Kotlin

Rafael Vieira Coelho
rafaelvc2@gmail.com

https://kotlinlang.org

1) Funções de Alta Ordem

Ex1HighOrderFunction.kt

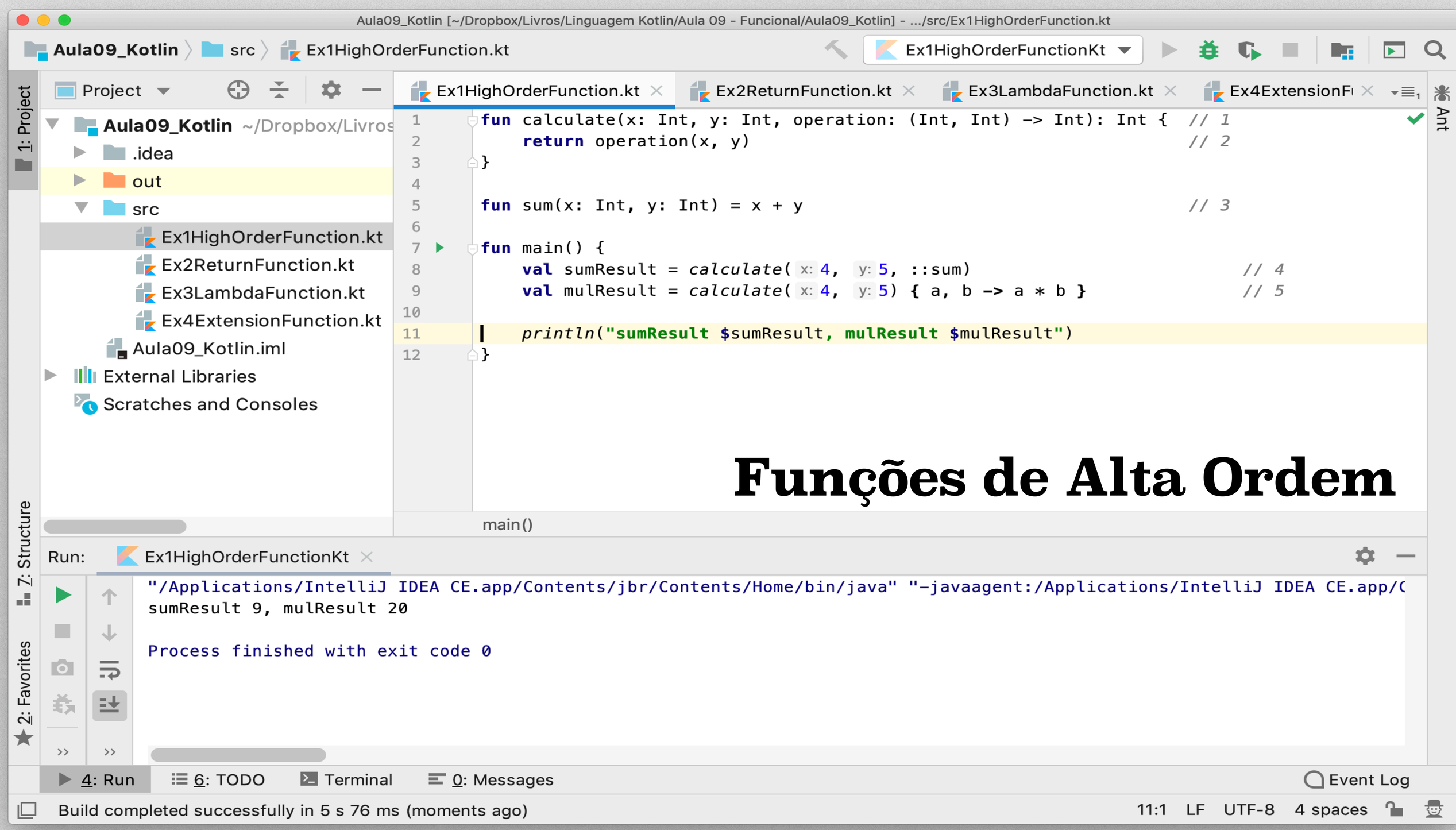
```kotlin
fun calculate(x: Int, y: Int, operation: (Int, Int) -> Int): Int {  // 1
    return operation(x, y)                                          // 2
}

fun sum(x: Int, y: Int) = x + y                                     // 3

fun main() {
    val sumResult = calculate( x: 4,  y: 5, ::sum)                  // 4
    val mulResult = calculate( x: 4,  y: 5) { a, b -> a * b }       // 5

    println("sumResult $sumResult, mulResult $mulResult")
}
```

1. Declara a função de alta ordem. Ela recebe dois inteiros e uma outra função operation() como parâmetros. Os parâmetros de operation() também são definidos.
2. A função calculate() retorna o resultado de operation()
3. Declara a função sum() que tem a assinatura igual a operation().
4. Invoca a função de alta ordem, passando os dois inteiros x e y como parâmetro. O símbolo :: é a notação que referencia uma função pelo nome em Kotlin.
5. Invoca a função de alta ordem passando diretamente a função como um lambda.


Aula09_Kotlin [~/Dropbox/Livros/Linguagem Kotlin/Aula 09 - Funcional/Aula09_Kotlin] - .../src/Ex1HighOrderFunction.kt
Aula09_Kotlin
src
Ex1HighOrderFunction.kt
Ex1HighOrderFunctionKt
Project
Aula09_Kotlin ~/Dropbox/Livros
.idea
out
src
Ex1HighOrderFunction.kt
Ex2ReturnFunction.kt
Ex3LambdaFunction.kt
Ex4ExtensionFunction.kt
Aula09_Kotlin.iml
External Libraries
Scratches and Consoles
Ex1HighOrderFunction.kt
Ex2ReturnFunction.kt
Ex3LambdaFunction.kt
Ex4ExtensionF
fun calculate(x: Int, y: Int, operation: (Int, Int) -> Int): Int { // 1
return operation(x, y) // 2
}
fun sum(x: Int, y: Int) = x + y // 3
fun main() {
val sumResult = calculate(x: 4, y: 5, ::sum) // 4
val mulResult = calculate(x: 4, y: 5) { a, b -> a * b } // 5
println("sumResult $sumResult, mulResult $mulResult")
}
Funções de Alta Ordem
main()
Run:
Ex1HighOrderFunctionKt
"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents/Home/bin/java" "-javaagent:/Applications/IntelliJ IDEA CE.app/C
sumResult 9, mulResult 20
Process finished with exit code 0
1: Project
7: Structure
2: Favorites
Ant
4: Run
6: TODO
Terminal
0: Messages
Event Log
Build completed successfully in 5 s 76 ms (moments ago)
11:1
LF
UTF-8
4 spaces

2) Retorno de Função

1. Declara a função de alta ordem que retorna uma função
2. Declara a função que é retornada no passo 1
3. Invoca operation2() para obter o resultado retornado e armazenar em uma variável. Aqui func() se transforma em square().
4. Invoca func() e square() é executada.

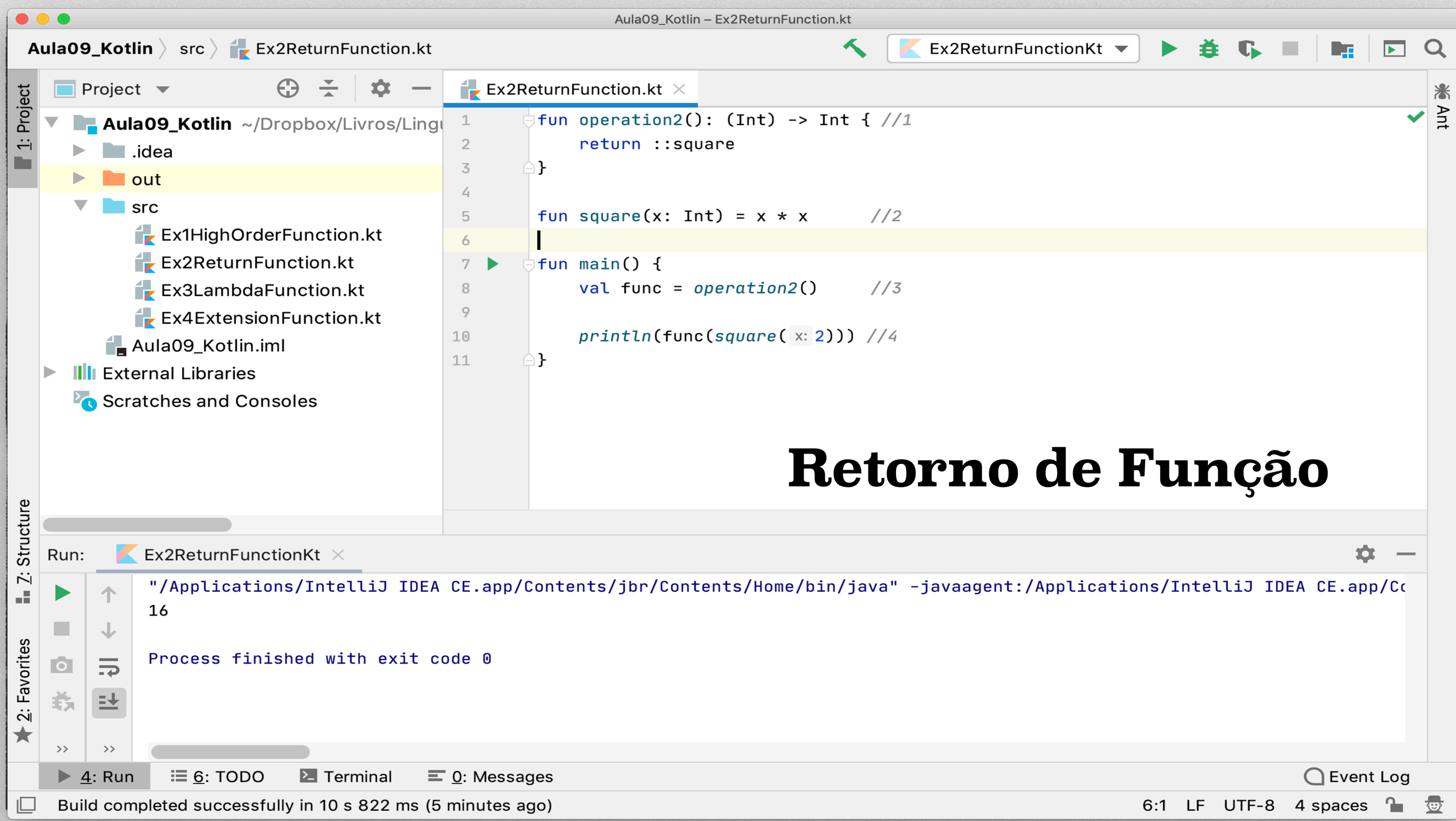
Ex2ReturnFunction.kt

```kotlin
fun operation2(): (Int) -> Int { //1
    return ::square
}

fun square(x: Int) = x * x       //2

fun main() {
    val func = operation2()      //3

    println(func(square( x: 2))) //4
}
```


Aula09_Kotlin – Ex2ReturnFunction.kt
Aula09_Kotlin 〉 src 〉 Ex2ReturnFunction.kt
Ex2ReturnFunctionKt
Project
Aula09_Kotlin ~/Dropbox/Livros/Lingu
.idea
out
src
Ex1HighOrderFunction.kt
Ex2ReturnFunction.kt
Ex3LambdaFunction.kt
Ex4ExtensionFunction.kt
Aula09_Kotlin.iml
External Libraries
Scratches and Consoles
Ex2ReturnFunction.kt
1 fun operation2(): (Int) -> Int { //1
2 return ::square
3 }
4
5 fun square(x: Int) = x * x //2
6
7 fun main() {
8 val func = operation2() //3
9
10 println(func(square(x: 2))) //4
11 }
Retorno de Função
Run: Ex2ReturnFunctionKt
"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents/Home/bin/java" -javaagent:/Applications/IntelliJ IDEA CE.app/Co
16
Process finished with exit code 0
4: Run
6: TODO
Terminal
0: Messages
Event Log
Build completed successfully in 10 s 822 ms (5 minutes ago)
6:1 LF UTF-8 4 spaces
1: Project
7: Structure
2: Favorites
Ant

Funções Lambda

Ex3LambdaFunction.kt

```kotlin
fun main() {
// All examples create a function object that performs upper-casing.
// So it's a function from String to String
    val upperCase1: (String) -> String = { str: String -> str.toUpperCase() } // 1
    val upperCase2: (String) -> String = { str -> str.toUpperCase() }         // 2
    val upperCase3 = { str: String -> str.toUpperCase() }                     // 3
// val upperCase4 = { str -> str.toUpperCase() }                              // 4
    val upperCase5: (String) -> String = { it.toUpperCase() }                 // 5
    val upperCase6: (String) -> String = String::toUpperCase                  // 6

    println(upperCase2("hello"))
    println(upperCase1("hello"))
    println(upperCase3("hello"))
    println(upperCase5("hello"))
    println(upperCase6("hello"))
}
```

1. O lambda é definido entre as chaves { } e assinalado o tipo String.

Funções Lambda

Ex3LambdaFunction.kt

```kotlin
fun main() {
// All examples create a function object that performs upper-casing.
// So it's a function from String to String
    val upperCase1: (String) -> String = { str: String -> str.toUpperCase() } // 1
    val upperCase2: (String) -> String = { str -> str.toUpperCase() }         // 2
    val upperCase3 = { str: String -> str.toUpperCase() }                     // 3
// val upperCase4 = { str -> str.toUpperCase() }                              // 4
    val upperCase5: (String) -> String = { it.toUpperCase() }                 // 5
    val upperCase6: (String) -> String = String::toUpperCase                  // 6

    println(upperCase2("hello"))
    println(upperCase1("hello"))
    println(upperCase3("hello"))
    println(upperCase5("hello"))
    println(upperCase6("hello"))
}
```

2. Inferência de tipo dentro do lambda a partir do tipo do parâmetro passado.

Funções Lambda

Ex3LambdaFunction.kt

```kotlin
fun main() {
// All examples create a function object that performs upper-casing.
// So it's a function from String to String
    val upperCase1: (String) -> String = { str: String -> str.toUpperCase() } // 1
    val upperCase2: (String) -> String = { str -> str.toUpperCase() }         // 2
    val upperCase3 = { str: String -> str.toUpperCase() }                     // 3
// val upperCase4 = { str -> str.toUpperCase() }                              // 4
    val upperCase5: (String) -> String = { it.toUpperCase() }                 // 5
    val upperCase6: (String) -> String = String::toUpperCase                  // 6

    println(upperCase2("hello"))
    println(upperCase1("hello"))
    println(upperCase3("hello"))
    println(upperCase5("hello"))
    println(upperCase6("hello"))
}
```

3. Inferência de tipo fora do lambda. O tipo da variável é assumido a partir do que é retornado pela função lambda.

Funções Lambda

Ex3LambdaFunction.kt

```kotlin
fun main() {
// All examples create a function object that performs upper-casing.
// So it's a function from String to String
    val upperCase1: (String) -> String = { str: String -> str.toUpperCase() } // 1
    val upperCase2: (String) -> String = { str -> str.toUpperCase() }         // 2
    val upperCase3 = { str: String -> str.toUpperCase() }                     // 3
// val upperCase4 = { str -> str.toUpperCase() }                              // 4
    val upperCase5: (String) -> String = { it.toUpperCase() }                 // 5
    val upperCase6: (String) -> String = String::toUpperCase                  // 6

    println(upperCase2("hello"))
    println(upperCase1("hello"))
    println(upperCase3("hello"))
    println(upperCase5("hello"))
    println(upperCase6("hello"))
}
```

4. Não é possível fazer os passos 2 e 3 ao mesmo tempo. O compilador não tem como saber qual tipo será inferido.

Funções Lambda

Ex3LambdaFunction.kt

```kotlin
fun main() {
// All examples create a function object that performs upper-casing.
// So it's a function from String to String
    val upperCase1: (String) -> String = { str: String -> str.toUpperCase() } // 1
    val upperCase2: (String) -> String = { str -> str.toUpperCase() }         // 2
    val upperCase3 = { str: String -> str.toUpperCase() }                     // 3
// val upperCase4 = { str -> str.toUpperCase() }                              // 4
    val upperCase5: (String) -> String = { it.toUpperCase() }                 // 5
    val upperCase6: (String) -> String = String::toUpperCase                  // 6

    println(upperCase2("hello"))
    println(upperCase1("hello"))
    println(upperCase3("hello"))
    println(upperCase5("hello"))
    println(upperCase6("hello"))
}
```

5. Para lambdas com um único parâmetro, não é necessário um nome. Pode ser usado a variável implícita it que contém o seu iterador.

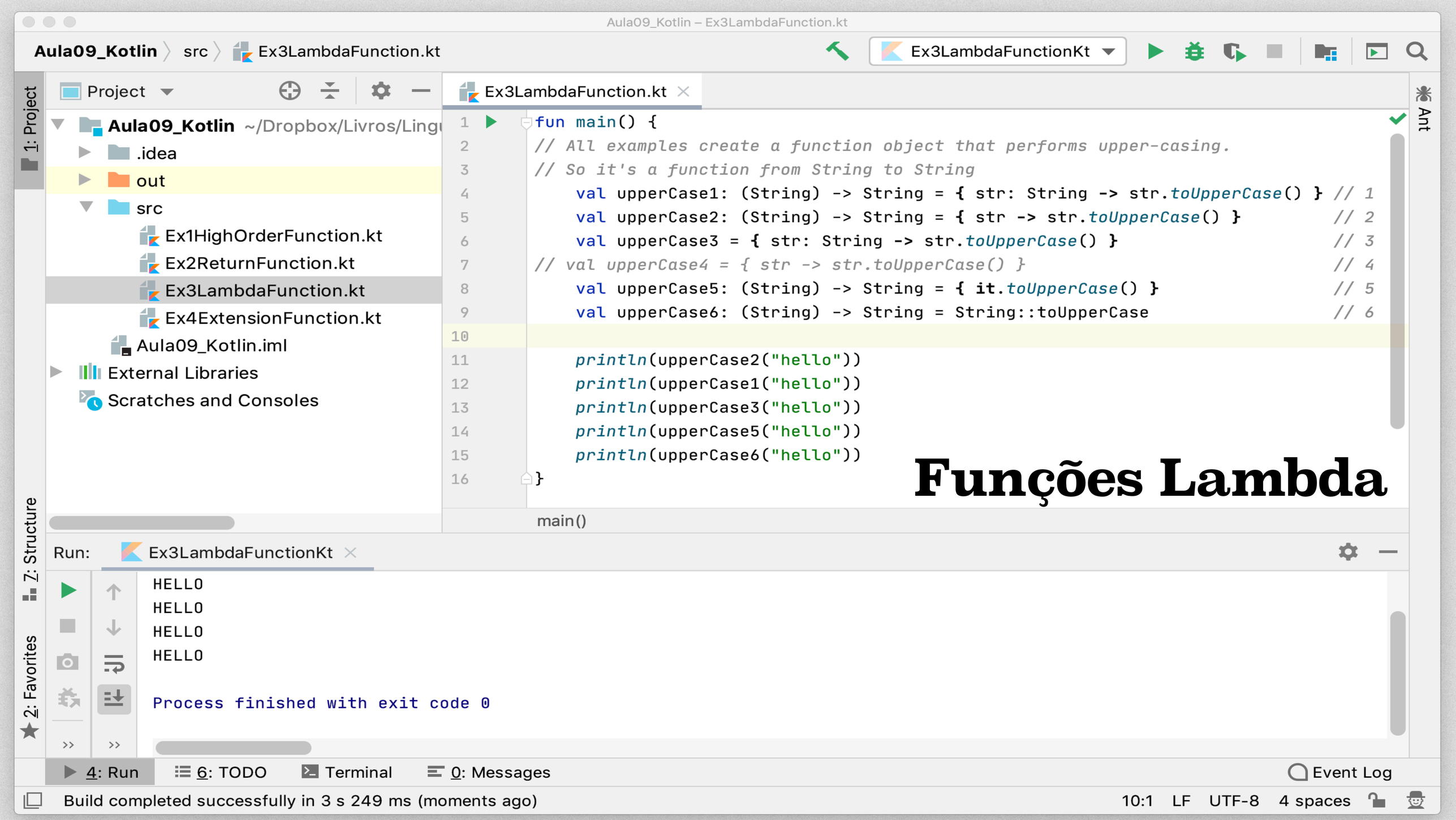
Funções Lambda

Ex3LambdaFunction.kt

```kotlin
fun main() {
// All examples create a function object that performs upper-casing.
// So it's a function from String to String
    val upperCase1: (String) -> String = { str: String -> str.toUpperCase() } // 1
    val upperCase2: (String) -> String = { str -> str.toUpperCase() }         // 2
    val upperCase3 = { str: String -> str.toUpperCase() }                     // 3
// val upperCase4 = { str -> str.toUpperCase() }                              // 4
    val upperCase5: (String) -> String = { it.toUpperCase() }                 // 5
    val upperCase6: (String) -> String = String::toUpperCase                  // 6

    println(upperCase2("hello"))
    println(upperCase1("hello"))
    println(upperCase3("hello"))
    println(upperCase5("hello"))
    println(upperCase6("hello"))
}
```

6. Se o lambda consistir em uma chamada simples de função, pode ser substituído por ::


Aula09_Kotlin – Ex3LambdaFunction.kt
Aula09_Kotlin 〉 src 〉 Ex3LambdaFunction.kt
Ex3LambdaFunctionKt
Project
Aula09_Kotlin ~/Dropbox/Livros/Lingu
.idea
out
src
Ex1HighOrderFunction.kt
Ex2ReturnFunction.kt
Ex3LambdaFunction.kt
Ex4ExtensionFunction.kt
Aula09_Kotlin.iml
External Libraries
Scratches and Consoles
Ex3LambdaFunction.kt
1 fun main() {
2 // All examples create a function object that performs upper-casing.
3 // So it's a function from String to String
4 val upperCase1: (String) -> String = { str: String -> str.toUpperCase() } // 1
5 val upperCase2: (String) -> String = { str -> str.toUpperCase() } // 2
6 val upperCase3 = { str: String -> str.toUpperCase() } // 3
7 // val upperCase4 = { str -> str.toUpperCase() } // 4
8 val upperCase5: (String) -> String = { it.toUpperCase() } // 5
9 val upperCase6: (String) -> String = String::toUpperCase // 6
10
11 println(upperCase2("hello"))
12 println(upperCase1("hello"))
13 println(upperCase3("hello"))
14 println(upperCase5("hello"))
15 println(upperCase6("hello"))
16 }
Funções Lambda
main()
Run: Ex3LambdaFunctionKt
HELLO
HELLO
HELLO
HELLO
Process finished with exit code 0
4: Run
6: TODO
Terminal
0: Messages
Event Log
Build completed successfully in 3 s 249 ms (moments ago)
10:1 LF UTF-8 4 spaces
1: Project
7: Structure
2: Favorites
Ant

1. Defines simple models of `Item` and `Order`. `Order` can contain a collection of `Item` objects.
2. Adds extension functions for the `Order` type.
3. Adds an extension property for the `Order` type.
4. Calls extension functions directly on an instance of `Order`.
5. Accesses the extension property on an instance of `Order`.

It is even possible to execute extensions on `null` references. In an extension function, you can check the object for `null` and use the result in your code:

```
fun <T> T?.nullSafeToString() = this?.toString() ?: "NULL"  // 1
```

```
1   data class Item(val name: String, val price: Float)                    // 1
2
3   data class Order(val items: Collection<Item>)
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20  }
```

1. Define as classes Item e Order. Order tem uma coleção de objetos Item.

```
1   data class Item(val name: String, val price: Float)                    // 1
2
3   data class Order(val items: Collection<Item>)
4
5   fun Order.maxPricedItemValue(): Float = this.items.maxBy { it.price }?.price ?: 0F    // 2
6   fun Order.maxPricedItemName() = this.items.maxBy { it.price }?.name ?: "NO_PRODUCTS"
7
8
9
10
11
12
13
14
15
16
17
18
19
20  }
```

2. Adicionamos duas funções de extensão para a classe Order.

```
1     data class Item(val name: String, val price: Float)                    // 1
2
3     data class Order(val items: Collection<Item>)
4
5     fun Order.maxPricedItemValue(): Float = this.items.maxBy { it.price }?.price ?: 0F    // 2
6     fun Order.maxPricedItemName() = this.items.maxBy { it.price }?.name ?: "NO_PRODUCTS"
7
8     val Order.commaDelimitedItemNames: String                              // 3
9         get() = items.map { it.name }.joinToString()
10
11
12
13
14
15
16
17
18
19
20    }
```

3. Adiciona uma propriedade de extensão para a classe Order.

Funções de Extensão

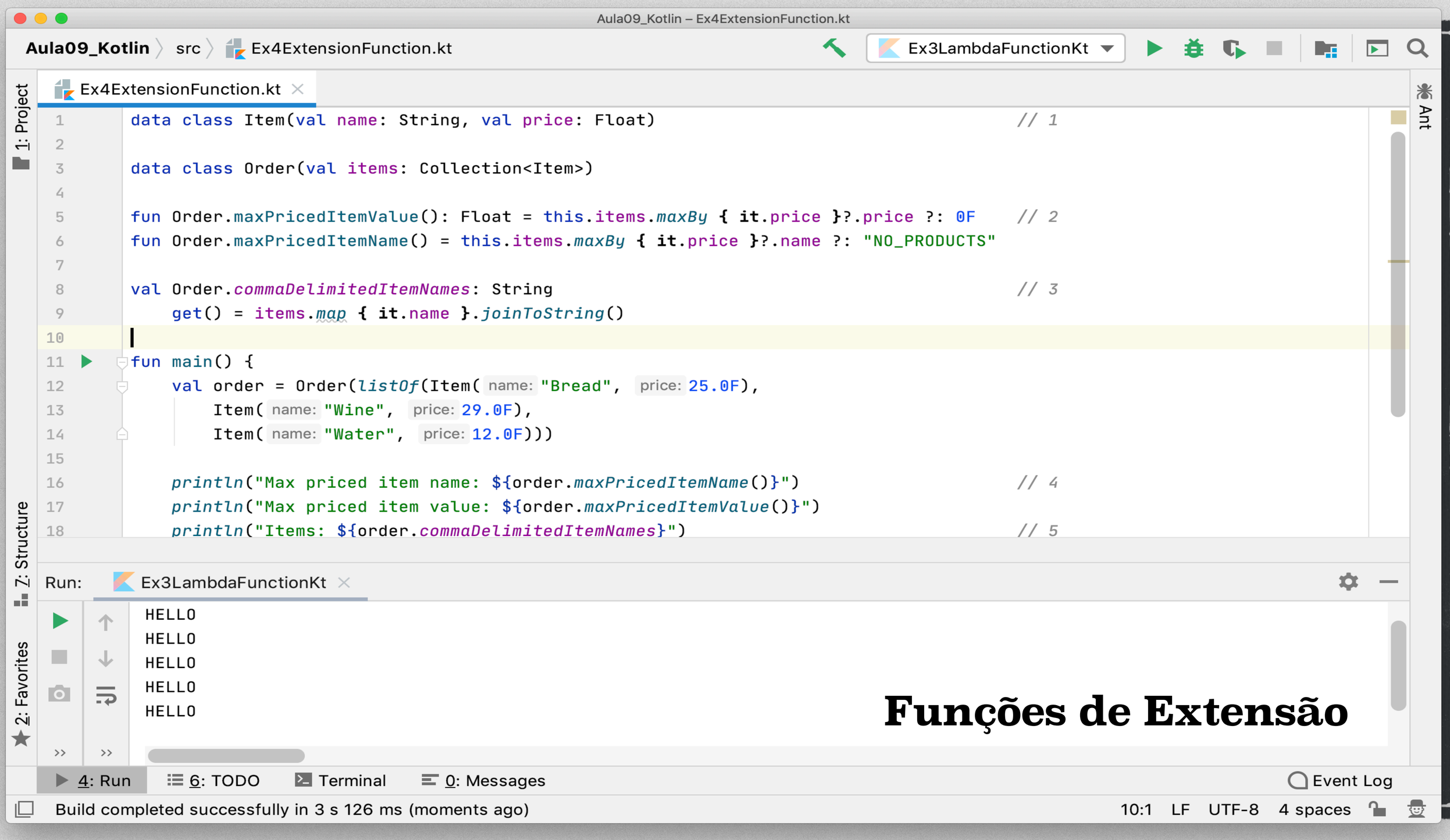
```
1   data class Item(val name: String, val price: Float)                    // 1
2
3   data class Order(val items: Collection<Item>)
4
5   fun Order.maxPricedItemValue(): Float = this.items.maxBy { it.price }?.price ?: 0F    // 2
6   fun Order.maxPricedItemName() = this.items.maxBy { it.price }?.name ?: "NO_PRODUCTS"
7
8   val Order.commaDelimitedItemNames: String                              // 3
9       get() = items.map { it.name }.joinToString()
10
11  fun main() {
12      val order = Order(listOf(Item( name: "Bread",  price: 25.0F),
13          Item( name: "Wine",  price: 29.0F),
14          Item( name: "Water",  price: 12.0F)))
15
16      println("Max priced item name: ${order.maxPricedItemName()}")          // 4
17      println("Max priced item value: ${order.maxPricedItemValue()}")
18
19
20  }
```

4. Chama as funções de extensão a partir de uma instância de Order.

Funções de Extensão

```
1   data class Item(val name: String, val price: Float)                    // 1
2
3   data class Order(val items: Collection<Item>)
4
5   fun Order.maxPricedItemValue(): Float = this.items.maxBy { it.price }?.price ?: 0F    // 2
6   fun Order.maxPricedItemName() = this.items.maxBy { it.price }?.name ?: "NO_PRODUCTS"
7
8   val Order.commaDelimitedItemNames: String                              // 3
9       get() = items.map { it.name }.joinToString()
10
11  fun main() {
12      val order = Order(listOf(Item( name: "Bread",  price: 25.0F),
13          Item( name: "Wine",  price: 29.0F),
14          Item( name: "Water",  price: 12.0F)))
15
16      println("Max priced item name: ${order.maxPricedItemName()}")          // 4
17      println("Max priced item value: ${order.maxPricedItemValue()}")
18      println("Items: ${order.commaDelimitedItemNames}")                     // 5
19  }
20
```

5. Acessa a propriedade de extensão criada a partir de um objeto Order.


Aula09_Kotlin – Ex4ExtensionFunction.kt
Aula09_Kotlin
src
Ex4ExtensionFunction.kt
Ex3LambdaFunctionKt
1: Project
Ex4ExtensionFunction.kt
data class Item(val name: String, val price: Float) // 1
data class Order(val items: Collection<Item>)
fun Order.maxPricedItemValue(): Float = this.items.maxBy { it.price }?.price ?: 0F // 2
fun Order.maxPricedItemName() = this.items.maxBy { it.price }?.name ?: "NO_PRODUCTS"
val Order.commaDelimitedItemNames: String // 3
get() = items.map { it.name }.joinToString()
fun main() {
val order = Order(listOf(Item(name: "Bread", price: 25.0F),
Item(name: "Wine", price: 29.0F),
Item(name: "Water", price: 12.0F)))
println("Max priced item name: ${order.maxPricedItemName()}") // 4
println("Max priced item value: ${order.maxPricedItemValue()}")
println("Items: ${order.commaDelimitedItemNames}") // 5
Ant
7: Structure
2: Favorites
Run:
Ex3LambdaFunctionKt
HELLO
HELLO
HELLO
HELLO
HELLO
Funções de Extensão
4: Run
6: TODO
Terminal
0: Messages
Event Log
Build completed successfully in 3 s 126 ms (moments ago)
10:1
LF
UTF-8
4 spaces

https://github.com/rafael-vieira-coelho/kotlin/tree/Aula-09---Funcional

rafaelvc2@gmail.com