' with the following Kotlin code:

```
1 fun main() {  
2     val a: Int  
3     // (...) |  
4     a = 1  
5     print(a)  
6 }
```

Below the code editor, the 'Run' tab is active, showing the execution output:

```
Run: MainKt x  
↑ "C:\Program Files\Java\jdk-17.0.1\  
1  
↓ Process finished with exit code 0
```

# Tipos de dados

```
fun main() {  
    val myNum = 5           // Int  
    val myDoubleNum = 5.99 // Double  
    val myLetter = 'D'      // Char  
    val myBoolean = true    // Boolean  
    val myText = "Hello"    // String  
}
```

# Operadores Aritméticos

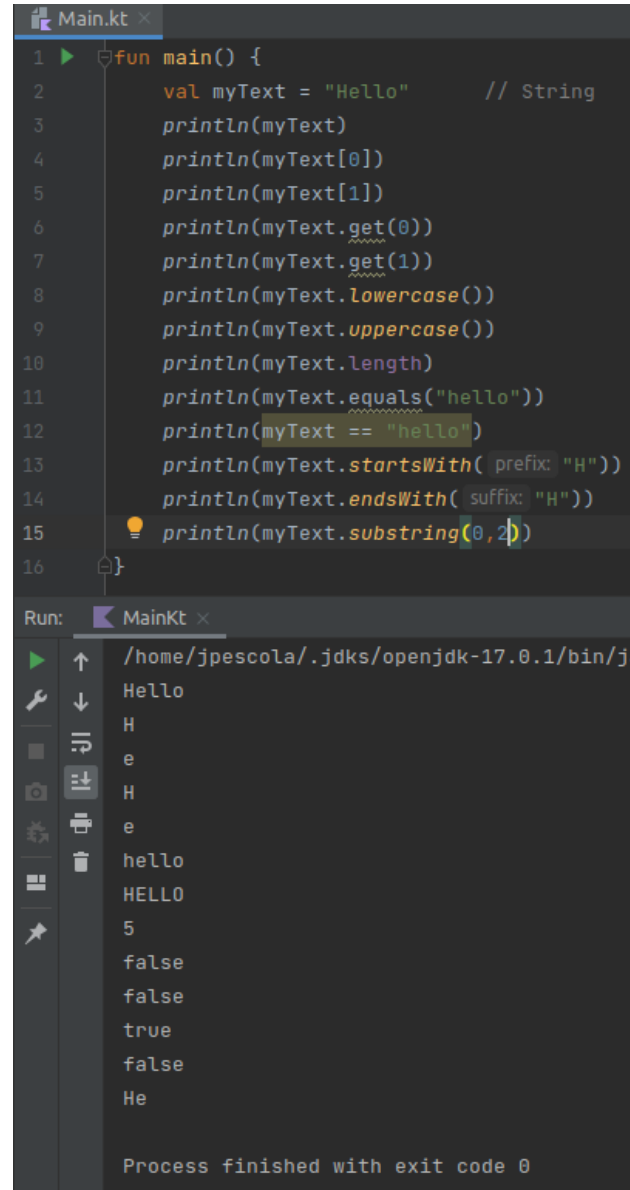
Operador	Descrição	Exemplo
+	Adição	<code>var a = x + y</code>
-	Subtração	<code>var a = x - y</code>
*	Multiplicação	<code>var a = x * y</code>
/	Divisão	<code>var a = x / y</code>
%	Módulo (resto da divisão)	<code>var a = x % y</code>
++	Incremento de 1	<code>x++</code>
--	Decremento	<code>x--</code>
+=	Incremento de N	<code>x+=2</code>
-=	Decremento de N	<code>x-=2</code>
*=	Substituição pelo produto	<code>x*=2</code>
/=	Substituição pela divisão	<code>x/=2</code>
%=	Substituição pelo módulo	<code>x%=2</code>

# Operadores lógicos

Operador	Descrição	Exemplo
==	igual	<code>x==y</code>
!=	diferente	<code>x!=y</code>
>	maior	<code>x &gt; y</code>
<	menor	<code>x &lt; y</code>
>=	maior ou igual	<code>x &gt;= y</code>
<=	menor ou igual	<code>x &lt;= y</code>
&&	operador 'e'	<code>x==0 &amp;&amp; y==0</code>
	operador 'ou'	<code>x==0    y==0</code>
!	operador 'não'	<code>!a.isEmpty()</code>

# String

- String é um tipo de dado que se comporta como um vetor de caracteres;
- Permite armazenar palavras, frases ou textos;



The screenshot displays a Kotlin program in an IDE. The code defines a `main` function that creates a `String` variable `myText` with the value "Hello". It then performs various operations on this string, including printing it, accessing individual characters, converting to lowercase and uppercase, checking length, and using `startsWith`, `endsWith`, and `substring` methods. The output window below shows the results of these operations.

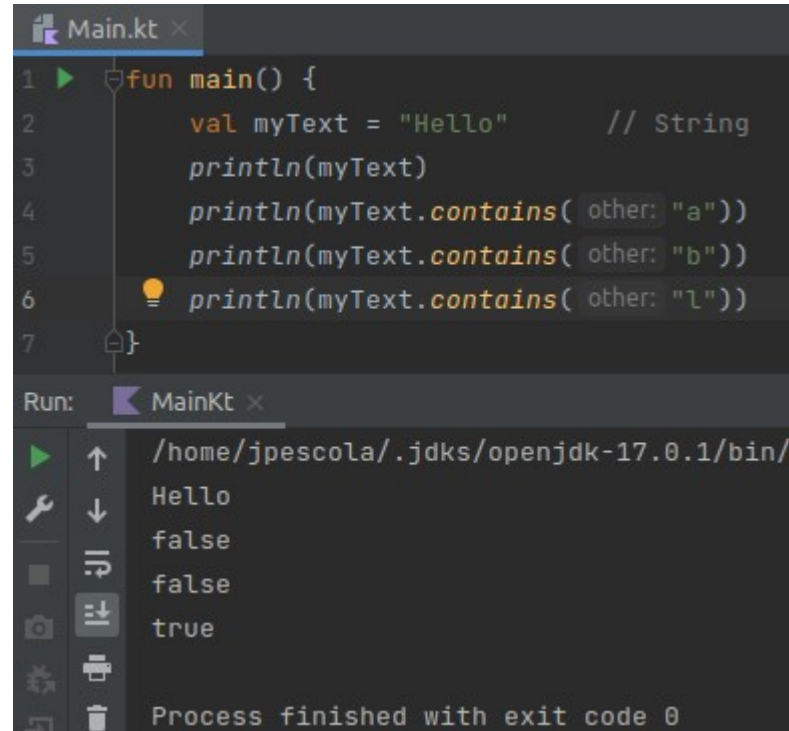
```
1 fun main() {  
2     val myText = "Hello" // String  
3     println(myText)  
4     println(myText[0])  
5     println(myText[1])  
6     println(myText.get(0))  
7     println(myText.get(1))  
8     println(myText.lowercase())  
9     println(myText.uppercase())  
10    println(myText.length)  
11    println(myText.equals("hello"))  
12    println(myText == "hello")  
13    println(myText.startsWith( prefix: "H"))  
14    println(myText.endsWith( suffix: "H"))  
15    println(myText.substring(0, 2))  
16 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/bin/j  
Hello  
H  
e  
H  
e  
hello  
HELLO  
5  
false  
false  
true  
false  
He

Process finished with exit code 0

# Pesquisa em Strings



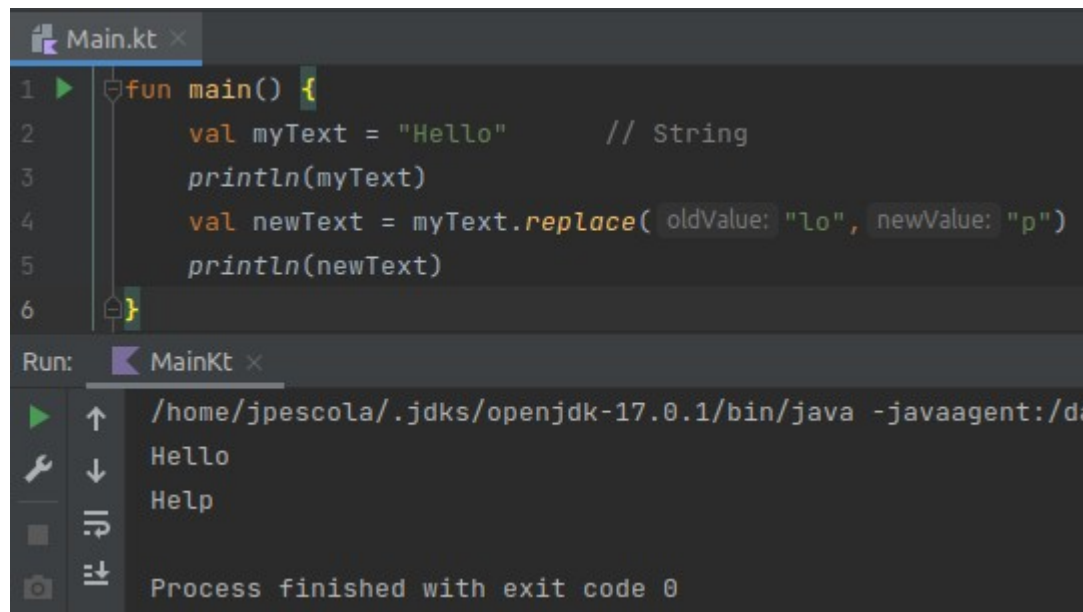
The image shows a screenshot of an IDE with a Kotlin file named `Main.kt`. The code defines a `main` function that creates a `String` variable `myText` with the value "Hello". It then uses the `contains` method to check if the string contains the characters 'a', 'b', and 'l'. The output of the program is shown in the Run window, displaying the results of these checks: 'Hello', 'false', 'false', and 'true'. The process finished with exit code 0.

```
1 fun main() {  
2     val myText = "Hello" // String  
3     println(myText)  
4     println(myText.contains( other: "a"))  
5     println(myText.contains( other: "b"))  
6     println(myText.contains( other: "l"))  
7 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/bin/  
Hello  
false  
false  
true  
Process finished with exit code 0

# Substituição em Strings



```
1 fun main() {  
2     val myText = "Hello"    // String  
3     println(myText)  
4     val newText = myText.replace( oldValue: "lo", newValue: "p")  
5     println(newText)  
6 }
```

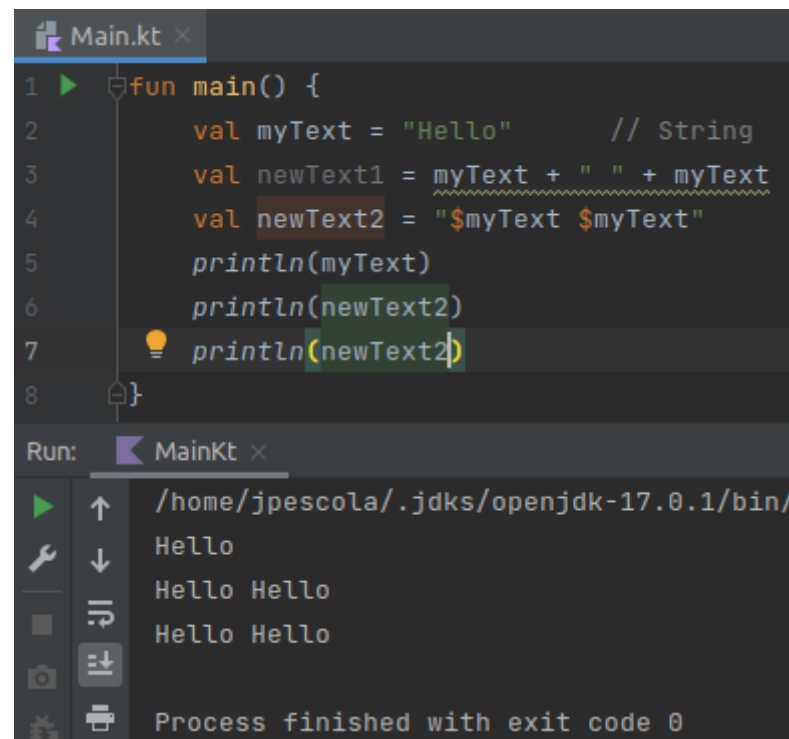
Run: MainKt x

↑ /home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:/da  
Hello  
↓ Help  
Process finished with exit code 0



# Concatenação e Interpolação

- Junção de duas ou mais Strings;
- Na concatenação utilizamos o operador '+';
- Na interpolação utilizamos o operador '\$';



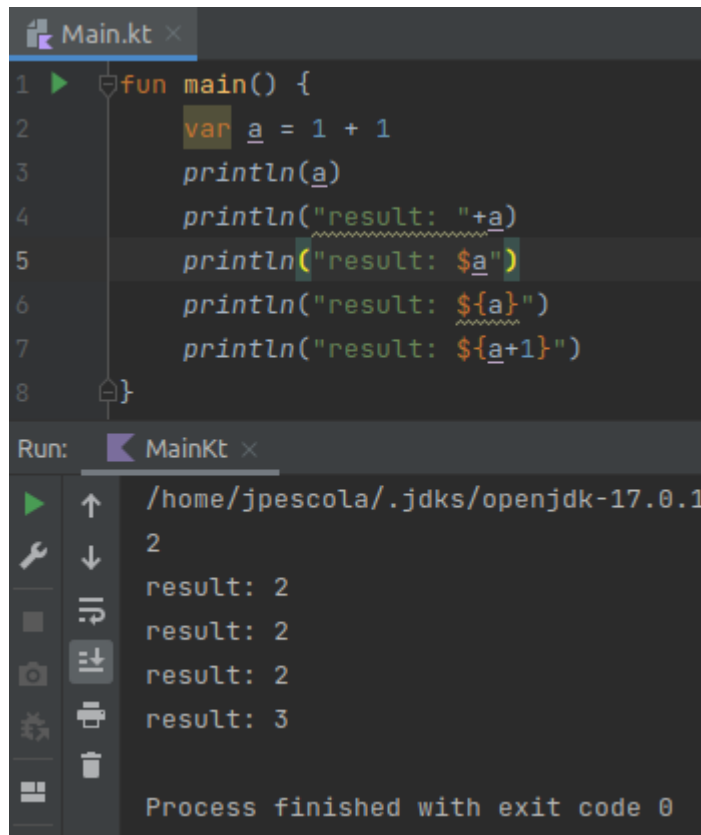
```
1 fun main() {  
2     val myText = "Hello" // String  
3     val newText1 = myText + " " + myText  
4     val newText2 = "$myText $myText"  
5     println(myText)  
6     println(newText1)  
7     println(newText2)  
8 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/bin/  
Hello  
Hello Hello  
Hello Hello  
Process finished with exit code 0

# Também possível com variáveis

- As chaves permitem inclusão de blocos de código.



The screenshot displays an IDE window titled 'Main.kt' containing the following Kotlin code:

```
1 fun main() {  
2     var a = 1 + 1  
3     println(a)  
4     println("result: "+a)  
5     println("result: $a")  
6     println("result: ${a}")  
7     println("result: ${a+1}")  
8 }
```

Below the code editor, the 'Run' console shows the output of the program:

```
Run: MainKt x  
/home/jpescola/.jdk/openjdk-17.0.1  
2  
result: 2  
result: 2  
result: 2  
result: 3  
Process finished with exit code 0
```

# Estruturas de decisão

```
Main.kt x
1 fun main() {
2     val nota = 4
3     var resultado: String
4     if (nota >= 6) {
5         resultado = "Aprovado"
6     }
7     else if (nota < 4) {
8         resultado = "Reprovado"
9     }
10    else {
11        resultado = "Recuperação"
12    }
13    println(resultado)
14 }
```

Run: MainKt x

↑ /home/jpescola/.jdk/openjdk-17.0.1/

↓ Recuperação

Process finished with exit code 0

```
Main.kt x
1 fun main() {
2     val nota = 4
3     var resultado: String
4     if (nota >= 6)
5         resultado = "Aprovado"
6     else if (nota < 4)
7         resultado = "Reprovado"
8     else
9         resultado = "Recuperação"
10    println(resultado)
11 }
```

Run: MainKt x

↑ /home/jpescola/.jdk/openjdk-17.0.1/

↓ Recuperação

Process finished with exit code 0

```
Main.kt x
1 fun main() {
2     val nota = 4
3     if (nota >= 6)
4         println("Aprovado")
5     else if (nota < 4)
6         println("Reprovado")
7     else
8         println("Recuperação")
9 }
```

Run: MainKt x

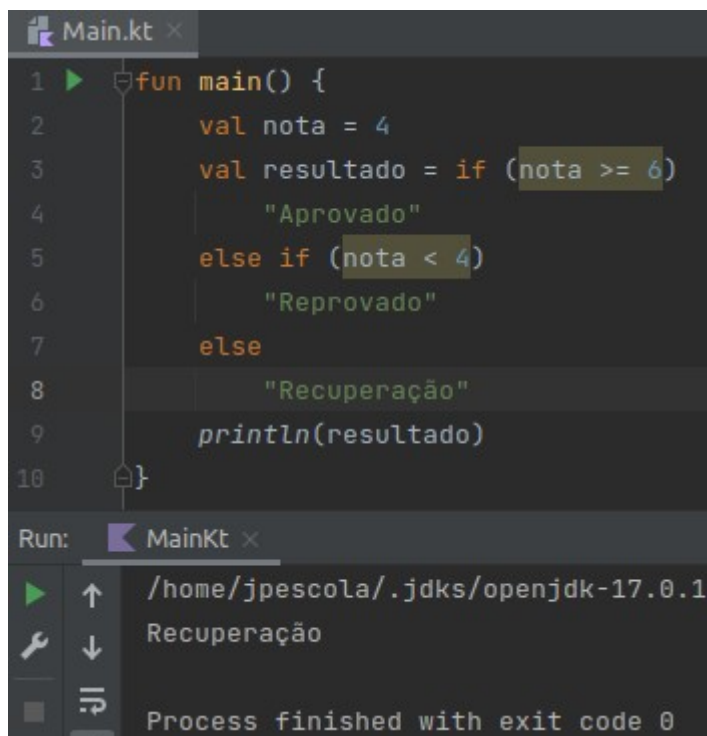
↑ /home/jpescola/.jdk/openjdk-17.0.1/

↓ Recuperação

Process finished with exit code 0

# Expressões

- São blocos de código que retornam valor.



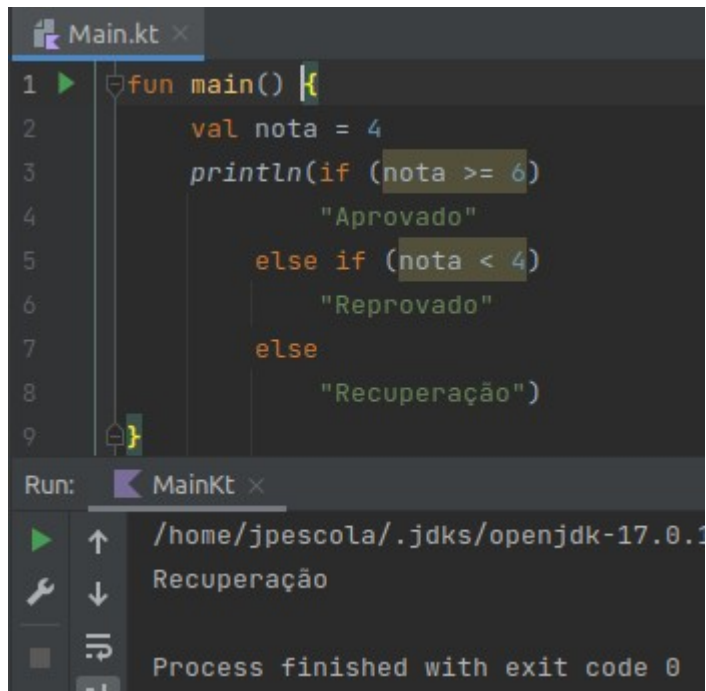
The image shows a screenshot of an IDE with a Kotlin file named `Main.kt`. The code defines a `main` function that assigns a value to `nota` and then uses an `if-else` expression to determine the result based on the value of `nota`. The code is as follows:

```
1 fun main() {  
2     val nota = 4  
3     val resultado = if (nota >= 6)  
4         "Aprovado"  
5     else if (nota < 4)  
6         "Reprovado"  
7     else  
8         "Recuperação"  
9     println(resultado)  
10 }
```

Below the code editor, the `Run` tab is visible, showing the execution of `MainKt`. The output indicates that the program ran successfully on a Linux system using OpenJDK 17.0.1, and the result printed was `Recuperação`. The process finished with an exit code of 0.

# Encurtando o código

- Ainda podemos diminuir o número de linhas:



```
1 fun main() {  
2     val nota = 4  
3     println(if (nota >= 6)  
4         "Aprovado"  
5     else if (nota < 4)  
6         "Reprovado"  
7     else  
8         "Recuperação")  
9 }
```

Run: MainKt x

Process finished with exit code 0



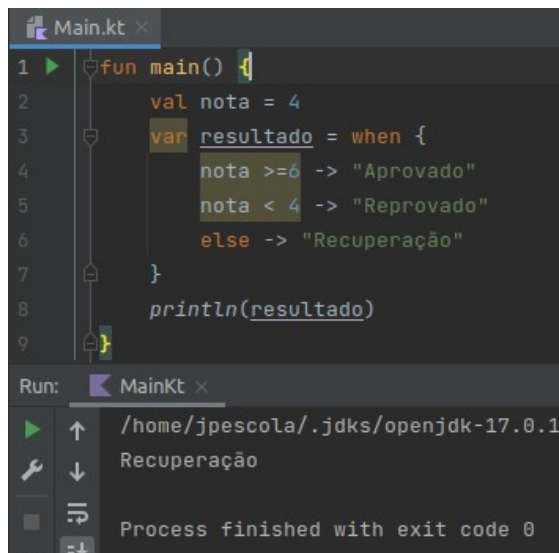
```
1 fun main() {  
2     val nota = 4  
3     println(if (nota >= 6) "Aprovado" else if (nota < 4) "Reprovado" else "Recuperação")  
4 }
```

Run: MainKt x

Process finished with exit code 0

# When

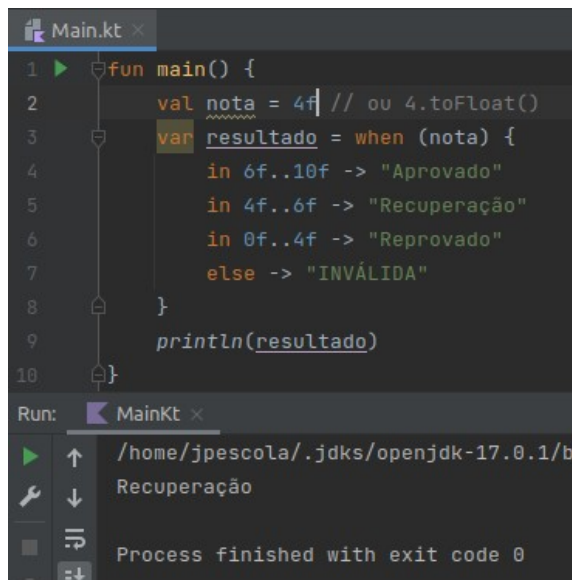
- Em Kotlin não há a estrutura switch / case;
- Mas temos a estrutura 'when':



```
1 fun main() {  
2     val nota = 4  
3     var resultado = when {  
4         nota >= 6 -> "Aprovado"  
5         nota < 4 -> "Reprovado"  
6         else -> "Recuperação"  
7     }  
8     println(resultado)  
9 }
```

Run: MainKt x

Process finished with exit code 0



```
1 fun main() {  
2     val nota = 4f // ou 4.toFloat()  
3     var resultado = when (nota) {  
4         in 6f..10f -> "Aprovado"  
5         in 4f..6f -> "Recuperação"  
6         in 0f..4f -> "Reprovado"  
7         else -> "INVÁLIDA"  
8     }  
9     println(resultado)  
10 }
```

Run: MainKt x

Process finished with exit code 0



```
1 fun main() {  
2     val estado = "RS"  
3  
4     var resultado = when (estado) {  
5         "AC", "AM", "RR", "AP", "PA", "RO", "TO" -> "Norte"  
6         "SP", "MG", "ES", "RJ" -> "Sudeste"  
7         else -> "Outra"  
8     }  
9  
10     print(resultado)  
11 }
```

# Vetores

```
Main.kt x
1 fun main() {
2     var estados = arrayOf("SP", "MG", "RJ", "RS")
3     print(estados[0])
4 }
```

Run: MainKt x

↑ /home/jpescola/.jdk/openjdk-17.0.1/bin/java -  
↓ SP  
Process finished with exit code 0

```
Main.kt x
1 fun main() {
2     var lista = ArrayList<String>()
3     lista.add("RS")
4     lista.add("MG")
5     lista.add("SP")
6     lista.add("RJ")
7     println(lista[0])
8 }
```

Run: MainKt x

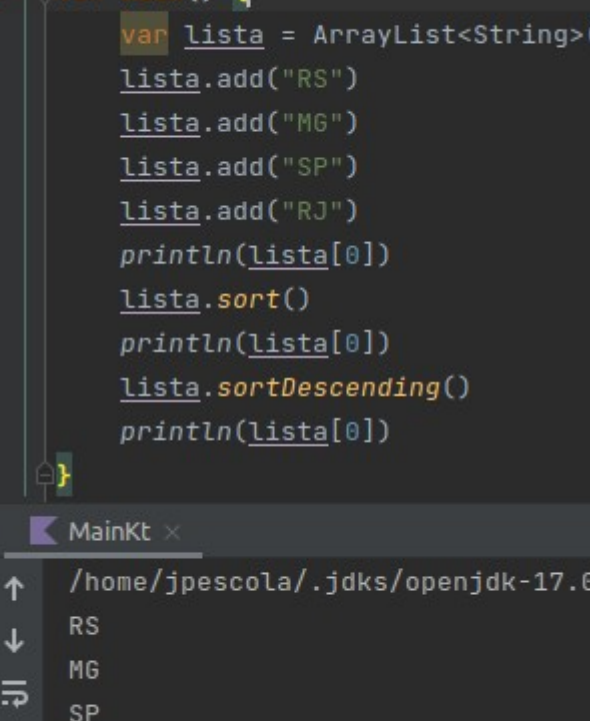
↑ /home/jpescola/.jdk/openjdk-17.0.1/bin  
↓ RS  
Process finished with exit code 0

```
Main.kt x
1 fun main() {
2     var valores = arrayOf(10, 20, 30, 50)
3     print(valores[0])
4 }
```

Run: MainKt x

↑ /home/jpescola/.jdk/openjdk-17.0.1/bin  
↓ 10  
Process finished with exit code 0

# Ordenação



The screenshot displays an IDE with a Kotlin file named `Main.kt`. The code defines a `main` function that creates an `ArrayList` named `lista` and adds the elements "RS", "MG", "SP", and "RJ". It then prints the first element, sorts the list, prints it again, sorts it in descending order, and prints it a third time. The Run console shows the output: "RS", "MG", "SP", and a message indicating the process finished with exit code 0.

```
1 fun main() {  
2     var lista = ArrayList<String>()  
3     lista.add("RS")  
4     lista.add("MG")  
5     lista.add("SP")  
6     lista.add("RJ")  
7     println(lista[0])  
8     lista.sort()  
9     println(lista[0])  
10    lista.sortDescending()  
11    println(lista[0])  
12 }
```

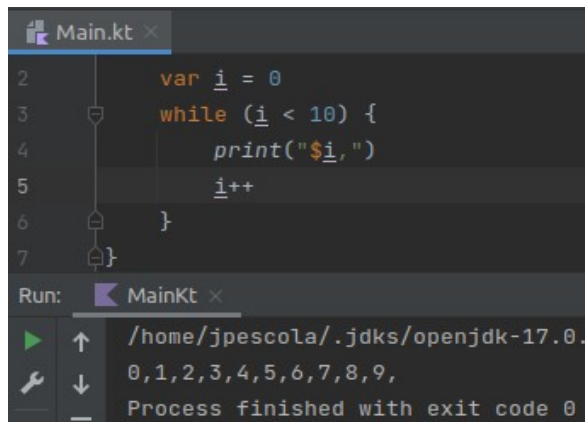
Run: MainKt ×

↑ /home/jpescola/.jdk/openjdk-17.0.1/bin/java  
↓ RS  
MG  
SP  
Process finished with exit code 0



# Laços de repetição

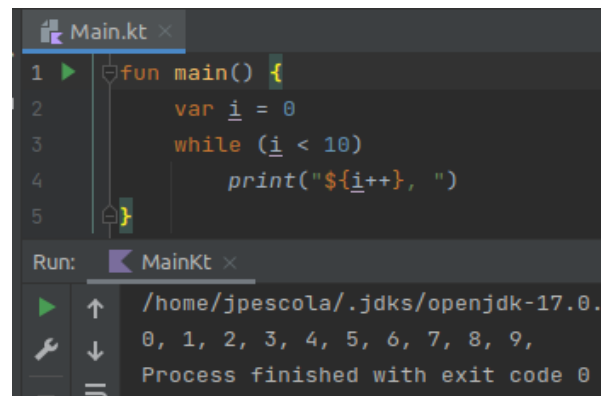
- FOR tradicional não existe em Kotlin!



```
1 Main.kt x
2   var i = 0
3   while (i < 10) {
4       print("$i,")
5       i++
6   }
7 }
```

Run: MainKt x

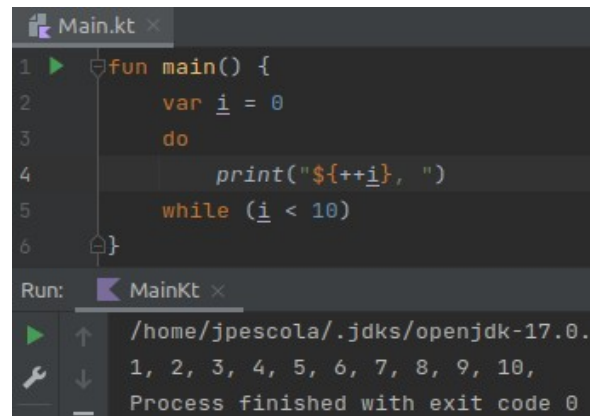
0,1,2,3,4,5,6,7,8,9,  
Process finished with exit code 0



```
1 Main.kt x
2   fun main() {
3       var i = 0
4       while (i < 10)
5           print("${i++}, ")
6   }
```

Run: MainKt x

0, 1, 2, 3, 4, 5, 6, 7, 8, 9,  
Process finished with exit code 0



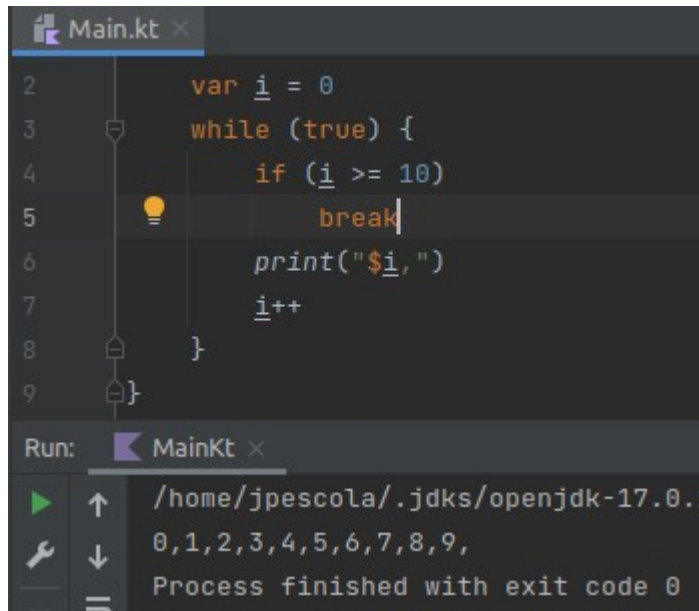
```
1 Main.kt x
2   fun main() {
3       var i = 0
4       do
5           print("${++i}, ")
6       while (i < 10)
7   }
```

Run: MainKt x

1, 2, 3, 4, 5, 6, 7, 8, 9, 10,  
Process finished with exit code 0

# Break e Continue

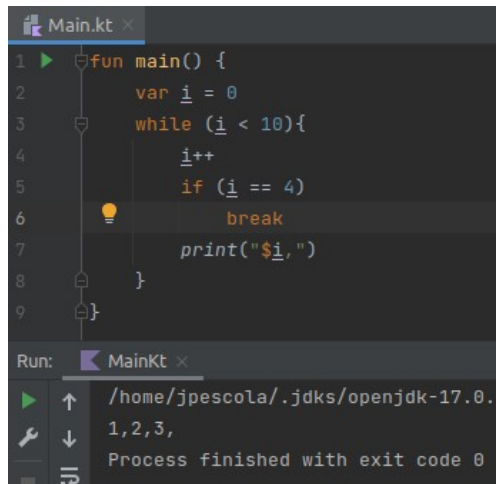
- 'break' encerra o laço de repetição;
- 'continue' pula a iteração atual e continua o laço.



```
1 Main.kt x
2     var i = 0
3     while (true) {
4         if (i >= 10)
5             break
6         print("$i,")
7         i++
8     }
9 }
```

Run: MainKt x

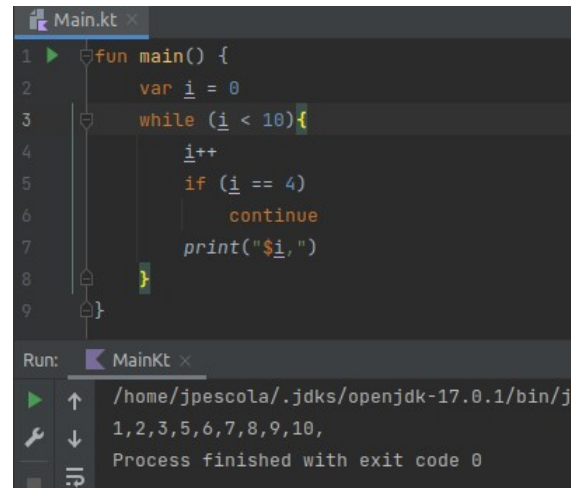
/home/jpescola/.jdk/openjdk-17.0.1/bin/java  
0,1,2,3,4,5,6,7,8,9,  
Process finished with exit code 0



```
1 Main.kt x
1 fun main() {
2     var i = 0
3     while (i < 10){
4         i++
5         if (i == 4)
6             break
7         print("$i,")
8     }
9 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/bin/java  
1,2,3,  
Process finished with exit code 0

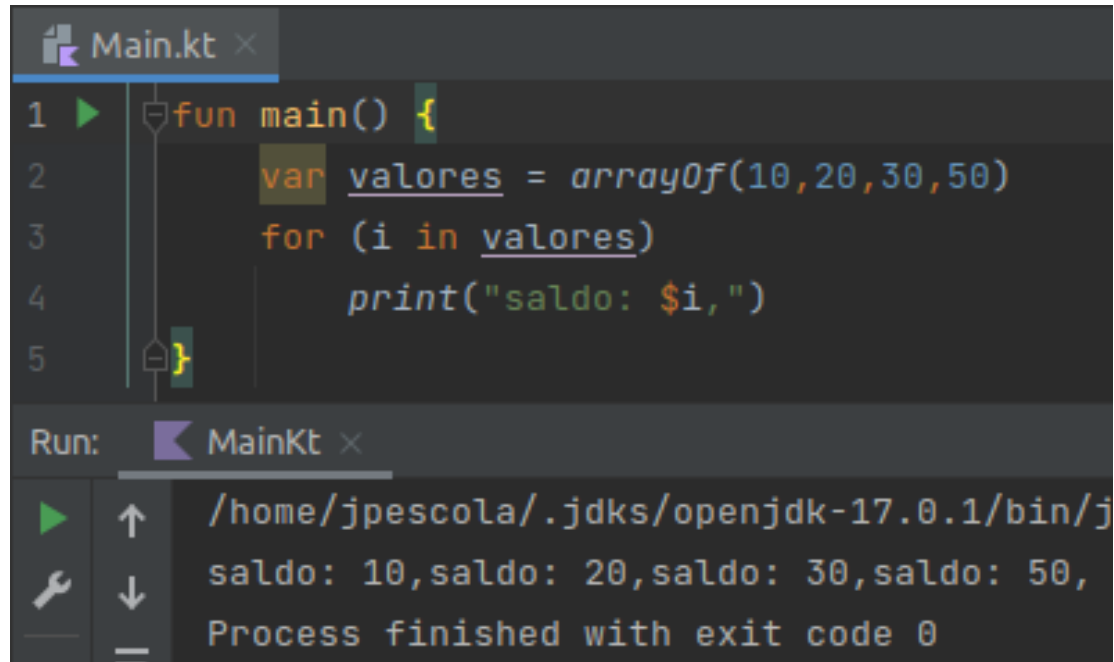


```
1 Main.kt x
1 fun main() {
2     var i = 0
3     while (i < 10){
4         i++
5         if (i == 4)
6             continue
7         print("$i,")
8     }
9 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/bin/java  
1,2,3,5,6,7,8,9,10,  
Process finished with exit code 0

# ‘For’ com ‘in’



The image shows a screenshot of an IDE with two panels. The top panel, titled 'Main.kt', contains the following Kotlin code:

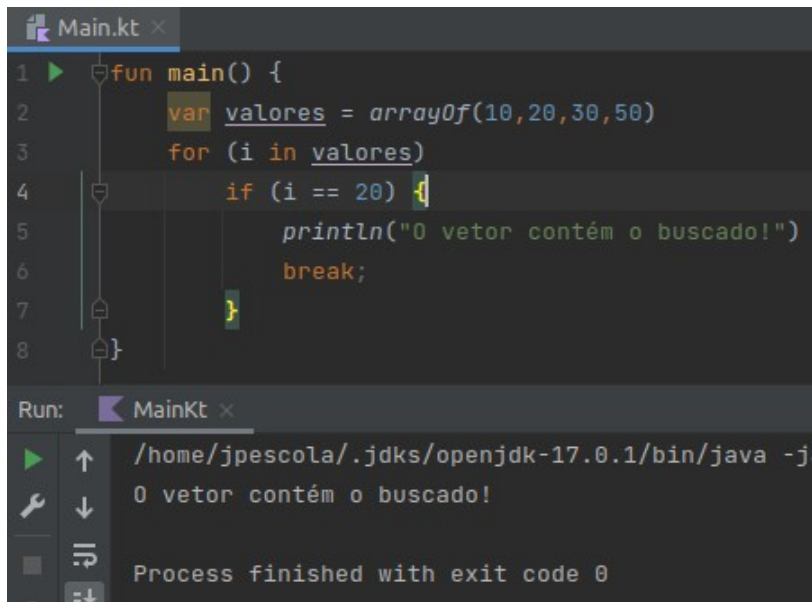
```
1 fun main() {  
2     var valores = arrayOf(10,20,30,50)  
3     for (i in valores)  
4         print("saldo: $i,")  
5 }
```

The bottom panel, titled 'Run: MainKt', shows the execution output:

```
/home/jpescola/.jdk/openjdk-17.0.1/bin/j  
saldo: 10,saldo: 20,saldo: 30,saldo: 50,  
Process finished with exit code 0
```

# Pesquisa em Vetores

- Crie um programa que procure um valor em um vetor, retornando o índice correspondente:

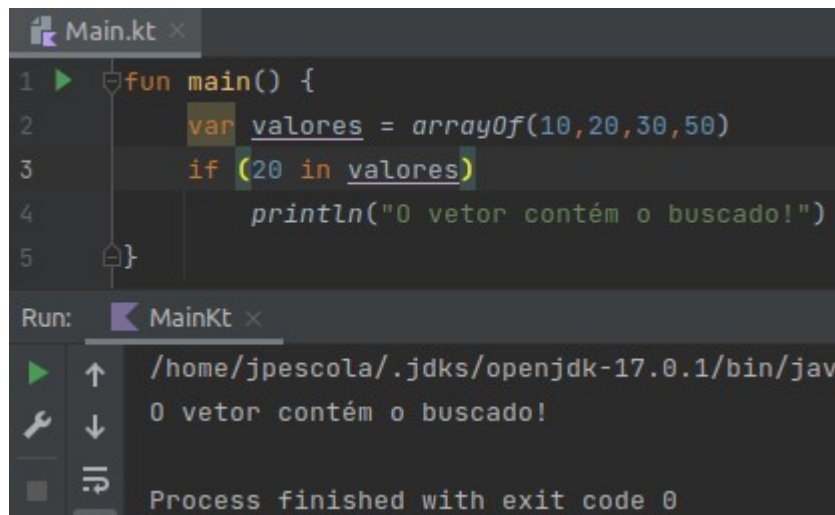


```
1 fun main() {  
2     var valores = arrayOf(10,20,30,50)  
3     for (i in valores)  
4         if (i == 20) {  
5             println("O vetor contém o buscado!")  
6             break;  
7         }  
8 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/bin/java -j  
0 vetor contém o buscado!

Process finished with exit code 0



```
1 fun main() {  
2     var valores = arrayOf(10,20,30,50)  
3     if (20 in valores)  
4         println("O vetor contém o buscado!")  
5 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/bin/jav  
0 vetor contém o buscado!

Process finished with exit code 0

# Ranges

```
Main.kt x
1 fun main() {
2     for (i in 1..10)
3         print("$i,")
4 }
```

Run: MainKt x

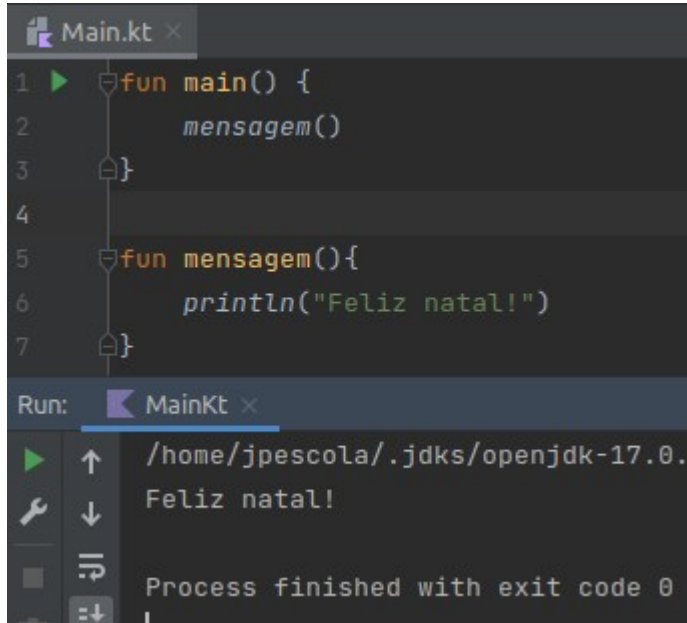
```
1 /home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:
2 1,2,3,4,5,6,7,8,9,10,
3 Process finished with exit code 0
```

```
Main.kt x
1 fun main() {
2     for (i in 'a'..'z')
3         print("$i,")
4 }
```

Run: MainKt x

```
1 /home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:
2 a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,
3 Process finished with exit code 0
```

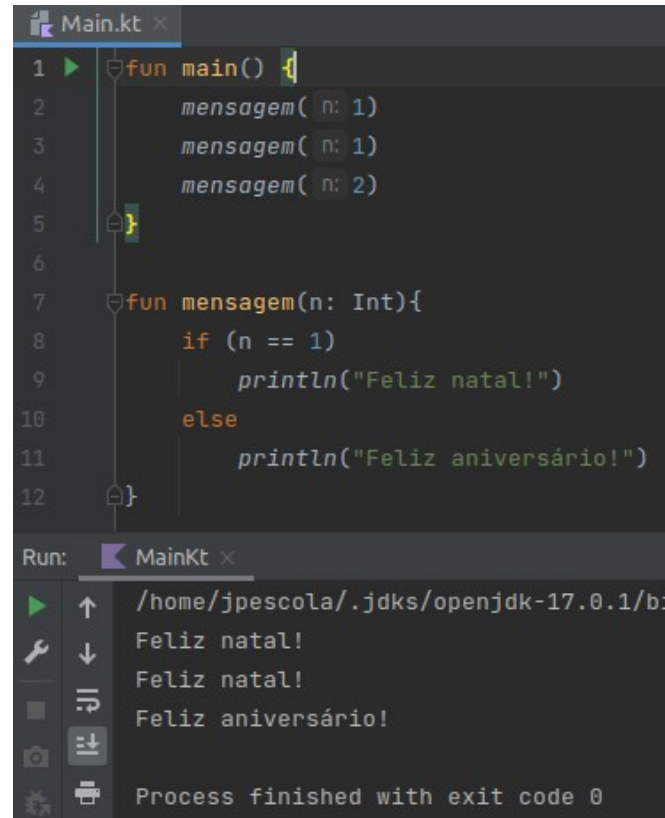
# Funções



```
1 fun main() {  
2     mensagem()  
3 }  
4  
5 fun mensagem(){  
6     println("Feliz natal!")  
7 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.  
Feliz natal!  
Process finished with exit code 0

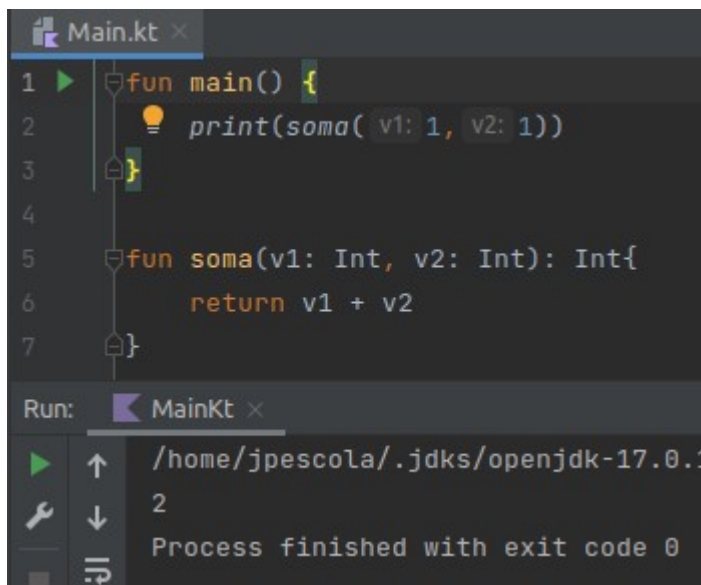


```
1 fun main() {  
2     mensagem(n: 1)  
3     mensagem(n: 1)  
4     mensagem(n: 2)  
5 }  
6  
7 fun mensagem(n: Int){  
8     if (n == 1)  
9         println("Feliz natal!")  
10    else  
11        println("Feliz aniversário!")  
12 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/b  
Feliz natal!  
Feliz natal!  
Feliz aniversário!  
Process finished with exit code 0

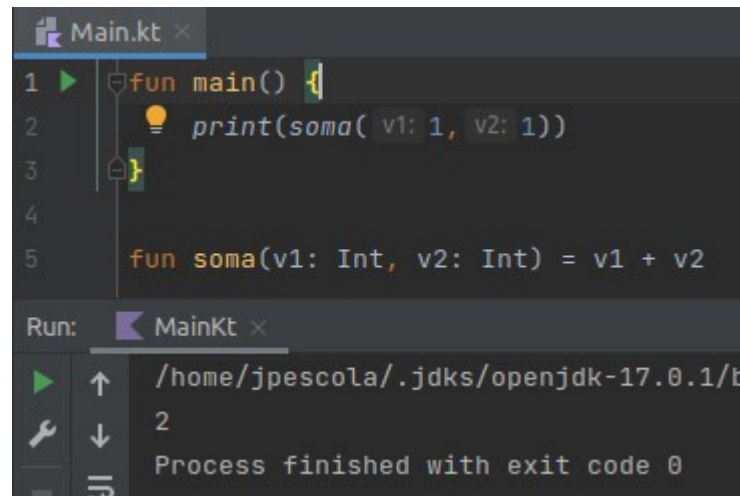
# Funções com retorno



```
1 fun main() {  
2     print(soma( v1: 1, v2: 1))  
3 }  
4  
5 fun soma(v1: Int, v2: Int): Int{  
6     return v1 + v2  
7 }
```

Run: MainKt x

Process finished with exit code 0



```
1 fun main() {  
2     print(soma( v1: 1, v2: 1))  
3 }  
4  
5 fun soma(v1: Int, v2: Int) = v1 + v2
```

Run: MainKt x

Process finished with exit code 0

# Classes

```
Main.kt x
1 fun main() {
2     val p1 = Pessoa()
3     val p2 = Pessoa()
4     val p3 = Pessoa()
5     println(p1.nome)
6     println(p2.nome)
7     println(p3.nome)
8 }
9
10 class Pessoa {
11     var nome = "Vazio"
12     var peso = 0f
13     var altura = 0f
14 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1  
Vazio  
Vazio  
Vazio  
Process finished with exit code 0

```
Main.kt x
1 fun main() {
2     val p1 = Pessoa()
3     val p2 = Pessoa()
4     val p3 = Pessoa()
5     p1.nome = "Maria"
6     println(p1.nome)
7     println(p2.nome)
8     println(p3.nome)
9 }
10
11 class Pessoa {
12     var nome = "Vazio"
13     var peso = 0f
14     var altura = 0f
15 }
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1  
Maria  
Vazio  
Vazio  
Process finished with exit code 0

```
Main.kt x
1 fun main() {
2     val p1 = Pessoa( nome: "Maria", peso: 60f, altura: 1.70f)
3     val p2 = Pessoa( nome: "Joaquim", peso: 70f, altura: 1.75f)
4     val p3 = Pessoa( nome: "Marcos", peso: 80f, altura: 1.80f)
5     println(p1.nome)
6     println(p2.nome)
7     println(p3.nome)
8 }
9
10 class Pessoa(var nome: String, var peso: Float, var altura: Float)
```

Run: MainKt x

/home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:/dados/ide  
Maria  
Joaquim  
Marcos  
Process finished with exit code 0



# Métodos

```
Main.kt x
1  fun main() {
2      val p1 = Pessoa( nome: "Maria",  peso: 60f,  altura: 1.70f)
3      val p2 = Pessoa( nome: "Joaquim", peso: 70f,  altura: 1.75f)
4      val p3 = Pessoa( nome: "Marcos",  peso: 80f,  altura: 1.80f)
5      println("${p1.nome} tem IMC = ${p1.imc()}")
6      println("${p2.nome} tem IMC = ${p2.imc()}")
7      println("${p3.nome} tem IMC = ${p3.imc()}")
8  }
9
10 class Pessoa(val nome: String, val peso: Float, val altura: Float){
11     fun imc() = peso / (altura * altura)
12 }

Run: MainKt x
  /home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:/dados/idea-I
  Maria tem IMC = 20.761246
  Joaquim tem IMC = 22.857143
  Marcos tem IMC = 24.69136
  Process finished with exit code 0
```

# Métodos Assessores

- Em Java utilizamos os métodos assessores para ‘termos acesso’ a um atributo: Ex: `getNome` para o campo `nome`;
- Em Kotlin esses métodos não são necessários, ou seja, eles são criados automaticamente quando definimos os argumentos do construtor como constantes, ou seja, ‘`val`’;
- Entretanto o padrão em Kotlin é utilizar o método ‘`nome`’ e não ‘`getNome`’ como acontece no Java;

# Assessores Customizados

- Vimos que o nosso método 'imc' retorna o resultado do cálculo do índice de massa corpórea do objeto Pessoa;
- Todo método (ou função) tem os parênteses, seja vazio ou com parâmetros;
- No caso de assessores customizados, devemos emitir os parênteses, já que agora temos um campo e não mais um método.

# Assessores Customizados

```
Main.kt x Main.decompiled.java x
1 fun main() {
2     val p1 = Pessoa( nome: "Maria", peso: 60f, altura: 1.70f)
3     val p2 = Colaborador( salario: 2000f, nome: "Joaquim", peso: 70f, altura: 1.75f)
4     val p3 = Cliente( renda: 1500f, nome: "Marcos", peso: 80f, altura: 1.80f)
5     println("${p1.nome} tem IMC = ${p1.imc}")
6     println("${p2.nome} tem IMC = ${p2.imc}")
7     println("${p3.nome} tem IMC = ${p3.imc}")
8 }
9
10 open class Pessoa(val nome: String, val peso: Float, val altura: Float){
11     // fun imc() = peso / (altura * altura)
12     val imc get() = peso / (altura * altura)
13 }
```

# Herança

```
Main.kt x
1 fun main() {
2     val p1 = Pessoa( nome: "Maria", peso: 60f, altura: 1.70f)
3     val p2 = Colaborador( salario: 2000f, nome: "Joaquim", peso: 70f, altura: 1.75f)
4     val p3 = Cliente( renda: 1500f, nome: "Marcos", peso: 80f, altura: 1.80f)
5     println("${p1.nome} tem IMC = ${p1.imc()}")
6     println("${p2.nome} tem IMC = ${p2.imc()} e ${if (p2.ativo) "está ativo" else "foi despedido"}")
7     println("${p3.nome} tem IMC = ${p3.imc()} e renda = ${p3.renda}")
8 }
9
10 open class Pessoa(val nome: String, val peso: Float, val altura: Float){
11     fun imc() = peso / (altura * altura)
12 }
13
14 class Cliente(var renda: Float, nome: String, peso: Float, altura: Float): Pessoa(nome, peso, altura) {
15     fun aprovacao() = renda > 1000
16 }
17
18 class Colaborador(var salario: Float, nome: String, peso: Float, altura: Float):Pessoa(nome, peso, altura){
19     val ativo = true
20     fun bonus() = salario > 0
21 }

Run: MainKt x
/home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:/dados/idea-IC-213.5744.223/lib/idea_rt.jar=32845:/dados/idea-IC-213.5744.223/bin
Maria tem IMC = 20.761246
Joaquim tem IMC = 22.857143 e está ativo
Marcos tem IMC = 24.69136 e renda = 1500.0
Process finished with exit code 0
```

# Imprimindo objetos

```
Main.kt x
1 fun main() {
2     val p1 = Pessoa( nome: "Maria", peso: 60f, altura: 1.70f)
3     val p2 = Pessoa( nome: "Joaquim", peso: 70f, altura: 1.75f)
4     val p3 = Pessoa( nome: "Marcos", peso: 80f, altura: 1.80f)
5     println(p1)
6     println(p2)
7     println(p3.toString())
8 }
9
10 class Pessoa(var nome: String, var peso: Float, var altura: Float)
```

Run: MainKt x

```
/home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:/dados/idea
Pessoa@27716f4
Pessoa@8efb846
Pessoa@2a84aee7
Process finished with exit code 0
```

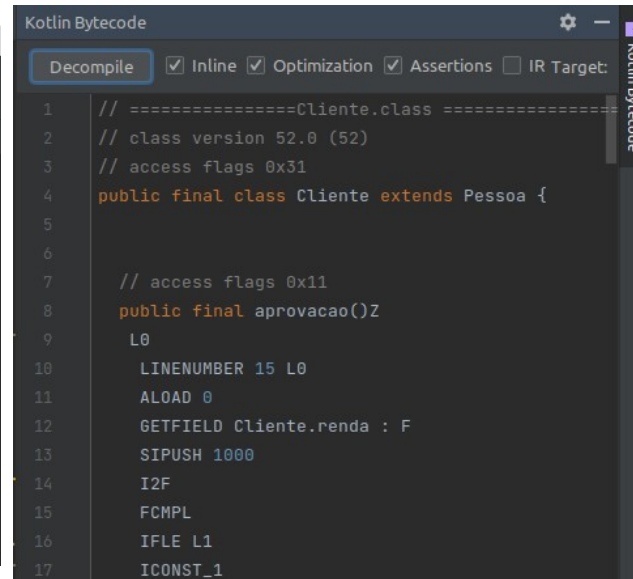
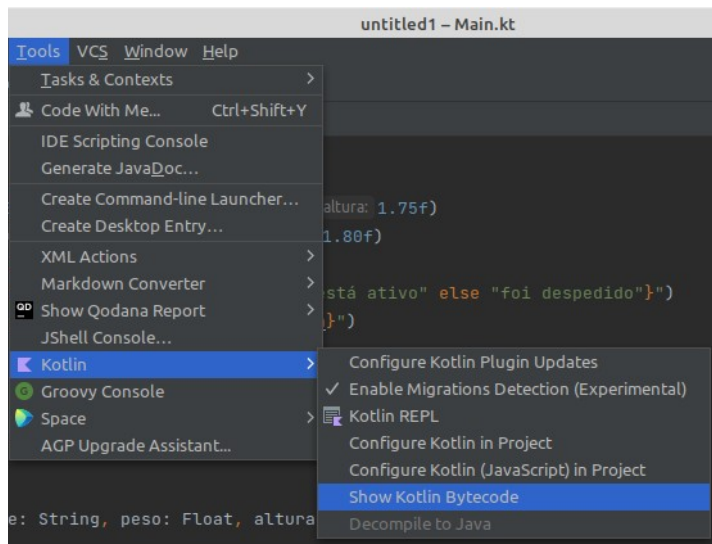
```
Main.kt x
1 fun main() {
2     val p1 = Pessoa( nome: "Maria", peso: 60f, altura: 1.70f)
3     val p2 = Pessoa( nome: "Joaquim", peso: 70f, altura: 1.75f)
4     val p3 = Pessoa( nome: "Marcos", peso: 80f, altura: 1.80f)
5     println(p1)
6     println(p2)
7     println(p3.toString())
8 }
9
10 class Pessoa(var nome: String, var peso: Float, var altura: Float){
11     override fun toString(): String{
12         return "$nome, $peso, $altura"
13     }
14 }
```

Run: MainKt x

```
/home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:/dados/idea
Maria, 60.0, 1.7
Joaquim, 70.0, 1.75
Marcos, 80.0, 1.8
Process finished with exit code 0
```

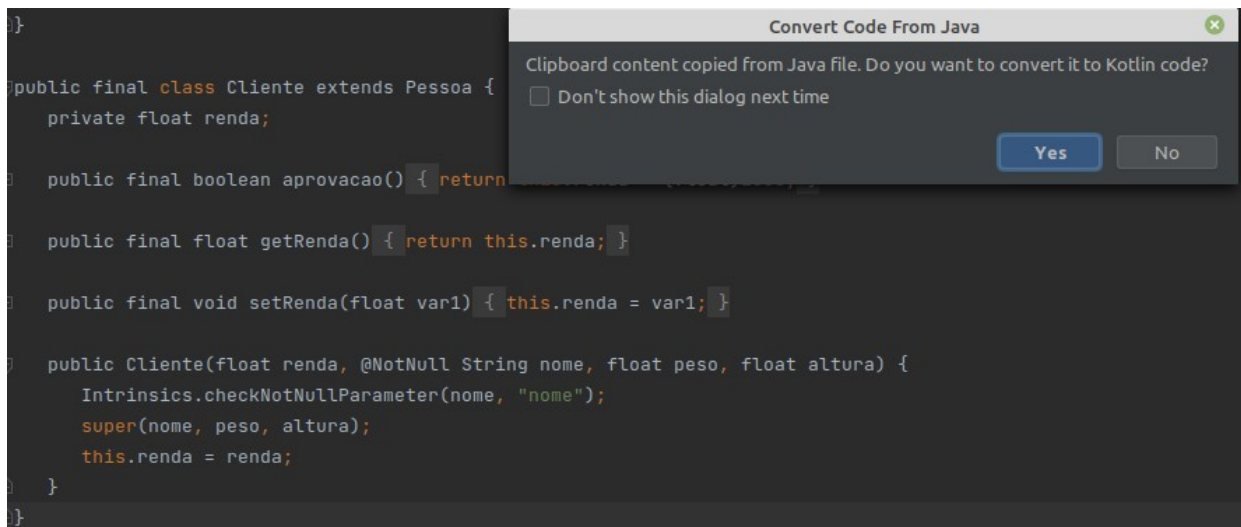
# Converter Kotlin para Java

- Como vimos, os Bytecodes são a forma que a JVM entende os códigos que criamos;
- Java e Kotlin geram Bytecodes;
- Vamos visualizar os Bytecodes, selecionando uma classe Kotlin e clicando no menu Tools / Kotlin / Show Kotlin Bytecode;
- Em seguida clique em 'Decompile' para visualizar o código correspondente em Java.



# Converter Java para Kotlin

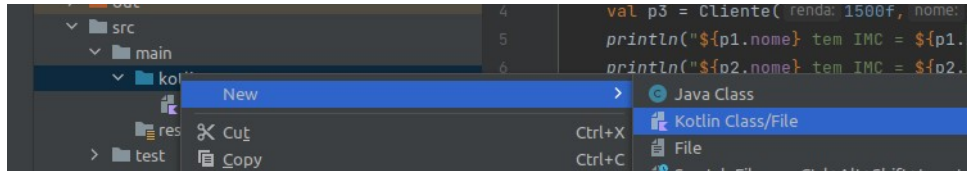
- Copie um trecho de código Java e cole em um arquivo .kt;
- O Idea vai mostrar uma mensagem perguntando se deseja converter o código para Kotlin:



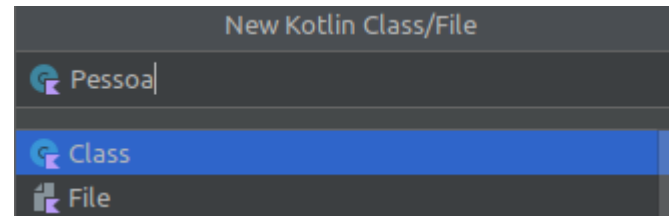


# Classes em arquivos separados

- Uma boa prática de desenvolvimento de software é separar cada classe em seu arquivo correspondente;
- Vamos criar um arquivo para cada classe criada anteriormente;
- Para isso, clique com o botão direito na pasta kotlin e escolha new / Kotlin Class/File:

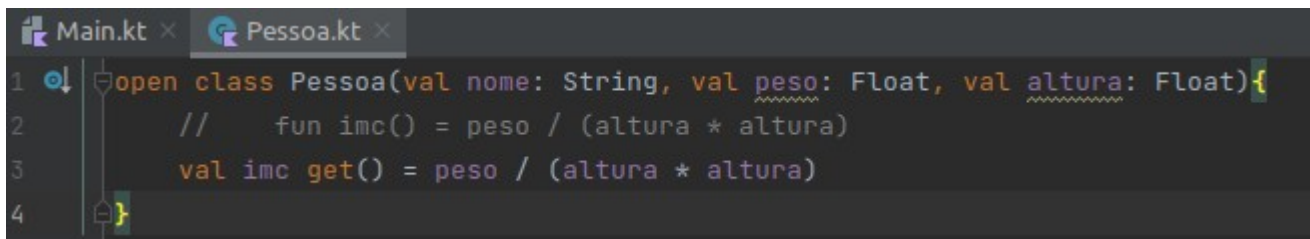


- Em seguida digite o nome da classe e pressione Enter:



# Classe Pessoa

- Recorte o código da classe Pessoa do arquivo Main.kt e cole no arquivo Pessoa.kt:

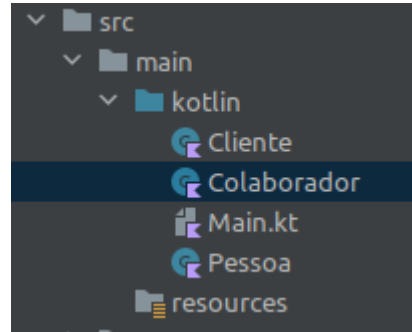


```
1 open class Pessoa(val nome: String, val peso: Float, val altura: Float){
2     // fun imc() = peso / (altura * altura)
3     val imc get() = peso / (altura * altura)
4 }
```

- Execute o projeto e veja que tudo está funcionando normalmente;
- Faça o mesmo para as classes Cliente e Colaborador.

# Classes em arquivos

- Veja como ficou a estrutura de arquivos:

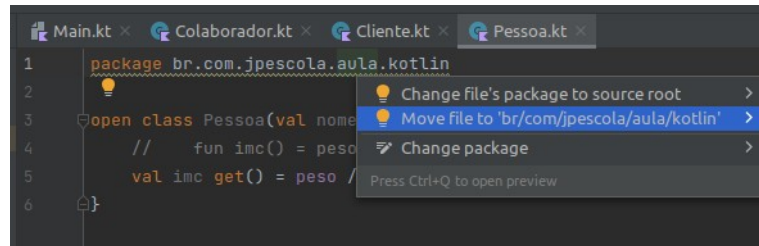


# Pacotes

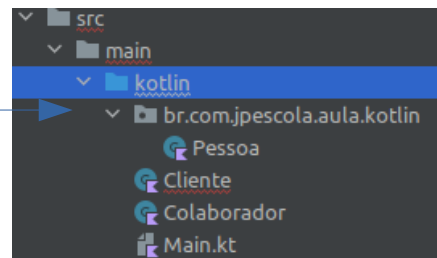
- Permitem organizar as classes e arquivos de um projeto;
- Graças aos pacotes, cada fabricante pode ter suas classes com os mesmos nomes dos concorrentes: Ex: Data, Util, BD etc;
- Assim como em Java, em Kotlin utilizamos as palavras reservadas 'package' para definir o pacote atual e 'import' para utilizar classes de pacotes externos.

# Colocando as classes em pacotes

- Digite a palavra-chave 'package' no início do arquivo e coloque o nome do pacote;
- O Idea vai sugerir a criação dos diretórios correspondentes;
- Escolha 'Move file to...'

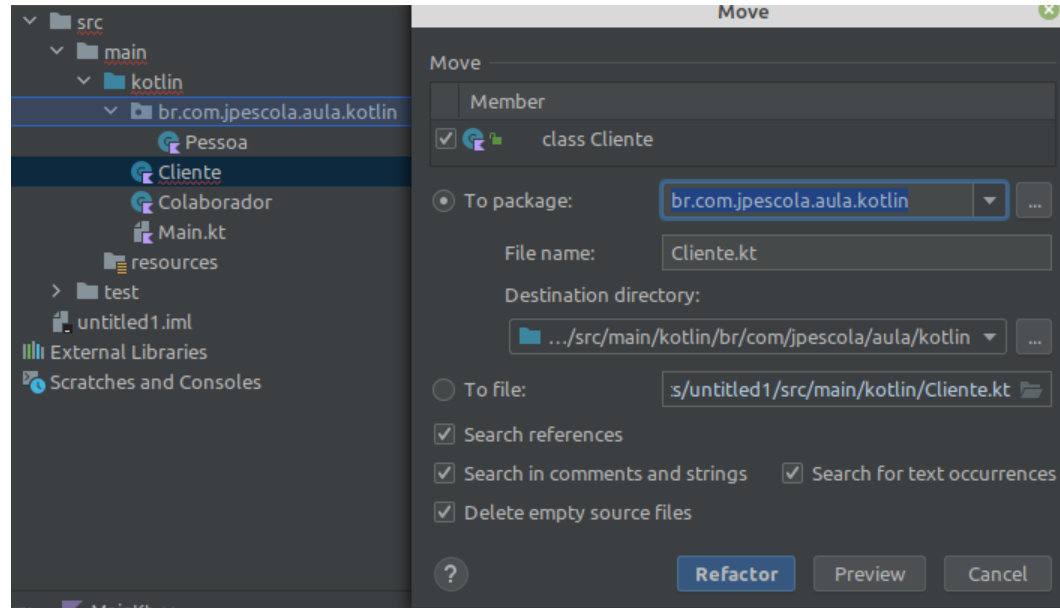


Veja o resultado



# Movendo as outras classes

- Para mover as demais classes, arraste os arquivos para dentro do pacote;
- Na janela de confirmação escolha 'Refactor';
- A IDE vai automaticamente adicionar a cláusula 'package' nos arquivos movidos.

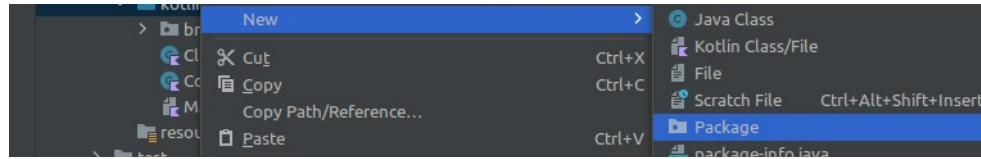


# Pacote Util

- Uma boa prática em desenvolvimento de software é criar um pacote com os métodos mais utilizados em nossos projetos;
- Permitindo reutilização de código dentro do projeto e também em outros projetos que desenvolvermos;
- Vamos criar o pacote Util com alguns métodos que serão aproveitados em nosso arquivo principal Main.kt.

# Criando o pacote Util

- Clique com o botão direito na pasta kotlin e escolha new / Package:





# Classe Calculadora

- No pacote Util, vamos criar a classe Calculadora;
- e executar um de seus métodos.

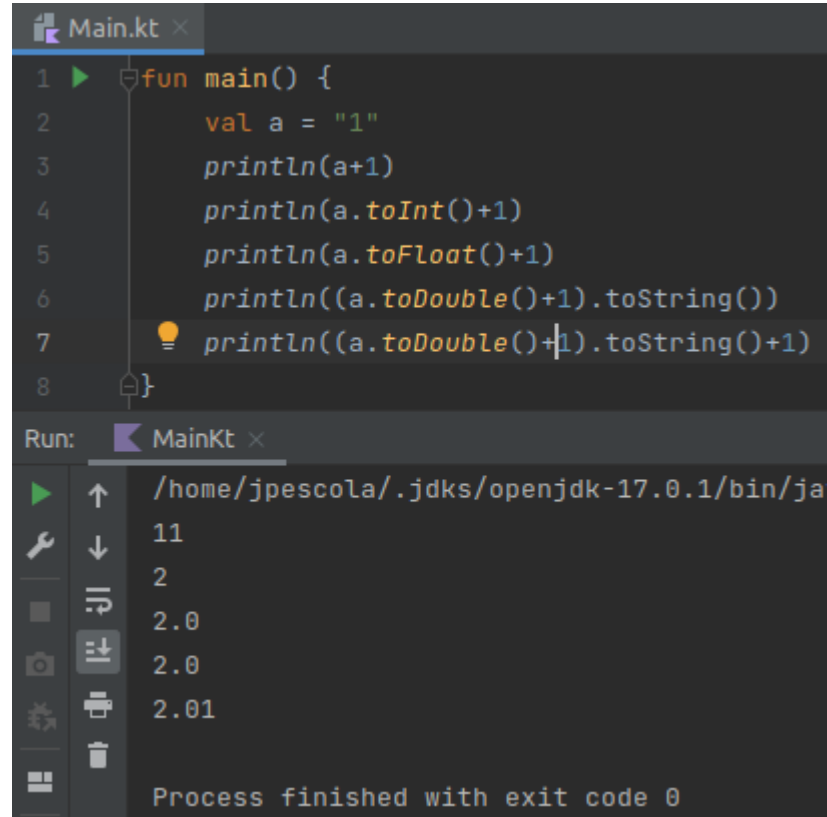
```
1 package br.com.jpescola.util
2
3 class Calculadora {
4     fun soma(v1: Int, v2: Int) = v1 + v2
5     fun subtracao(v1: Int, v2: Int) = v1 - v2
6     fun multiplicacao(v1: Int, v2: Int) = v1 * v2
7     fun divisao(v1: Int, v2: Int) = v1 / v2
8 }
```

```
1 import br.com.jpescola.util.Calculadora
2
3 fun main() {
4     println(Calculadora().soma(1, 2))
5 }
6
7
8
9
10
11
```

# Sobrecarga

```
1 package br.com.jpescola.util
2
3 class Calculadora {
4     fun soma(v1: Int, v2: Int) = v1 + v2
5     fun subtracao(v1: Int, v2: Int) = v1 - v2
6     fun multiplicacao(v1: Int, v2: Int) = v1 * v2
7     // fun divisao(v1: Int, v2: Int) = v1 / v2
8     fun divisao(v1: Int, v2: Int) = v1 / v2.toFloat()
9     fun divisao(v1: Float, v2: Float) = v1 / v2
10 }
```

# Cast



The screenshot shows an IDE window with a file named `Main.kt`. The code defines a `main` function that performs several type conversions on the string `"1"`. The code is as follows:

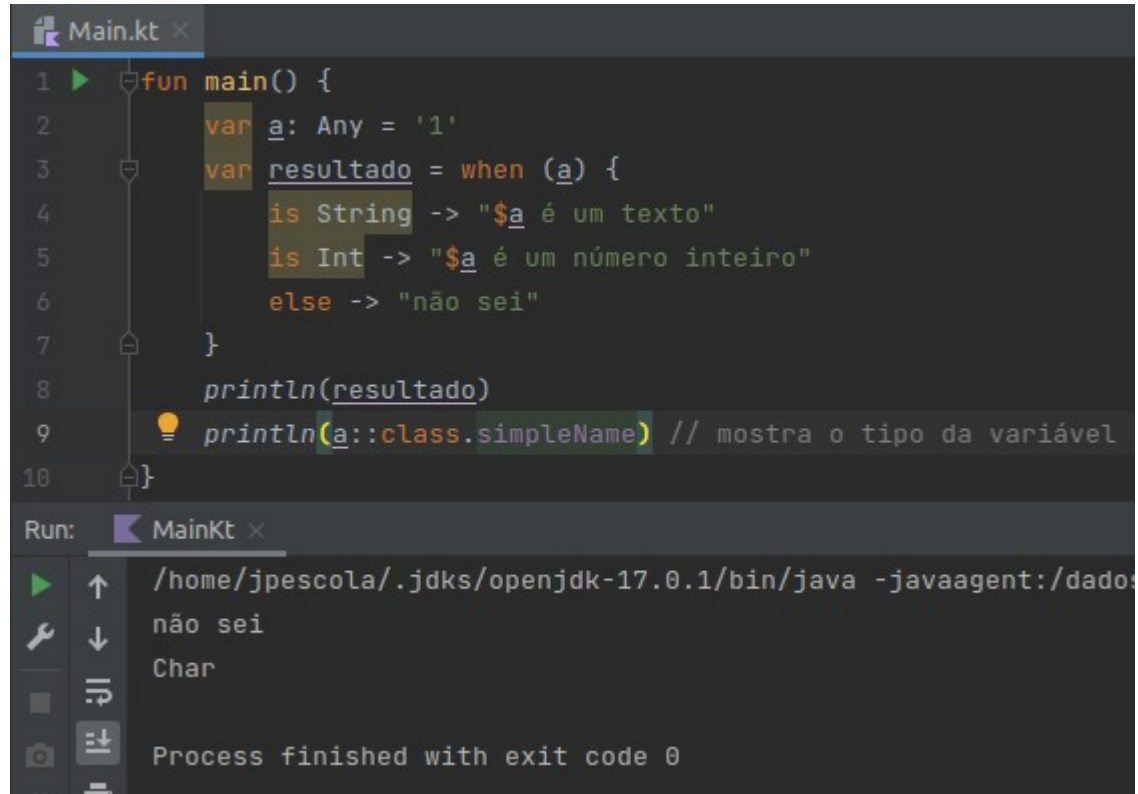
```
1 fun main() {  
2     val a = "1"  
3     println(a+1)  
4     println(a.toInt()+1)  
5     println(a.toFloat()+1)  
6     println((a.toDouble()+1).toString())  
7     println((a.toDouble()+1).toString()+1)  
8 }
```

Below the code editor, the `Run` tab is active, showing the execution command and its output:

```
Run: MainKt x  
/home/jpescola/.jdk/openjdk-17.0.1/bin/java  
11  
2  
2.0  
2.0  
2.01  
Process finished with exit code 0
```

The output demonstrates the results of the different casts: `a+1` results in `11`, `a.toInt()+1` results in `2`, `a.toFloat()+1` results in `2.0`, `(a.toDouble()+1).toString()` results in `2.0`, and `(a.toDouble()+1).toString()+1` results in `2.01`.

# Smart cast



```
1 fun main() {  
2     var a: Any = '1'  
3     var resultado = when (a) {  
4         is String -> "$a é um texto"  
5         is Int -> "$a é um número inteiro"  
6         else -> "não sei"  
7     }  
8     println(resultado)  
9     println(a::class.simpleName) // mostra o tipo da variável  
10 }
```

Run: MainKt x

↑ /home/jpescola/.jdk/openjdk-17.0.1/bin/java -javaagent:/dados  
↓ não sei  
Char  
Process finished with exit code 0

# Exceções

- Em Kotlin podemos utilizar as palavras reservadas try, catch, throws e finally como aprendemos em Java.

```
Main.kt x
1  fun main() {
2      var r: Any = "s"
3      var t = 0f
4      do{
5          print("Digite um número inteiro ou 's' para sair: ")
6          try {
7              r = readLine()!!
8              if (r == "s")
9                  break
10             t += r.toInt()
11         }
12         catch (e: NumberFormatException) {
13             println("    Número inválido, tente novamente!")
14         }
15         catch (e: Exception) {
16             println("    !!!Houve uma exceção!!!")
17         }
18         finally {
19             println("    total parcial: $t")
20         }
21     }while (r != "s")
22 }
```

```
Run: MainKt x
/home/jpescola/.jids/openjdk-17.0.1/bin/java -javaagent:/dad
Digite um número inteiro ou 's' para sair: 1
    total parcial: 1.0
Digite um número inteiro ou 's' para sair: 2
    total parcial: 3.0
Digite um número inteiro ou 's' para sair: s
    total parcial: 3.0
Process finished with exit code 0
```

# O que aprendemos?

- Histórico da linguagem;
- Instalação da IDE;
- Sintaxe básica;
- Iterações (laços de repetição);
- Classes / métodos / herança / pacotes / sobrecarga;
- Conversões de tipos;
- Exceções

# Na próxima aula...

- Projetos Android com Kotlin.

# Atividade

Vamos treinar a linguagem Kotlin executando os exemplos dos slides apresentados na aula 4:

- Crie um projeto no Idea chamado “Aula4”;
  - Para cada ‘main’, crie uma função com o número do slide, ex: fun slide1(), fun slide2()...
  - Dentro de cada função, adicione o código apresentado no respectivo slide;
  - Adicione um comentário na primeira linha de cada função com uma explicação resumida do código.



# Referências

- [kotlinlang.org](https://kotlinlang.org)
- [w3schools.com/kotlin](https://w3schools.com/kotlin)