



Curso de Kotlin

Autor: Leonardo Leitão

Co-autor: Arthur Oliveira



Índice

1. Introdução	2
1.1. Visão Geral do Curso	2
1.2. Assine o Nosso Canal	2
1.3. Repositório do Curso	3
1.4. Autor	4
1.5. Suporte	4
2. Configuração do Ambiente	5
2.1. Windows.....	5
2.2. MacOS	14
2.3. Linux	23
3. Conceitos Básicos	30
3.1. Primeiro Programa em Kotlin.....	30
3.2. Comentários	30
3.3. Variáveis e Constantes	31
3.4. Tipos Básicos	34
3.5. Função	35
3.6. Template Strings.....	37
3.7. Notação Ponto	37
3.8. Null Safety	38
3.9. Import	39
3.10. Estruturas de Controle	41
3.11. While.....	43
3.12. Operadores	48
3.13. Conversão	53
4. Funções	56
4.1. Funções estáticas <i>top-level</i>	56
4.2. Argumentos Nomeados	57
4.3. Parâmetros são Imutáveis	57
4.4. Parâmetro Padrão	58
4.5. Múltiplos Retornos Com Destructuring	58
4.6. Adicionando Métodos em Classes Existentes	59
4.7. Chamando Função em Java	59
4.8. Argumentos Variáveis (<i>varargs</i>)	60
4.9. Funções Infix.....	60
4.10. Funções Sempre Retornam um Valor	61
4.11. Inline Function	64
5. Classes e Objetos	67
5.1. Classe vs Objeto	67

5.2. Exemplos Básicos	68
5.3. Membros	68
5.4. Tipos de Variáveis/Constantes	70
5.5. Data Classe	71
5.6. Construtor	72
5.7. Getters & Setters	74
5.8. Membros Classe vs Instância	76
5.9. Passagem por Referência	77
5.10. Enum	78
6. Programando com Lambdas	80
6.1. Minhas Lambdas	80
6.2. Lambdas Encontradas na API	81
7. Arrays e Coleções	87
7.1. Arrays e Collections	87
7.2. Arrays	87
7.3. Array List	87
7.4. For (each)	90
7.5. Matriz	91
7.6. Set	92
7.7. HashMap	94
7.8. Hashcode & Equals	96
8. Orientação a Objetos	98
8.1. Os Pilares da OO	98
8.2. Encapsulamento	98
8.3. Herança	100
8.4. Polimorfismo	103
9. Tópicos Avançados	106
9.1. Recursividade	106
9.2. Genéricos	106
9.3. Sobrecarga de Operadores	107
9.4. Anotação & Reflexão	108
10. Conclusão	110
10.1. Próximos Passos	110
Appendix A: Tabela de Códigos	111
Glossário	112

Sumário

Apostila do curso de de Kotlin da Cod3r.

<https://www.cod3r.com.br>

1. Introdução

1.1. Visão Geral do Curso

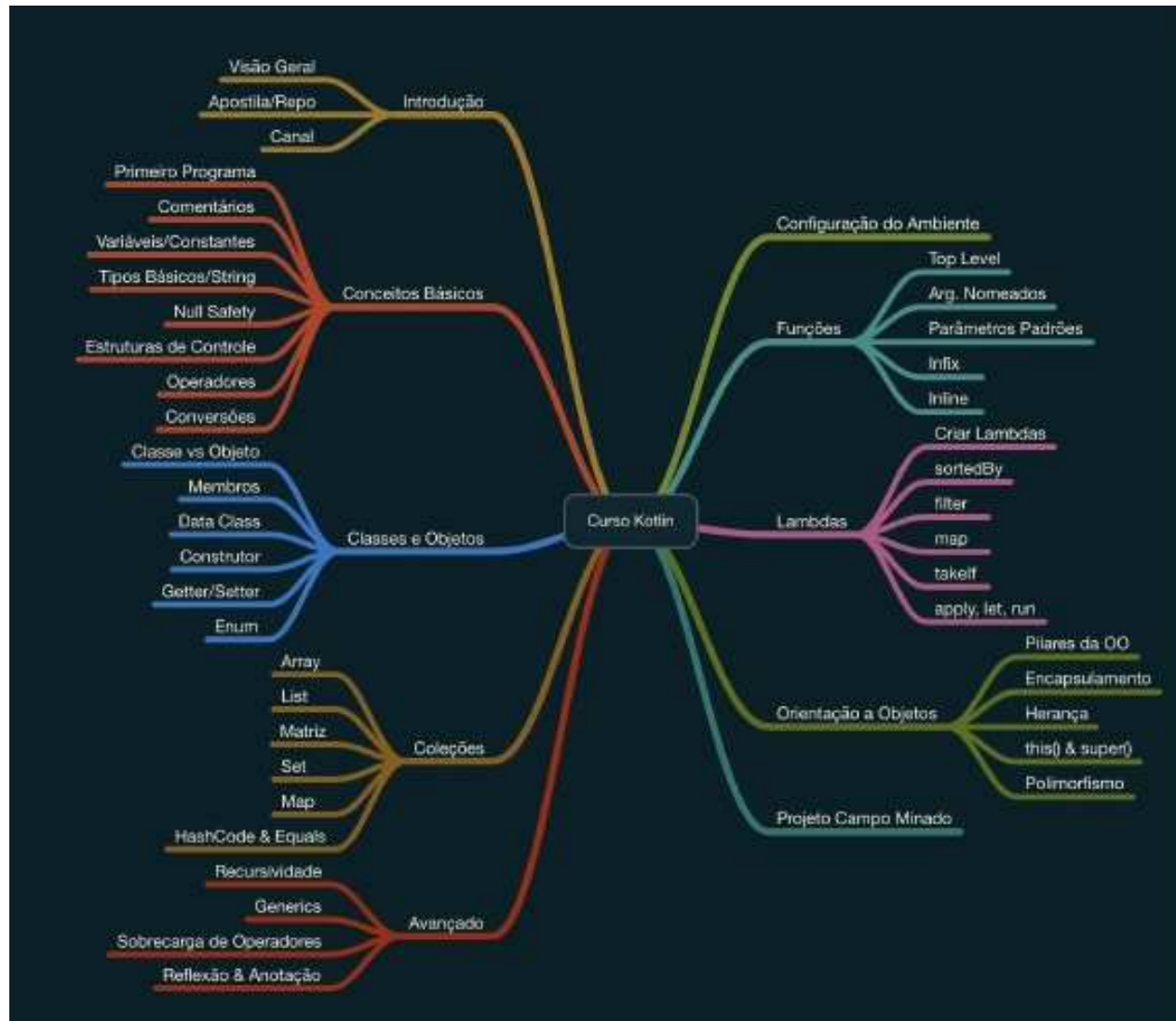


Figura 1. Visão Geral do Curso

1.2. Assine o Nosso Canal

Estamos também no Youtube! Temos um canal no youtube com várias séries de vídeos para complementar o seu aprendizado, então gostaria de te convidar para fazer parte da nossa comunidade em <https://www.youtube.com/aulasdeprogramacao>



Figura 2. Algumas Aulas do Canal

1.3. Repositório do Curso

Precisa ter acesso ao **código fonte** dos exercícios do curso? Todos os exemplos desenvolvidos durante as aulas estão disponíveis no **Github** da **Cod3r**: <https://github.com/cod3rcursos/curso-kotlin>



Se você não nunca trabalhou com repositórios Git, uma excelente ferramenta para facilitar a obtenção do código pode ser obtida no seguinte endereço:

<https://desktop.github.com/>

É possível ter acesso ao código fonte do curso sem precisar instalar nenhuma ferramenta, baixando diretamente o arquivo **.ZIP** da página do **Github**: <https://github.com/cod3rcursos/curso-kotlin/archive/master.zip>

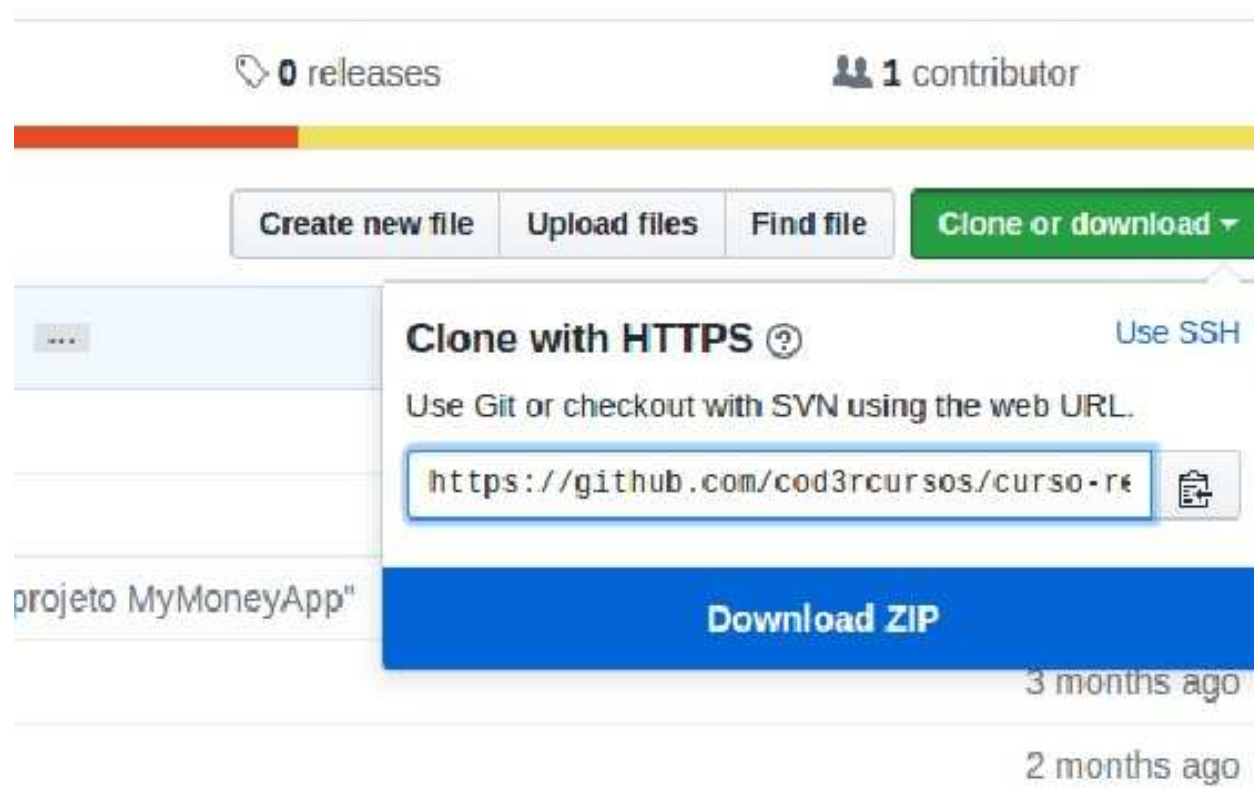


Figura 3. Opção para Baixar Repositório como .ZIP

1.4. Autor



Leonardo Leitão é graduado em Engenharia Elétrica pela Universidade Federal do Ceará e Mestre em Informática Aplicada pela Universidade de Fortaleza, na qual trabalhou com Integração de Redes de Sensores sem Fio e Computação em Nuvem. Há doze anos atua na área como desenvolvedor de softwares e atualmente trabalha na DATAPREV como arquiteto, desenvolvendo sistemas para o INSS. Professor de desenvolvimento de software há quase dez anos.

Em 2016 foi um dos fundadores da **Cod3r**, na qual atua na criação de cursos online voltados para desenvolvimento de software. Com mais de 5.000 alunos, a Cod3r tem se tornado uma referência em ensino de qualidade no Brasil na área de programação.

1.5. Suporte

Não basta um curso ter qualidade, é fundamental um bom suporte para que você tenha uma excelente experiência no aprendizado. E nossa equipe estará à disposição para sanar eventuais dúvidas que você tenha durante as aulas.

E pra que a gente consiga oferecer um melhor suporte e de forma mais rápida, é importante que algumas regras sejam seguidas:

- **Manter perguntas separadas** - Não aproveite perguntas de outros alunos para fazer novas perguntas. Mesmo quando o erro parece ser o mesmo, as causas geralmente são diferentes.
- **Disponibilize seu código no Git** - Quando um problema ocorre, é muito comum eu ter que analisar o seu código para saber de fato o que está gerando o problema reportado, então disponibilize o seu código em algum repositório público. Poder ser o Github, Bitbucket, Gitlab... Você escolhe!
- **Suporte dentro do escopo** - Manter as dúvidas sobre questões relacionadas à execução dos exercícios e as dúvidas conceituais relativas ao escopo do curso. É inviável para nós professores, analisar problemas específicos ou implementar soluções para atender necessidades específicas de um aluno.
- **Curso está aberto** - Se entendermos que existe a demanda de adicionar um tópico que seja necessidade de um conjunto expressivo de alunos e que julgarmos que faz parte do escopo do curso, pode ter certeza que iremos acrescentar o material com o maior prazer.

Conto com vocês e pode ter certeza que estaremos presente na plataforma para te ajudar!

2. Configuração do Ambiente

2.1. Windows

2.1.1. Instalação do Java



Baixar o Java JDK em:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

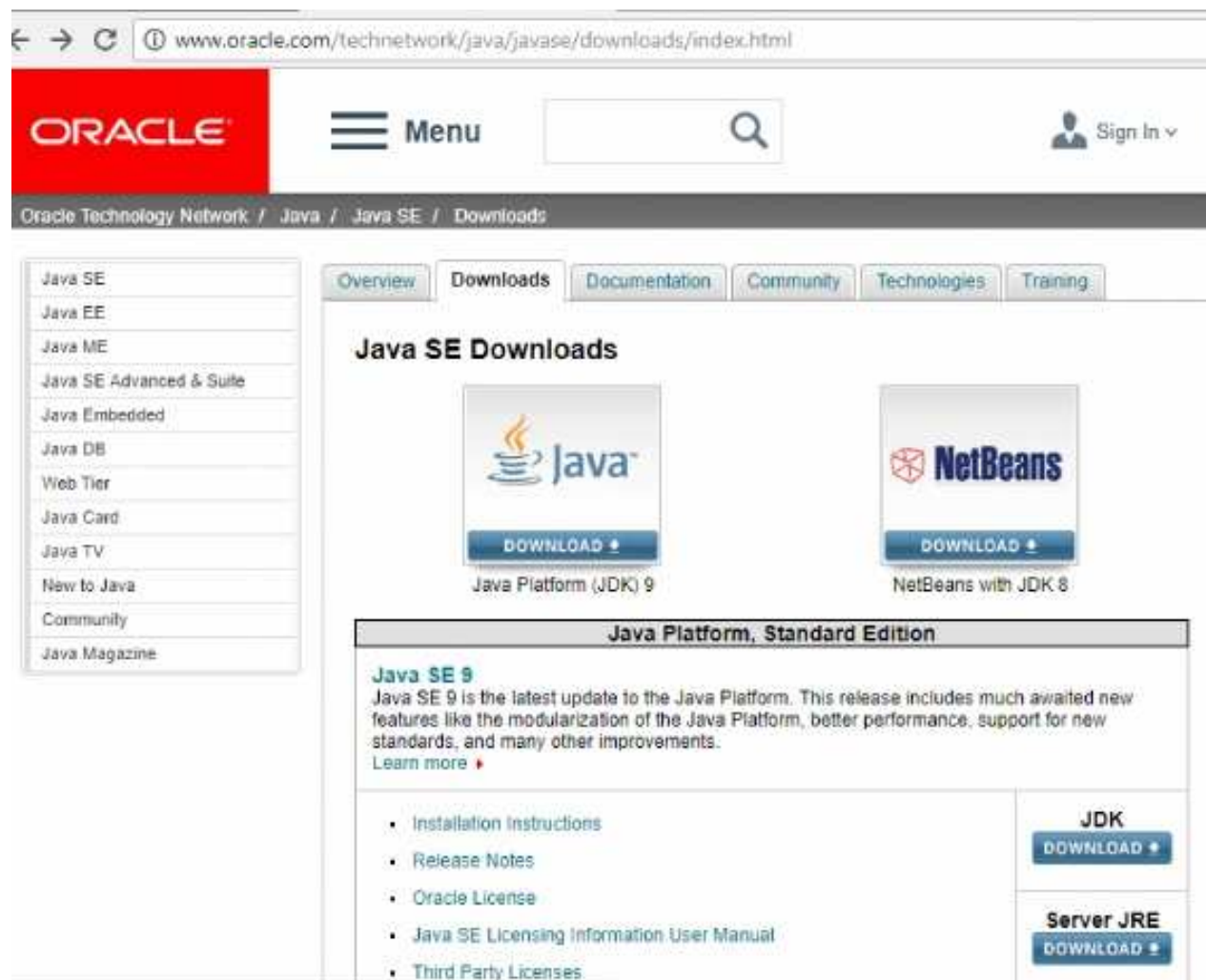


Figura 4. Instalação do Java (Parte 1 de 7)



Figura 5. Instalação do Java (Parte 2 de 7)

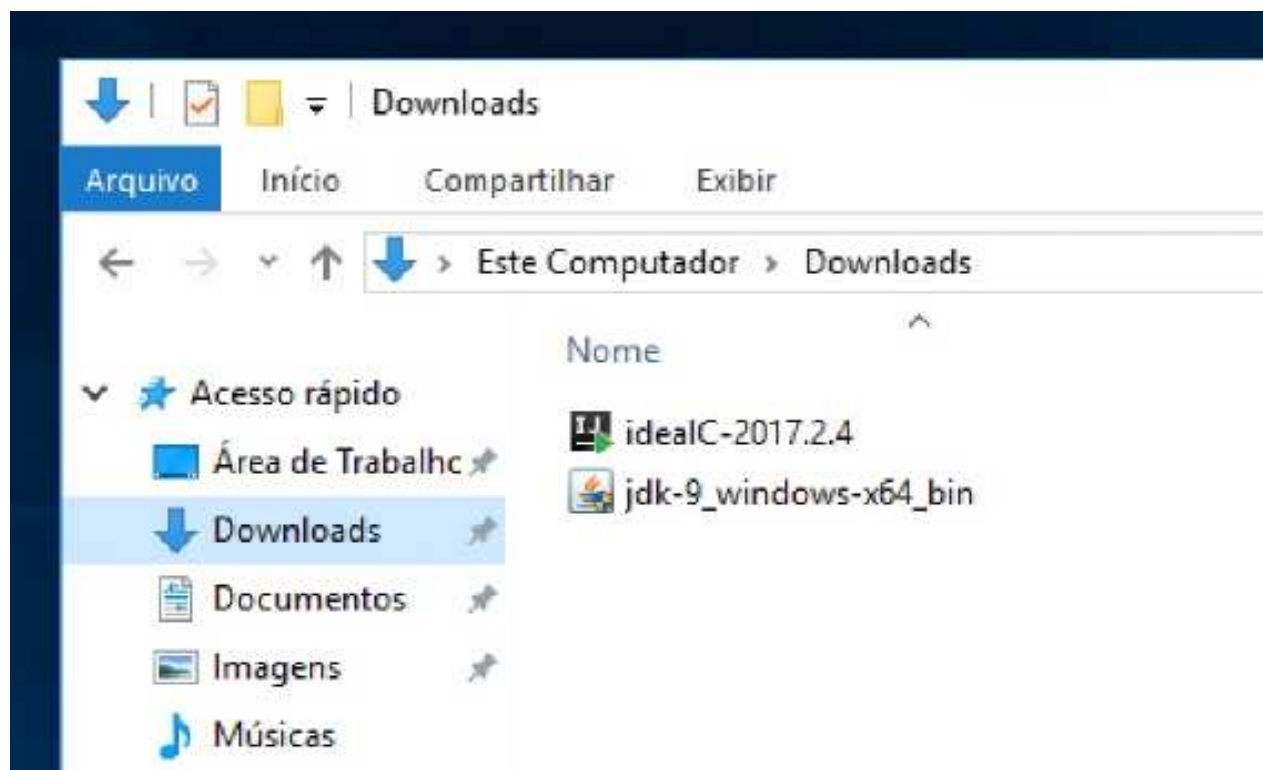


Figura 6. Instalação do Java (Parte 3 de 7)

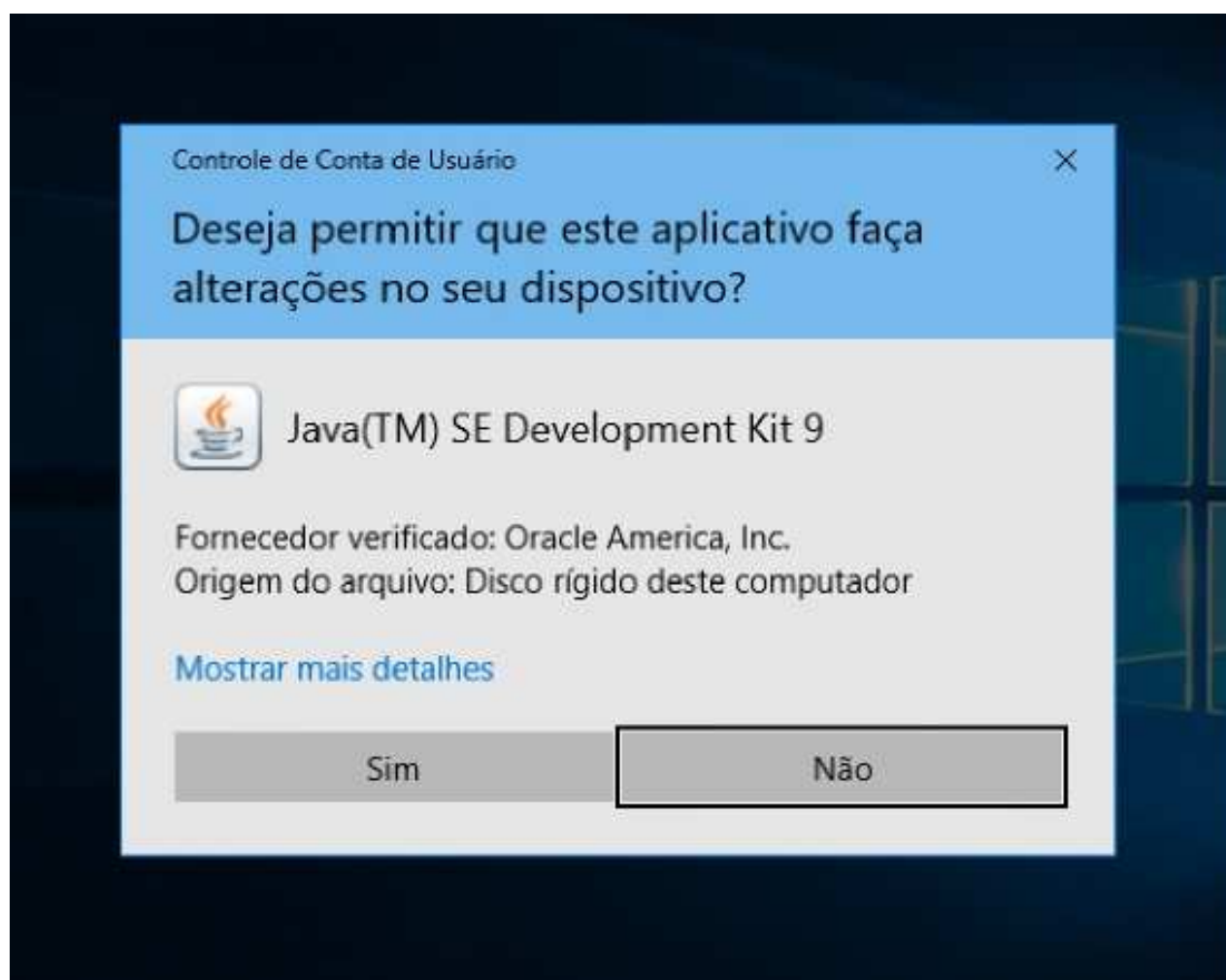


Figura 7. Instalação do Java (Parte 4 de 7)



Figura 8. Instalação do Java (Parte 5 de 7)

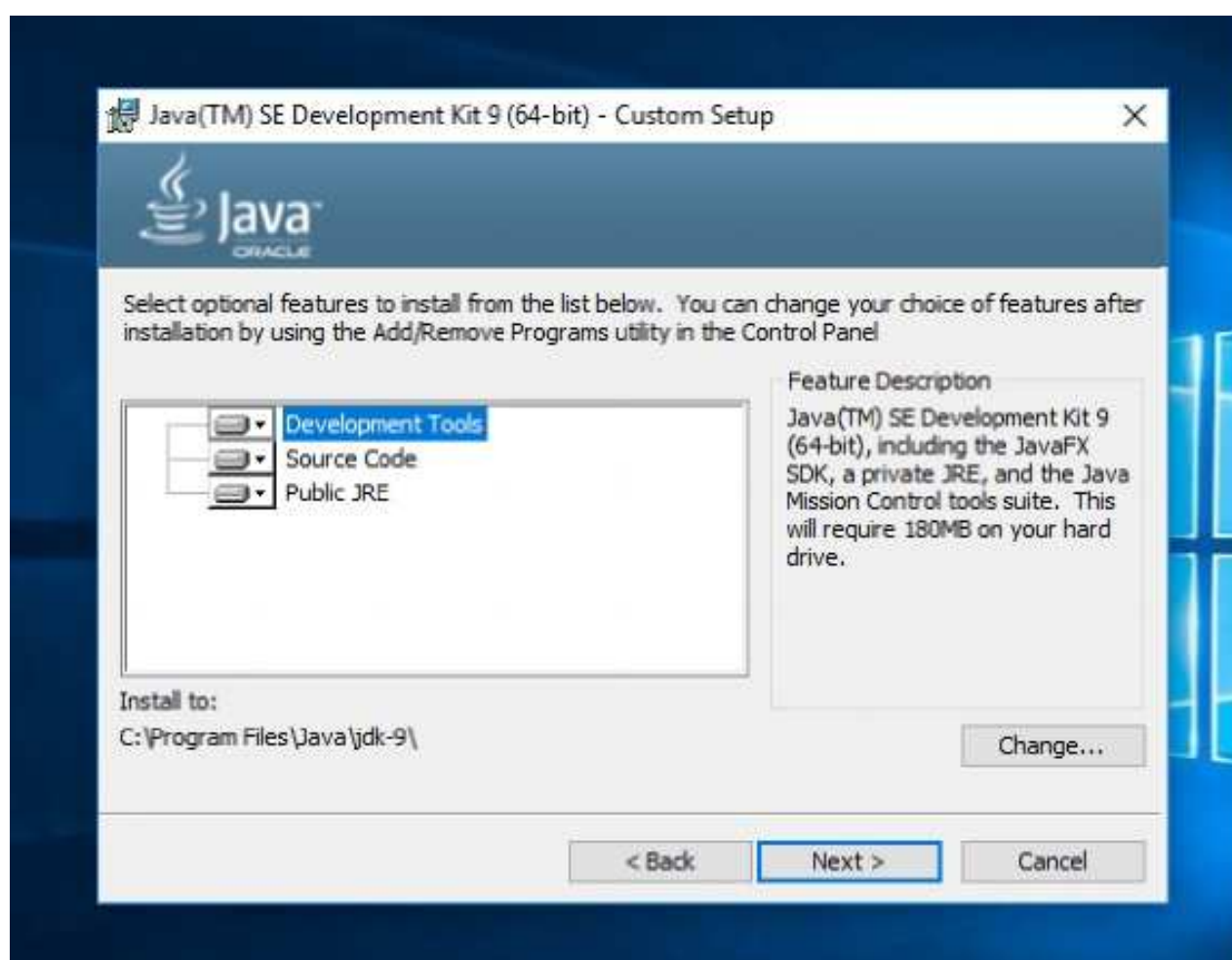


Figura 9. Instalação do Java (Parte 6 de 7)

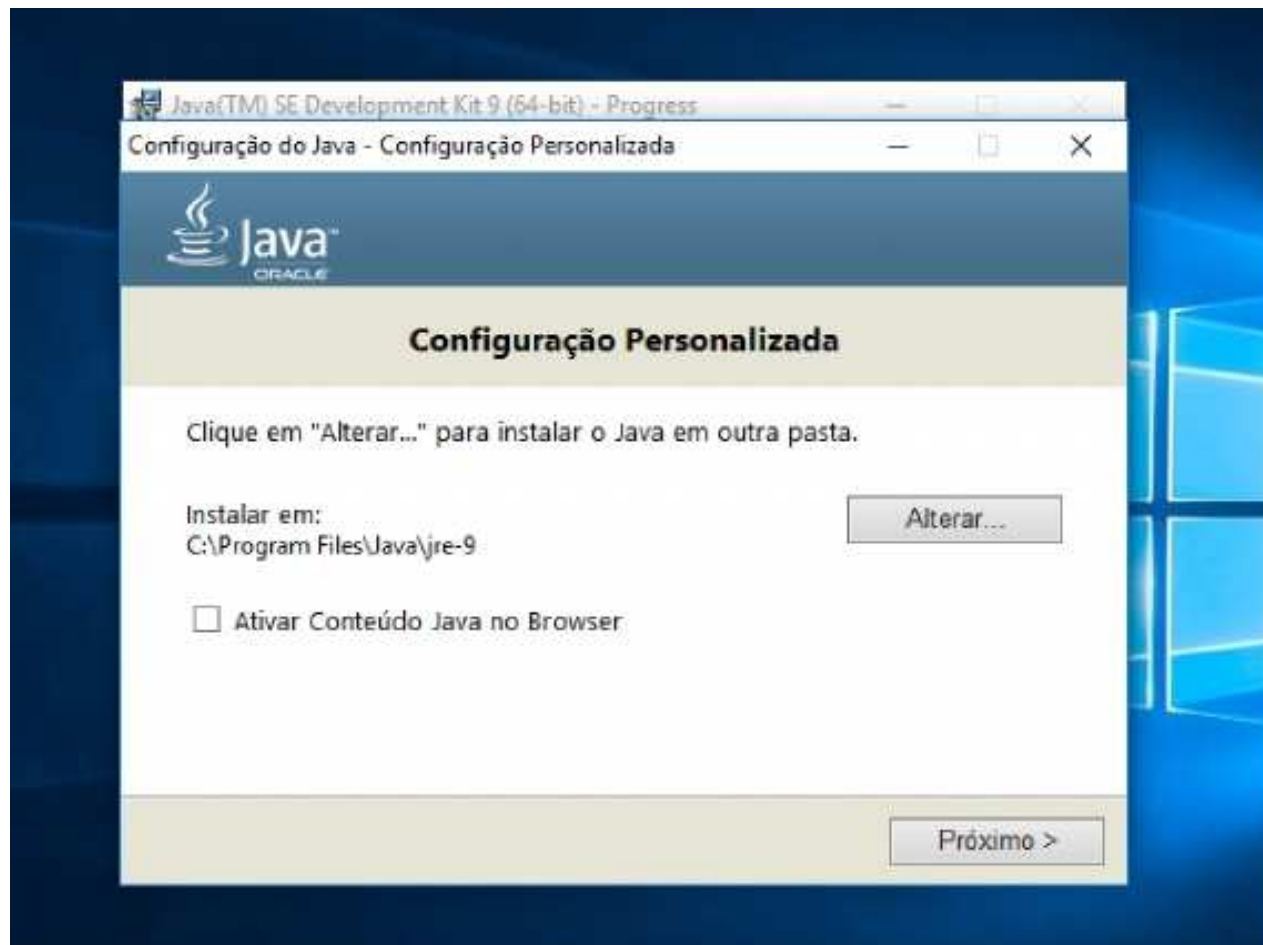


Figura 10. Instalação do Java (Parte 7 de 7)

2.1.2. Instalação IntelliJ IDEA



Baixar o IntelliJ IDEA em:

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



Figura 11. Instalação do IntelliJ (Parte 1 de 13)

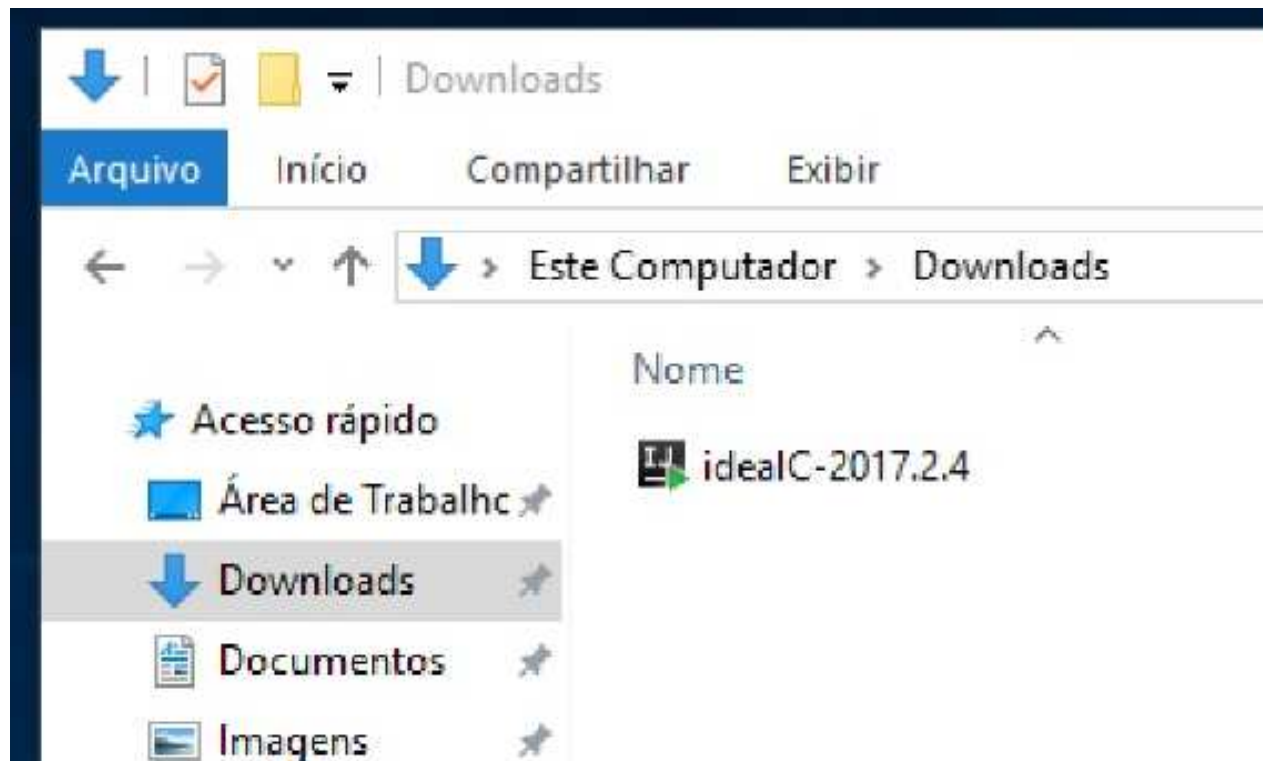


Figura 12. Instalação do IntelliJ (Parte 2 de 13)

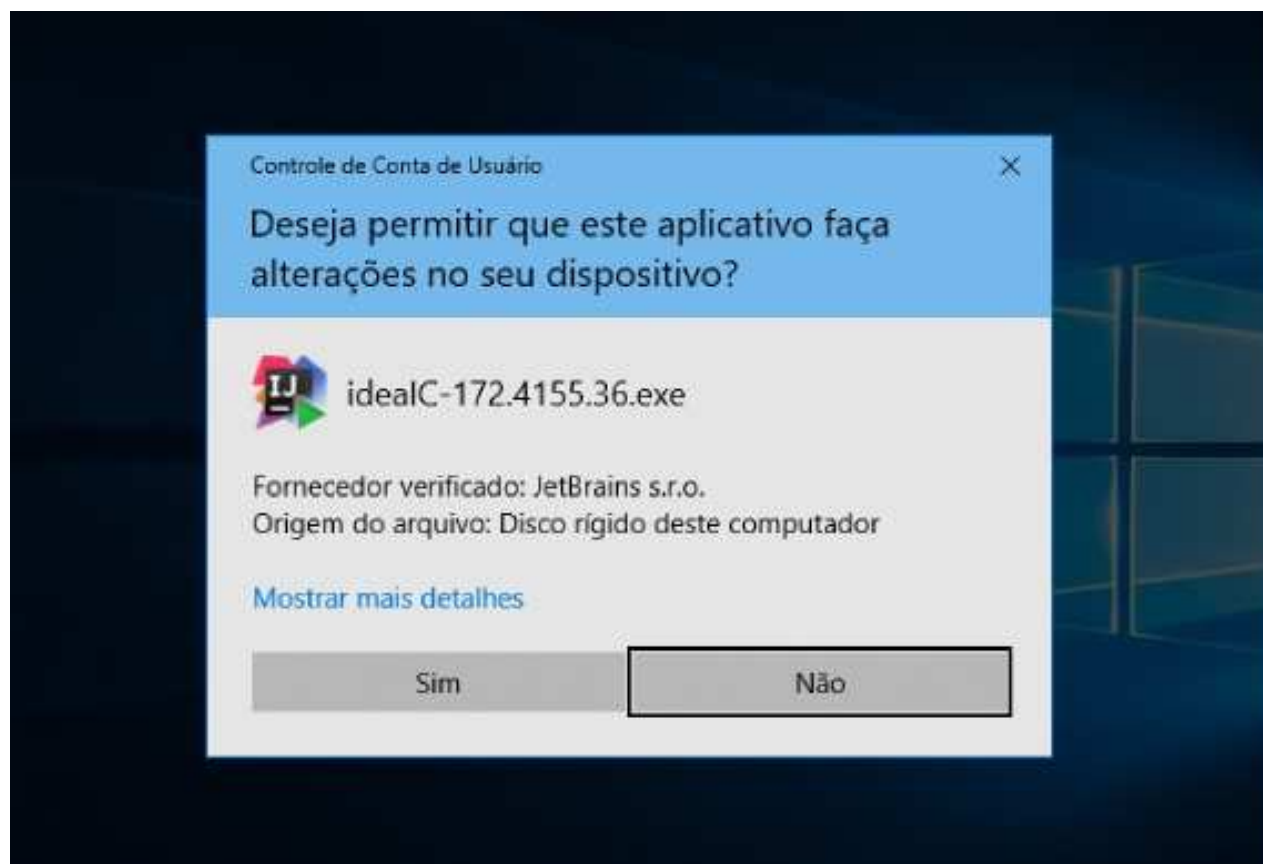


Figura 13. Instalação do IntelliJ (Parte 3 de 13)



Figura 14. Instalação do IntelliJ (Parte 4 de 13)

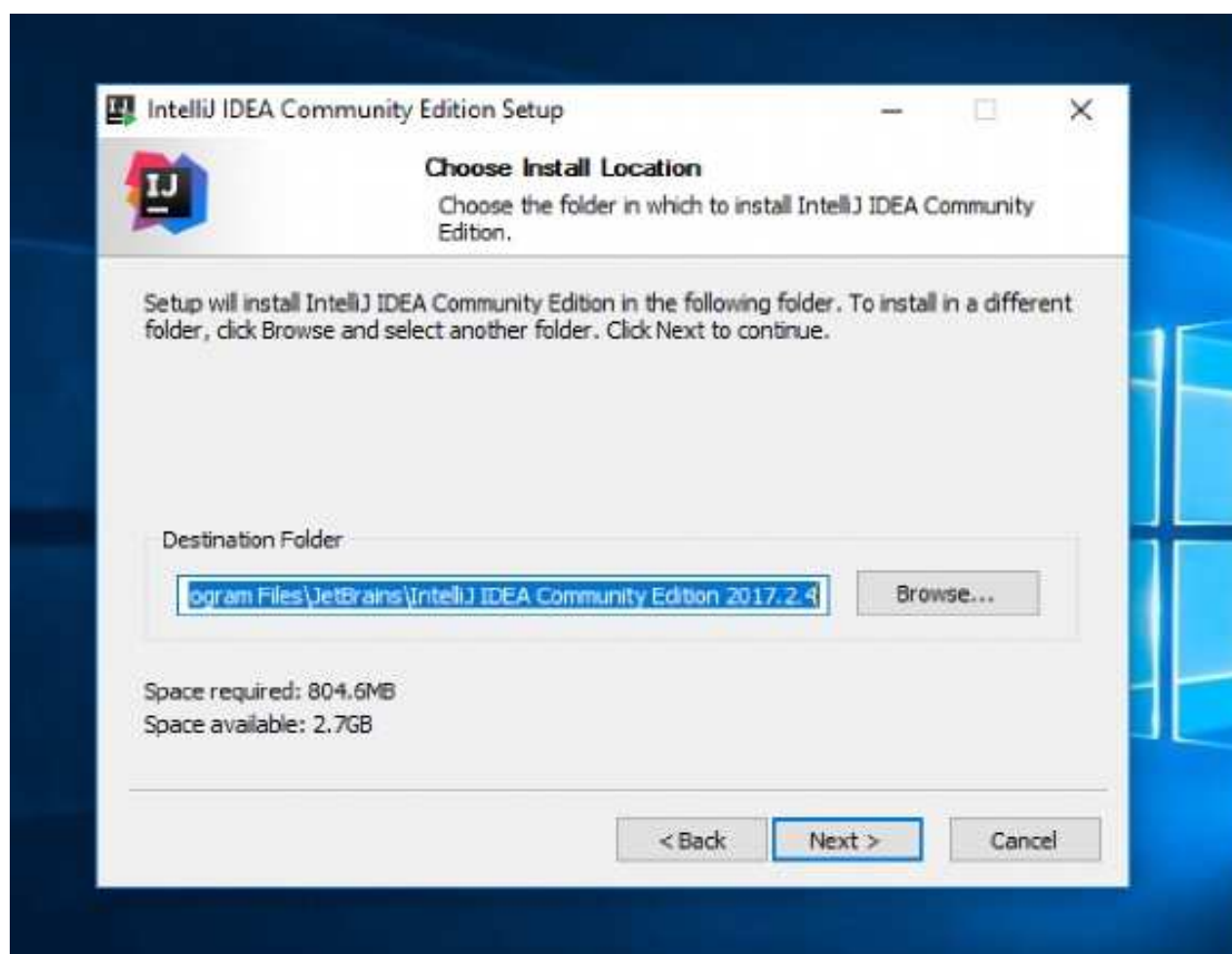


Figura 15. Instalação do IntelliJ (Parte 5 de 13)

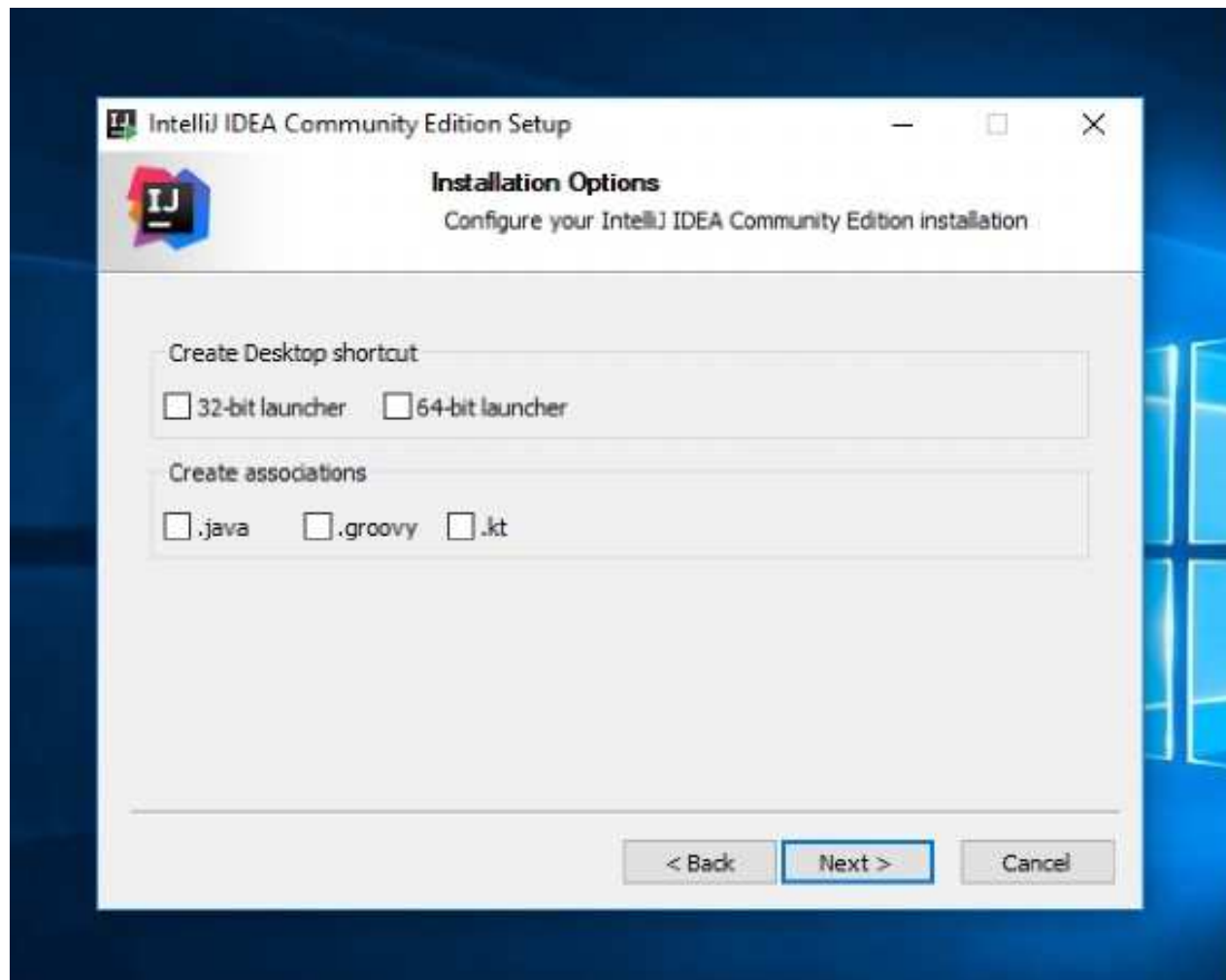


Figura 16. Instalação do IntelliJ (Parte 6 de 13)

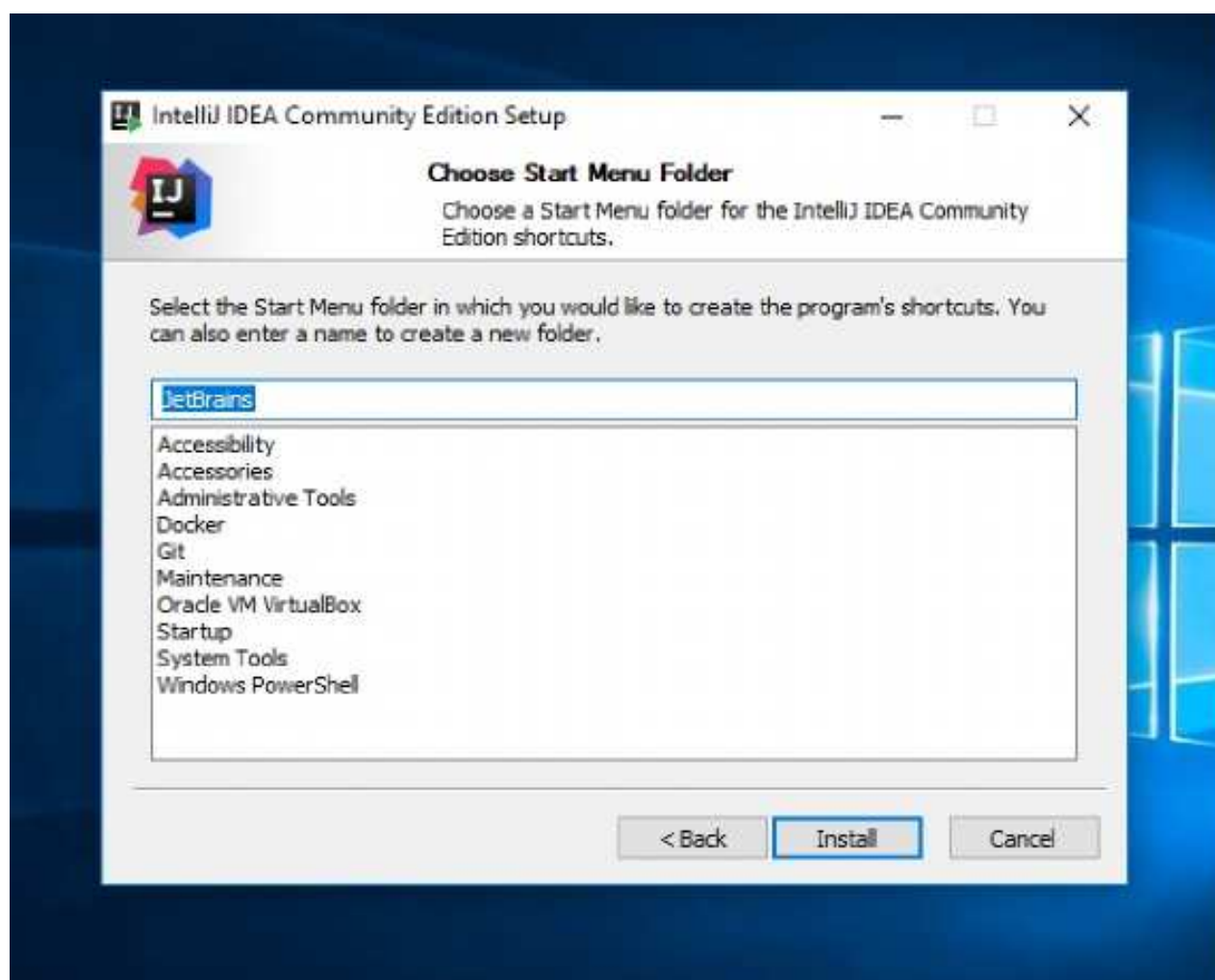


Figura 17. Instalação do IntelliJ (Parte 7 de 13)



Figura 18. Instalação do IntelliJ (Parte 8 de 13)

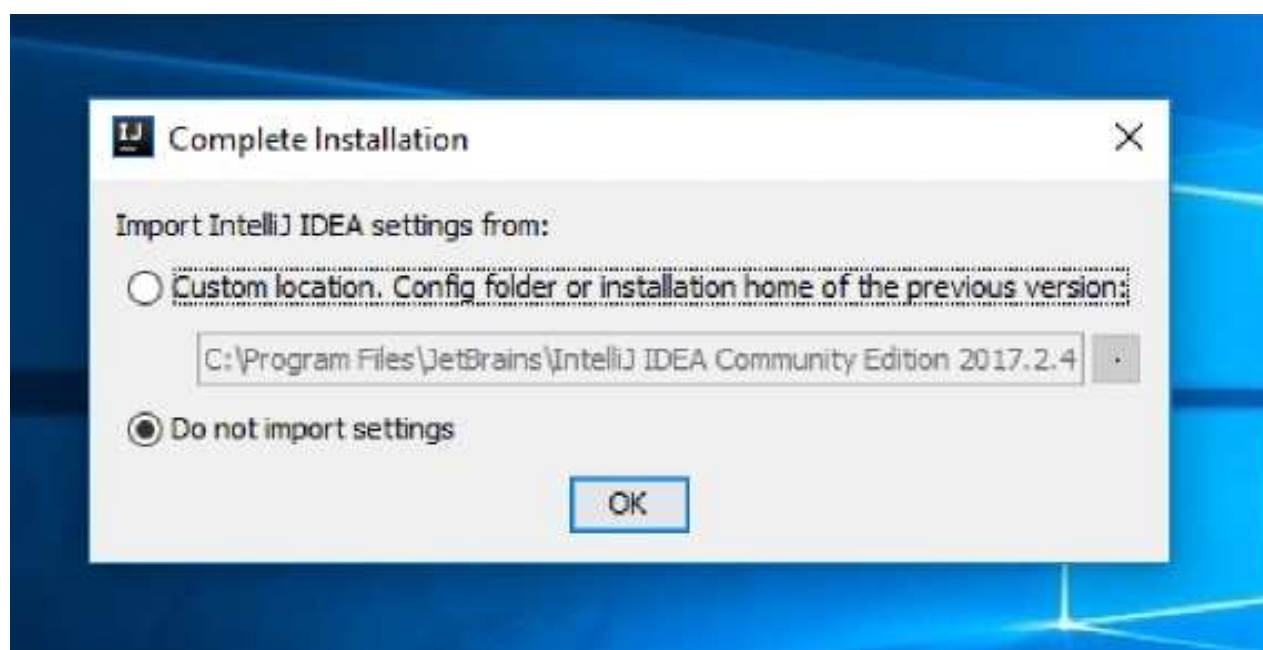


Figura 19. Instalação do IntelliJ (Parte 9 de 13)

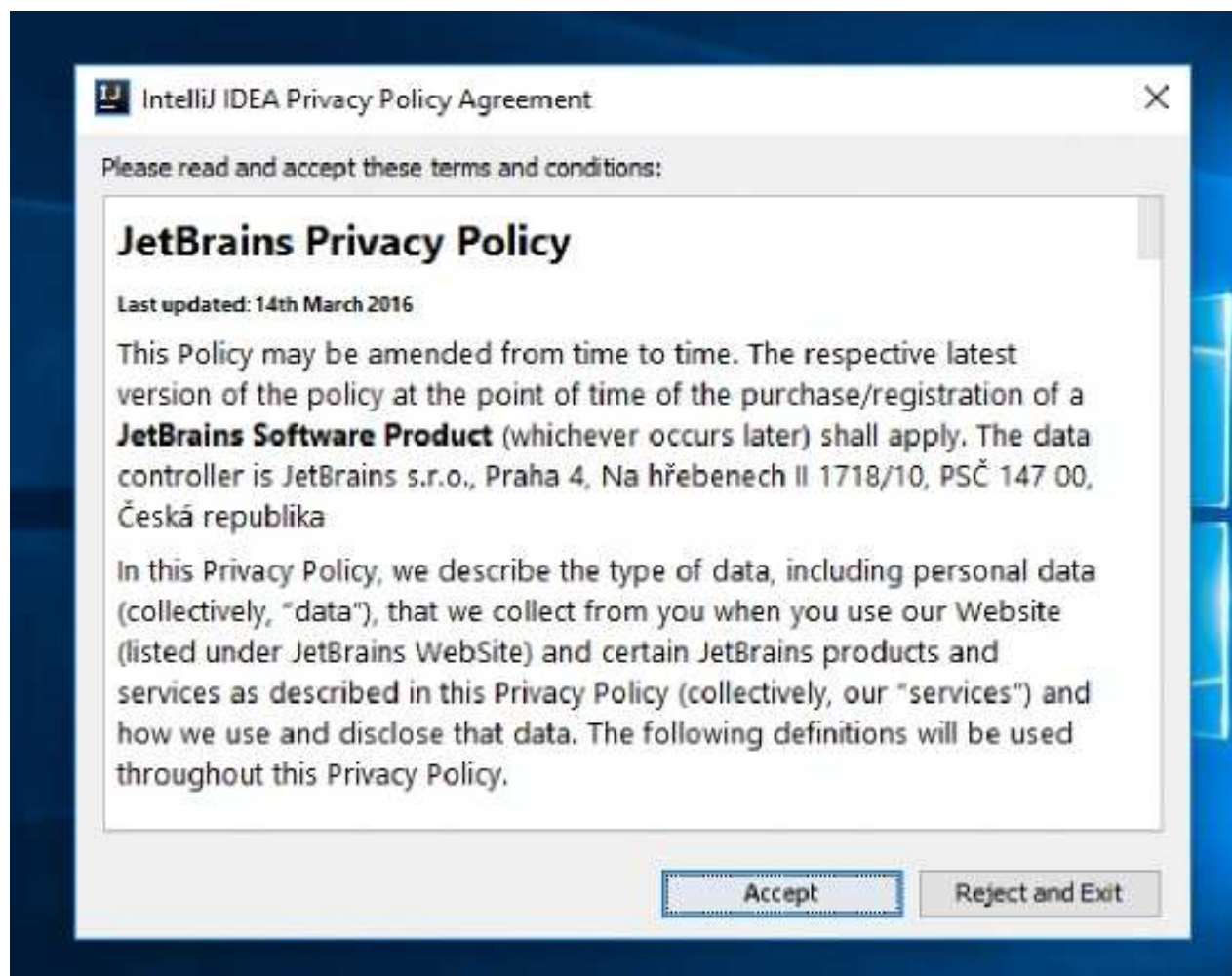


Figura 20. Instalação do IntelliJ (Parte 10 de 13)

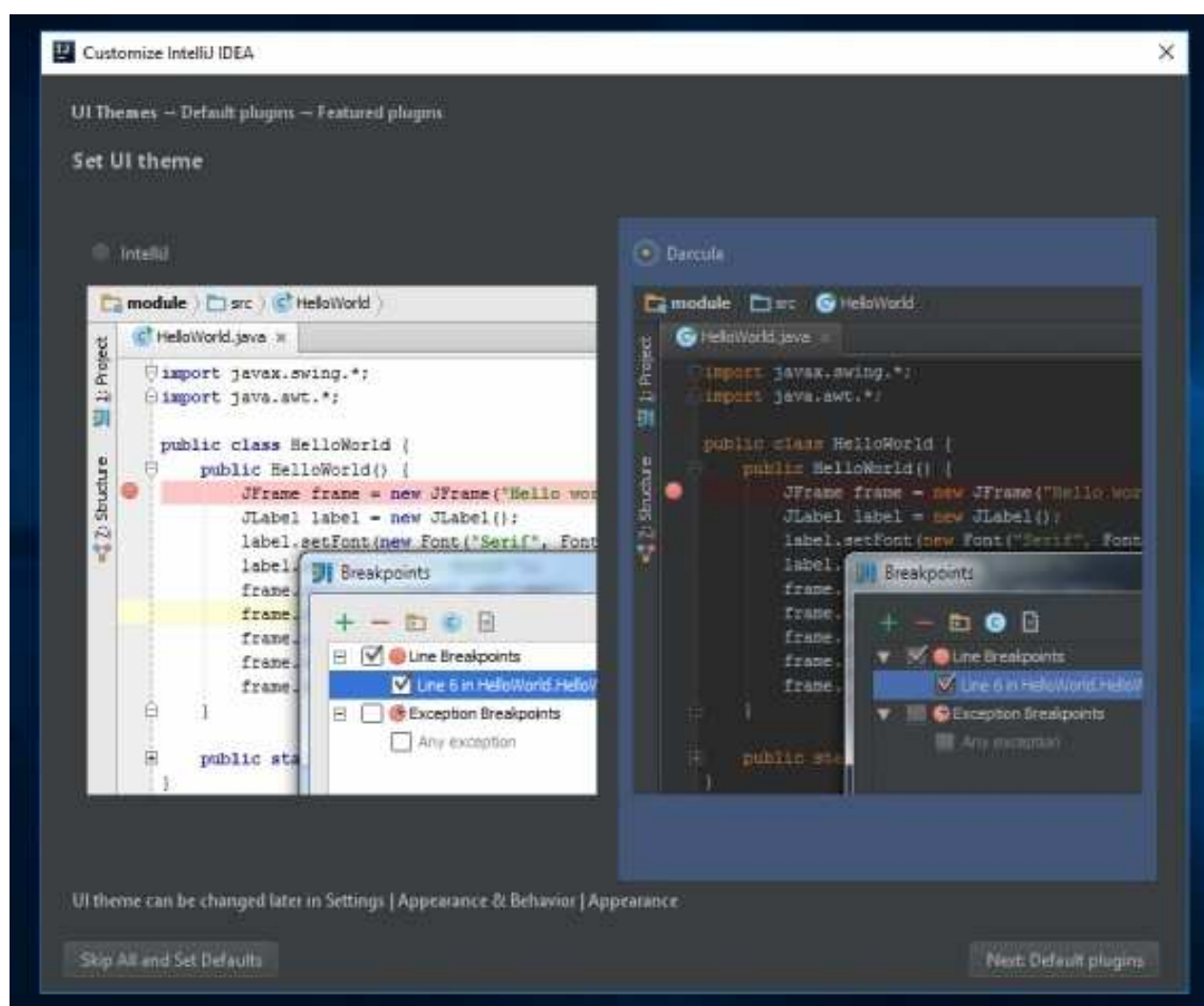


Figura 21. Instalação do IntelliJ (Parte 11 de 13)

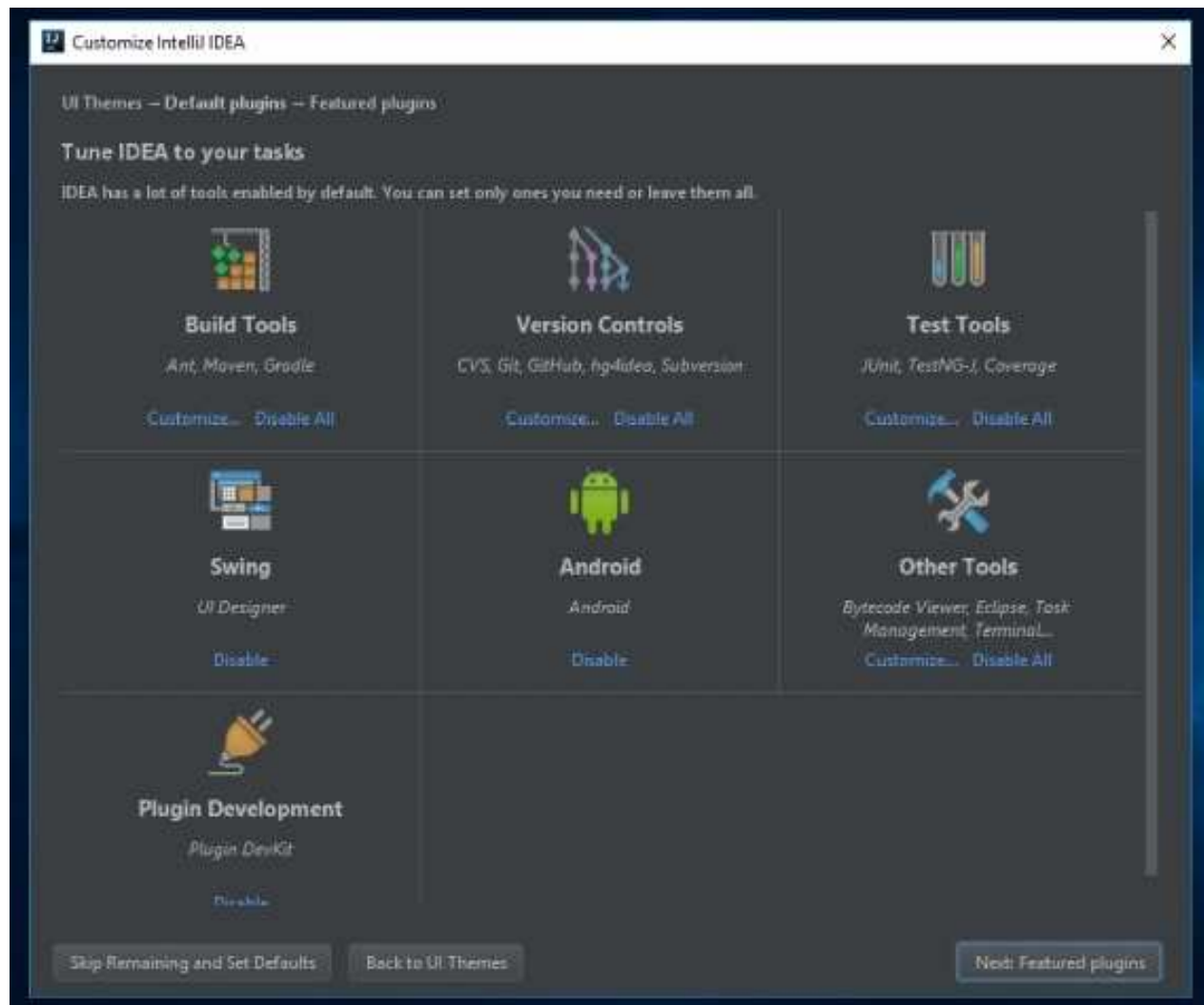


Figura 22. Instalação do IntelliJ (Parte 12 de 13)

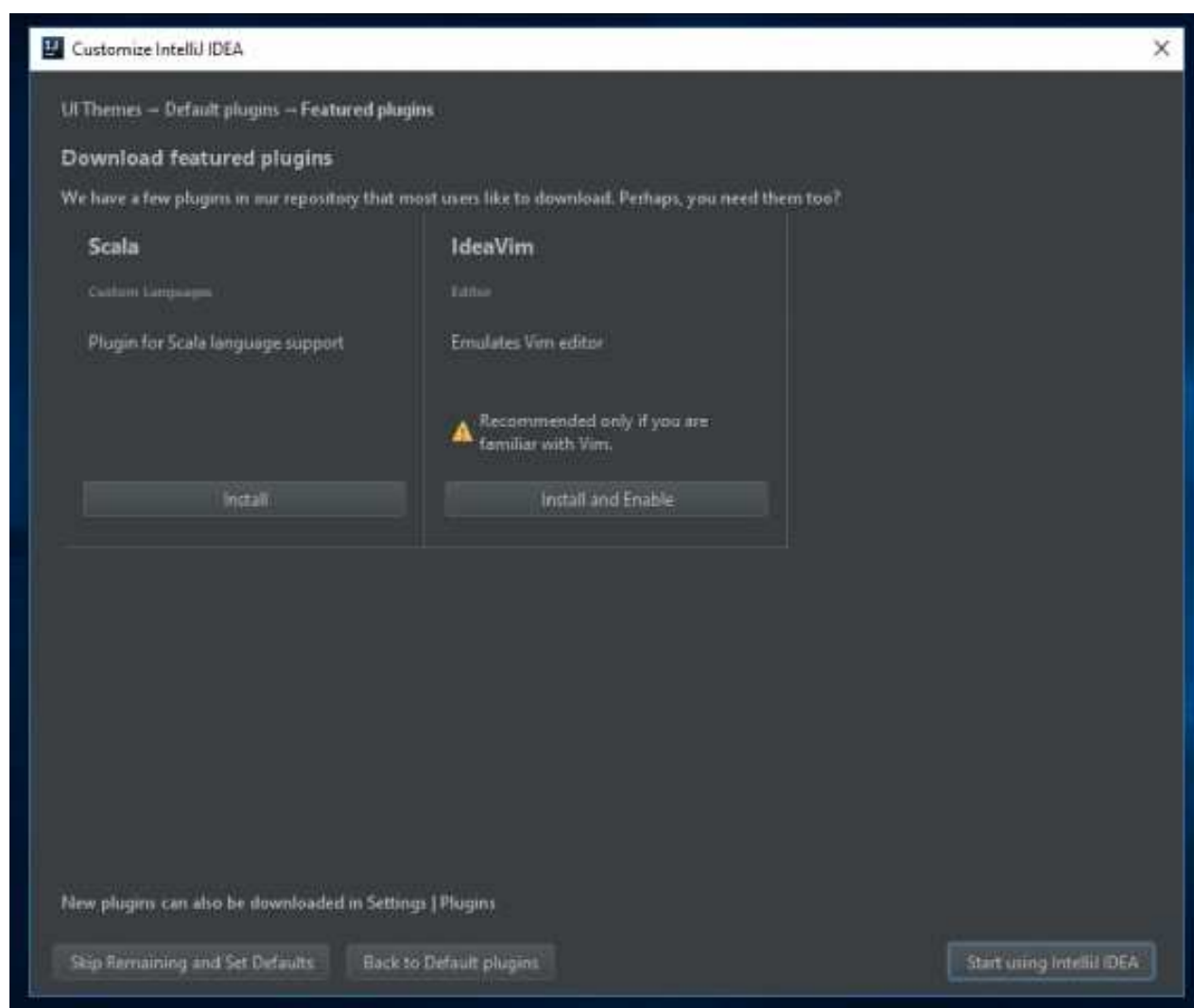


Figura 23. Instalação do IntelliJ (Parte 13 de 13)

2.2. MacOS

2.2.1. Instalação do Java



Baixar o Java JDK em:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

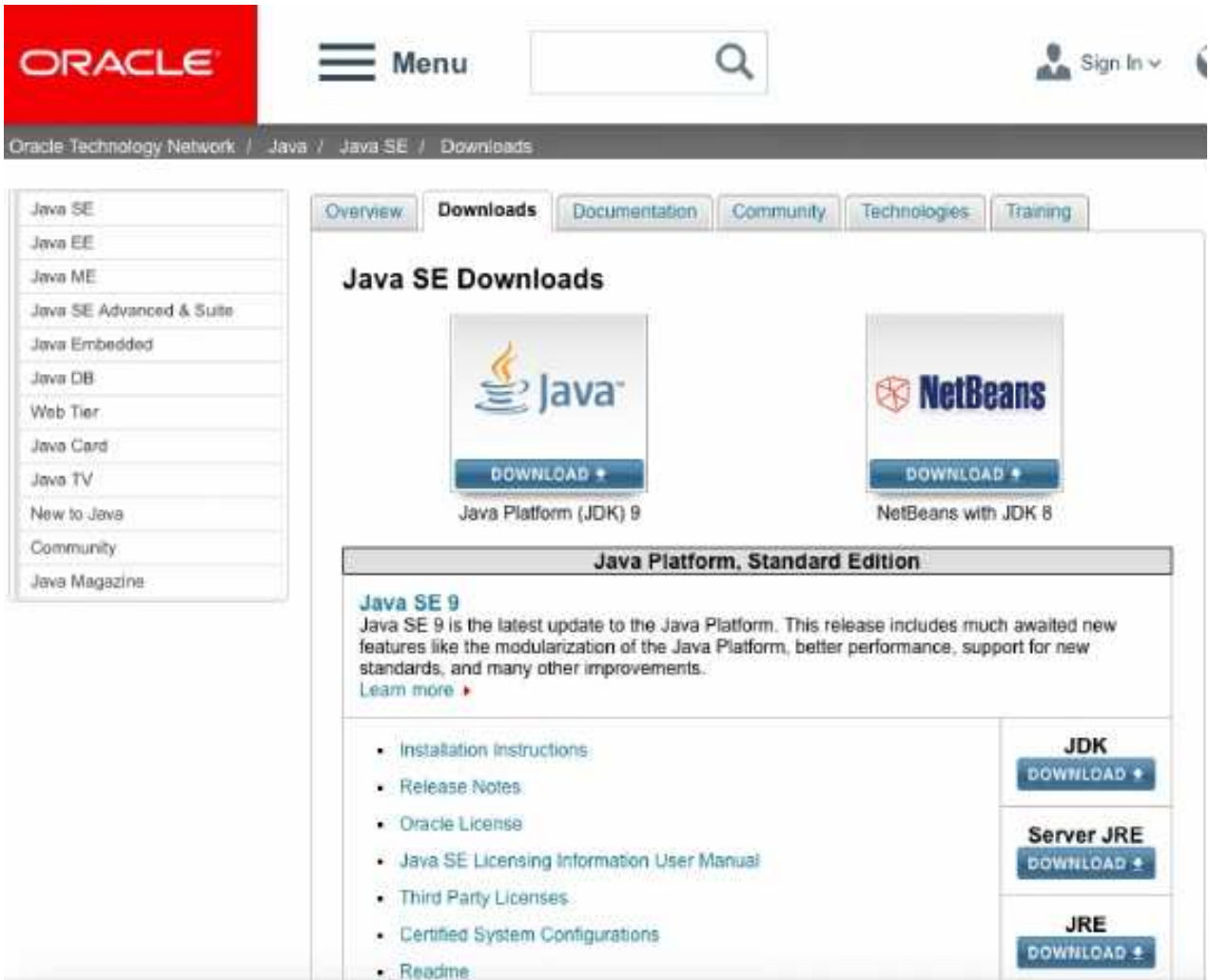


Figura 24. Instalação do Java (Parte 1 de 8)

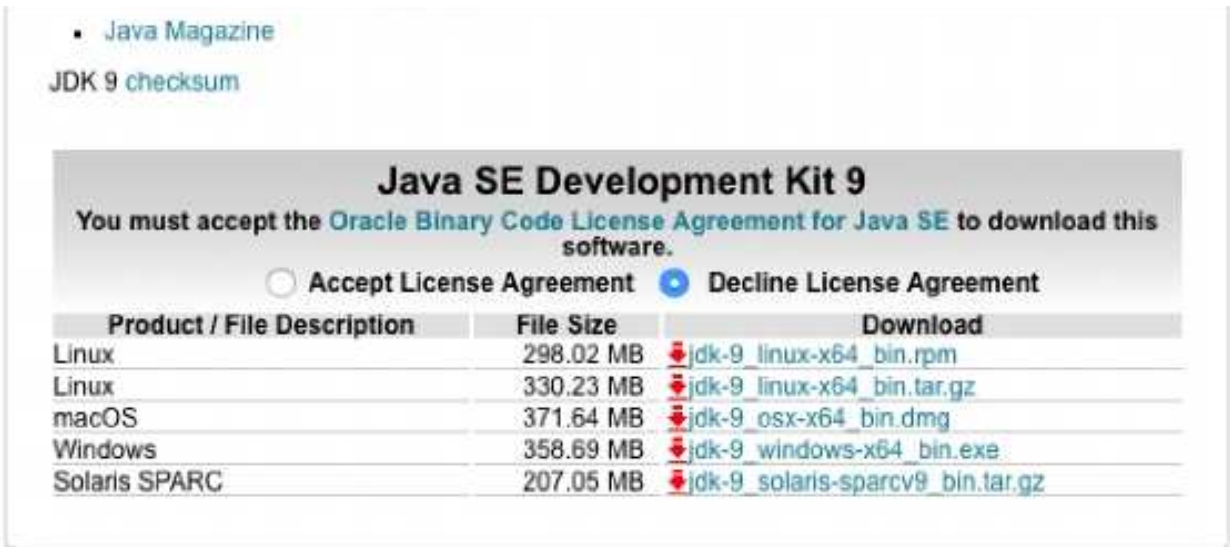


Figura 25. Instalação do Java (Parte 2 de 8)

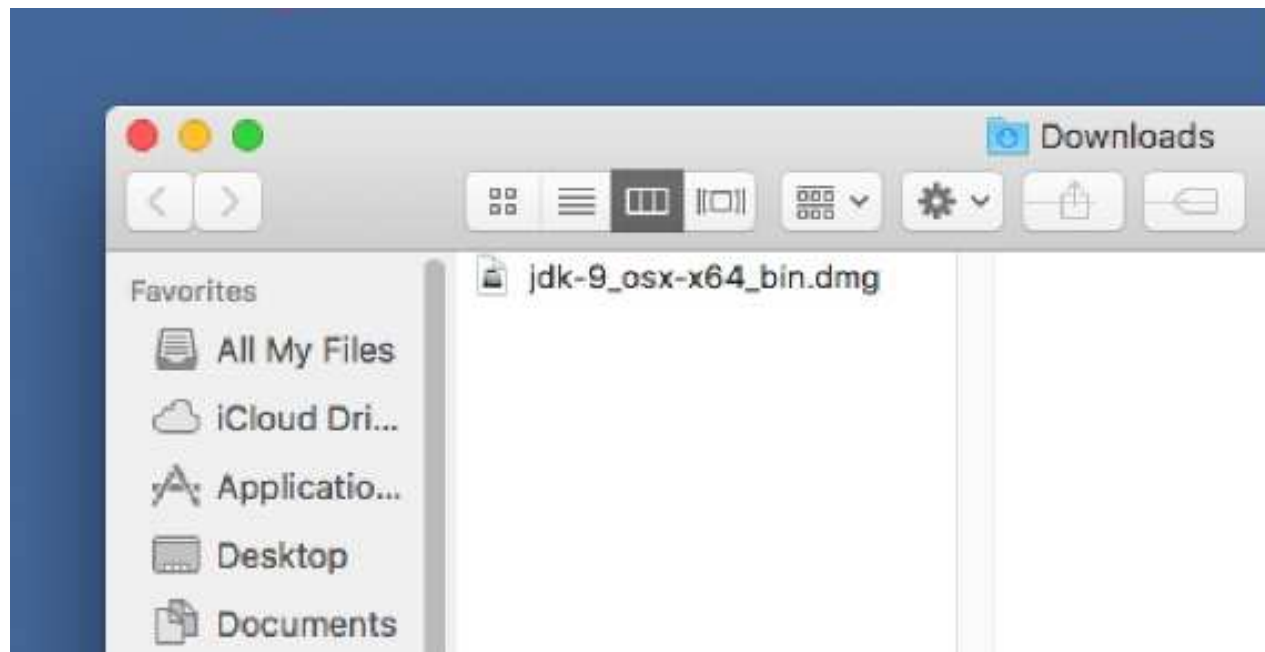


Figura 26. Instalação do Java (Parte 3 de 8)



Figura 27. Instalação do Java (Parte 4 de 8)



Figura 28. Instalação do Java (Parte 5 de 8)



Figura 29. Instalação do Java (Parte 6 de 8)



Figura 30. Instalação do Java (Parte 7 de 8)



Figura 31. Instalação do Java (Parte 8 de 8)

2.2.2. Instalação IntelliJ IDEA



Baixar o IntelliJ IDEA em:

<https://www.jetbrains.com/idea/download/#section=mac>



Figura 32. Instalação do IntelliJ (Parte 1 de 12)

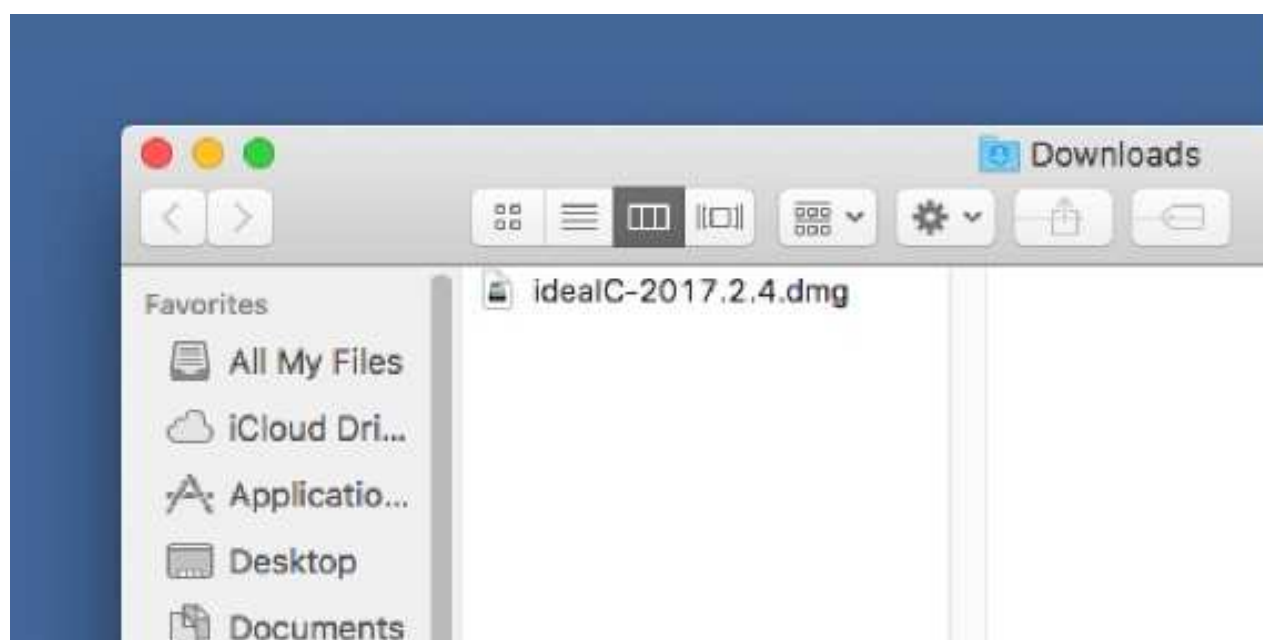


Figura 33. Instalação do IntelliJ (Parte 2 de 12)

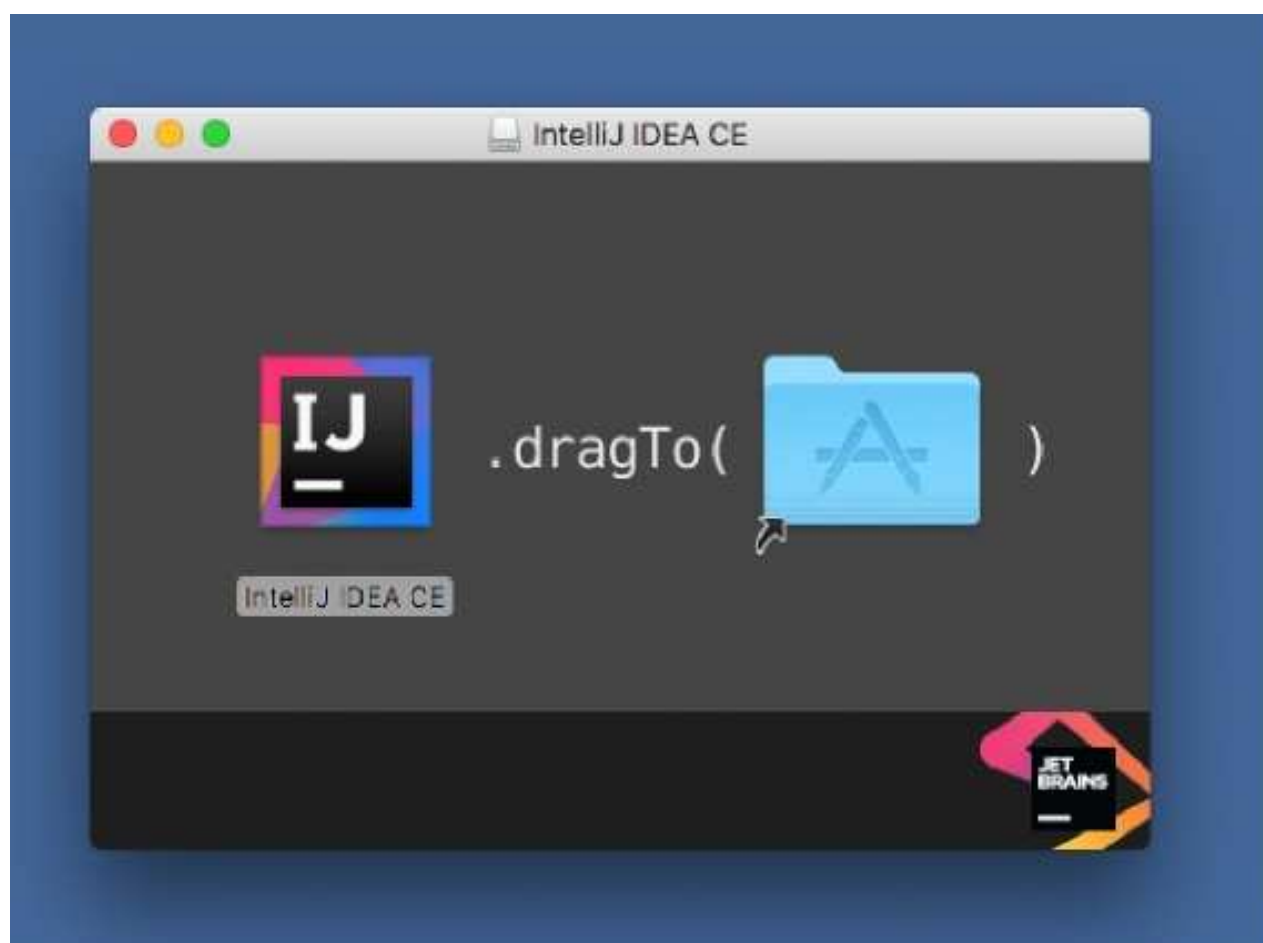


Figura 34. Instalação do IntelliJ (Parte 3 de 12)

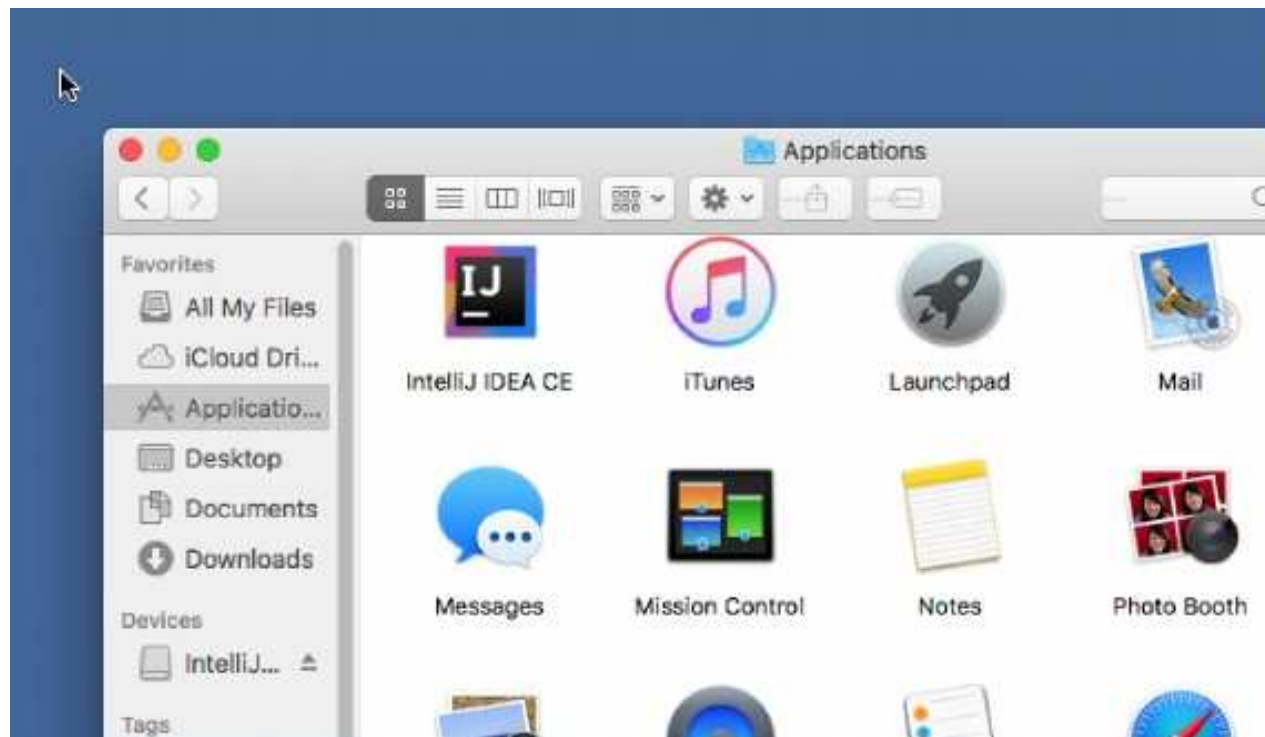


Figura 35. Instalação do IntelliJ (Parte 4 de 12)



Figura 36. Instalação do IntelliJ (Parte 5 de 12)

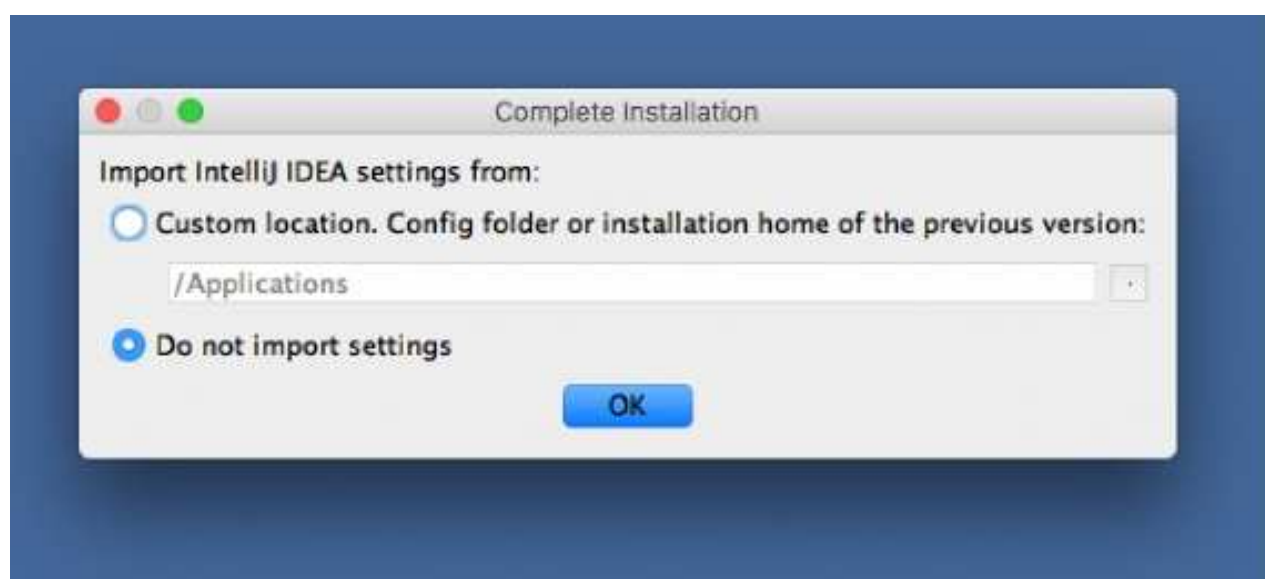


Figura 37. Instalação do IntelliJ (Parte 6 de 12)

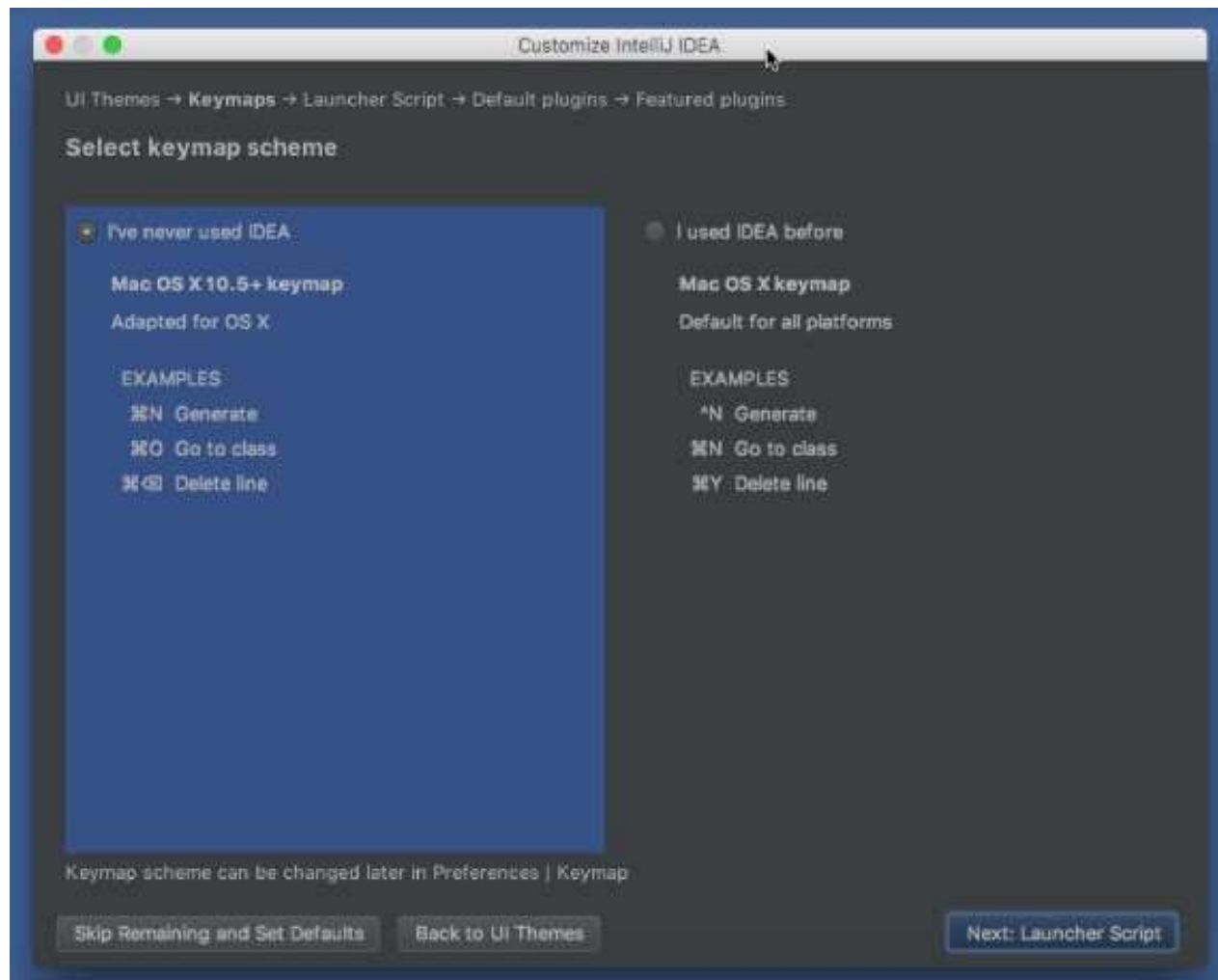


Figura 40. Instalação do IntelliJ (Parte 9 de 12)

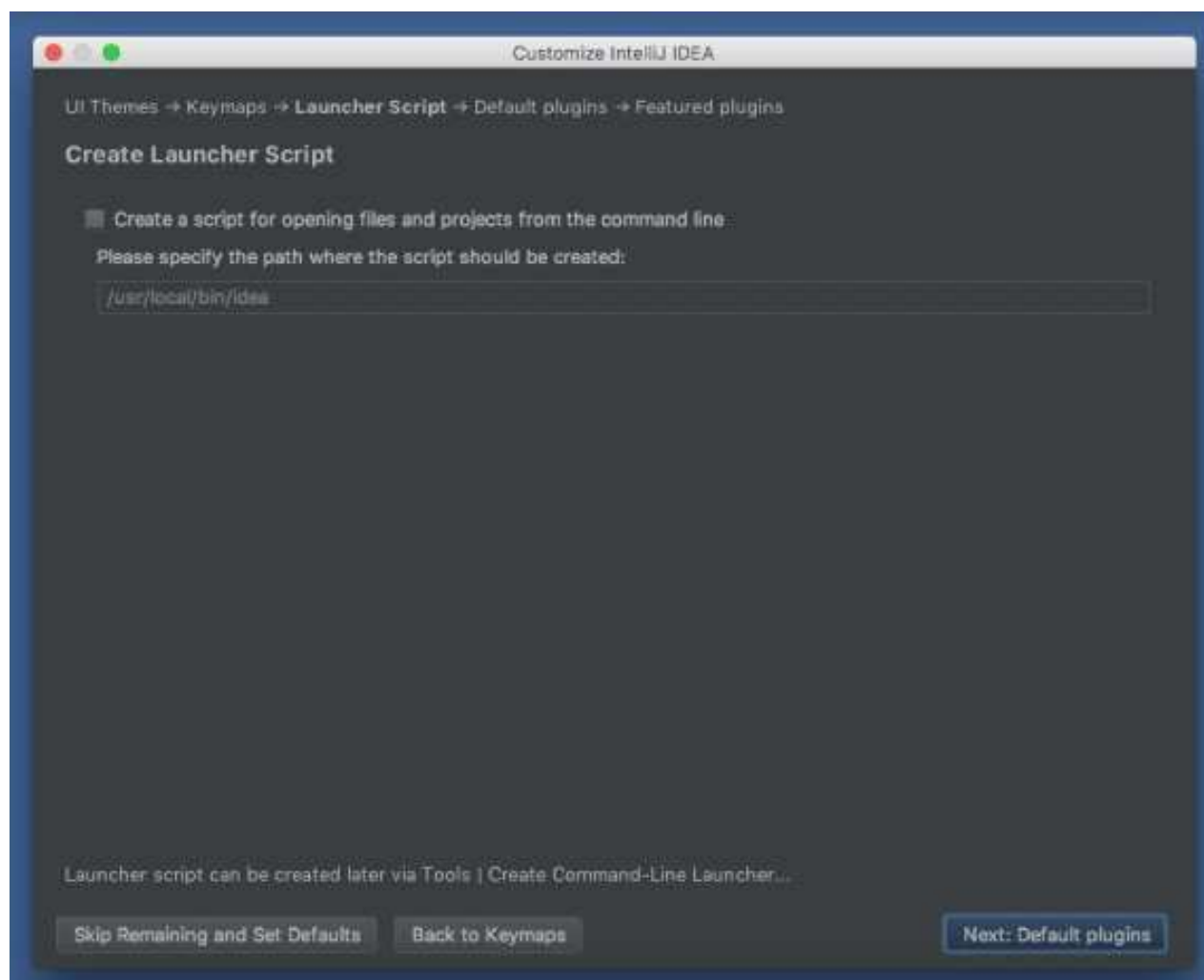


Figura 41. Instalação do IntelliJ (Parte 10 de 12)

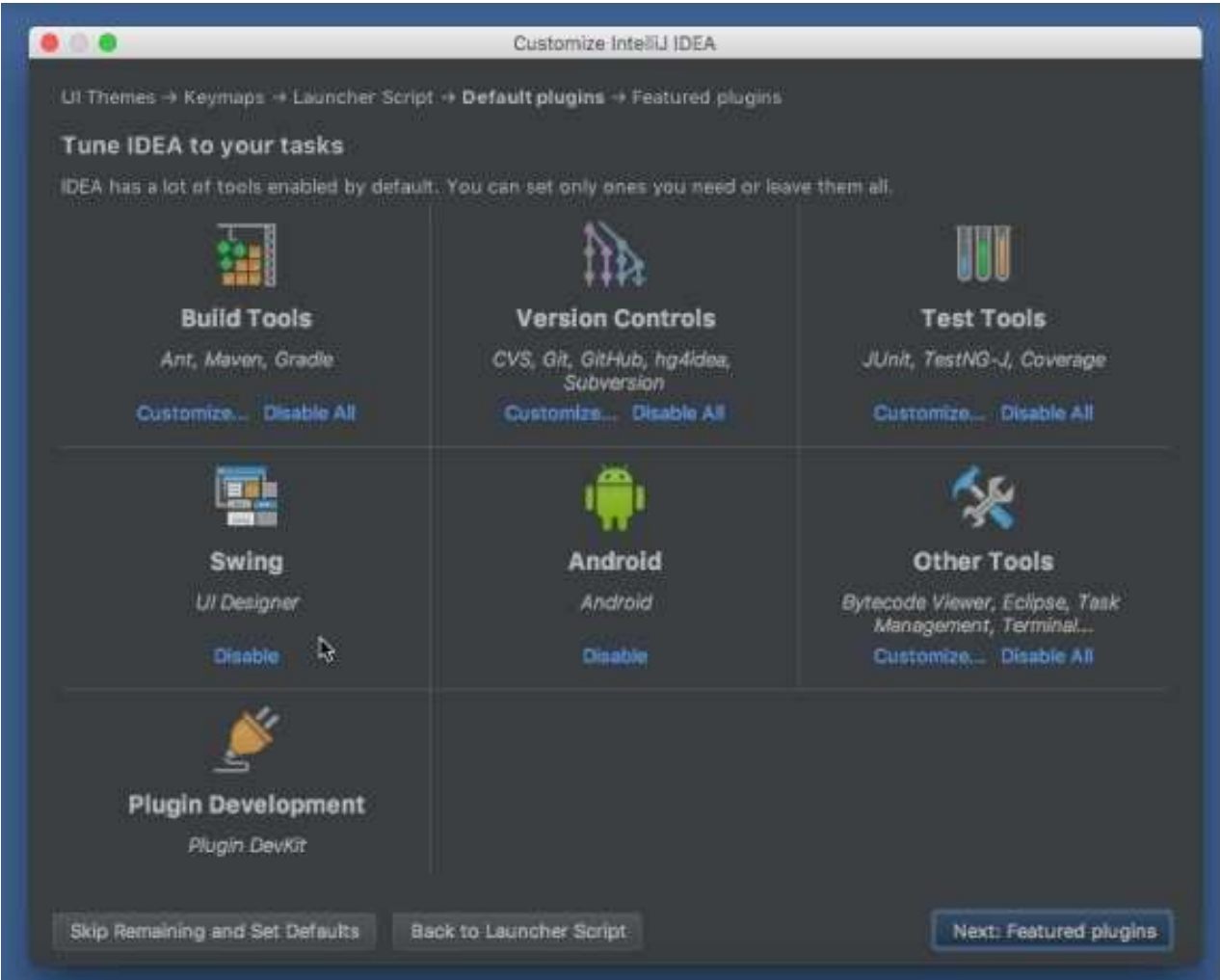


Figura 42. Instalação do IntelliJ (Parte 11 de 12)

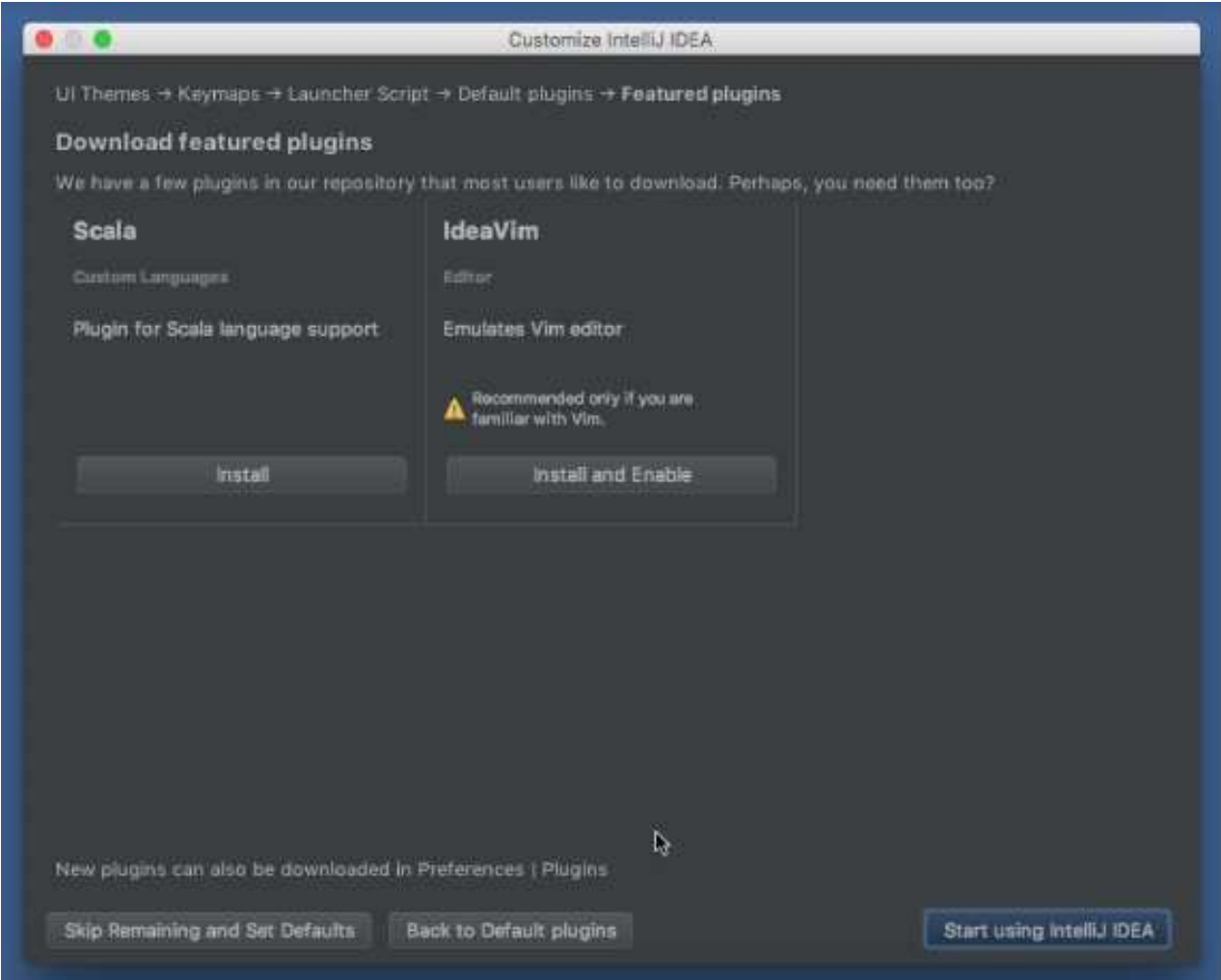


Figura 43. Instalação do IntelliJ (Parte 12 de 12)

2.3. Linux

2.3.1. Instalação do Java

Listagem 1 - Adicionando o repositório do JDK

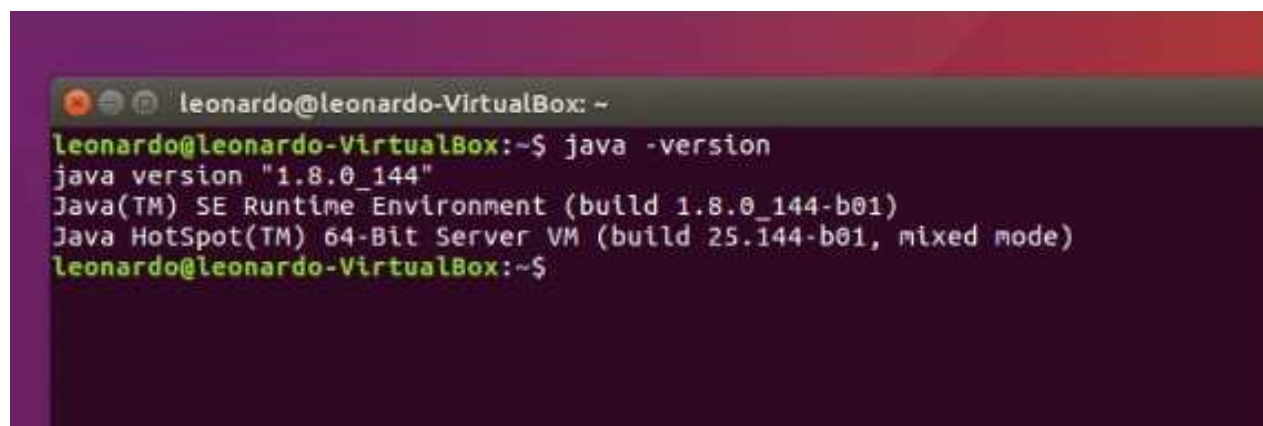
```
sudo add-apt-repository ppa:webupd8team/java
```

Listagem 2 - Atualizando repositório

```
sudo apt-get update
```

Listagem 3 - Instalando o JDK 8

```
sudo apt-get install oracle-java8-installer
```

A terminal window titled 'leonardo@leonardo-VirtualBox: ~' showing the command 'java -version' and its output. The output indicates that Java version '1.8.0_144' is installed, specifically the Java(TM) SE Runtime Environment (build 1.8.0_144-b01) and Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode).

```
leonardo@leonardo-VirtualBox: ~  
leonardo@leonardo-VirtualBox:~$ java -version  
java version "1.8.0_144"  
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)  
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)  
leonardo@leonardo-VirtualBox:~$
```

Figura 44. Instalação do Java

2.3.2. Instalação IntelliJ IDEA



Baixar o IntelliJ IDEA em:

<https://www.jetbrains.com/idea/download/#section=linux>



Figura 45. Instalação do IntelliJ (Parte 1 de 11)

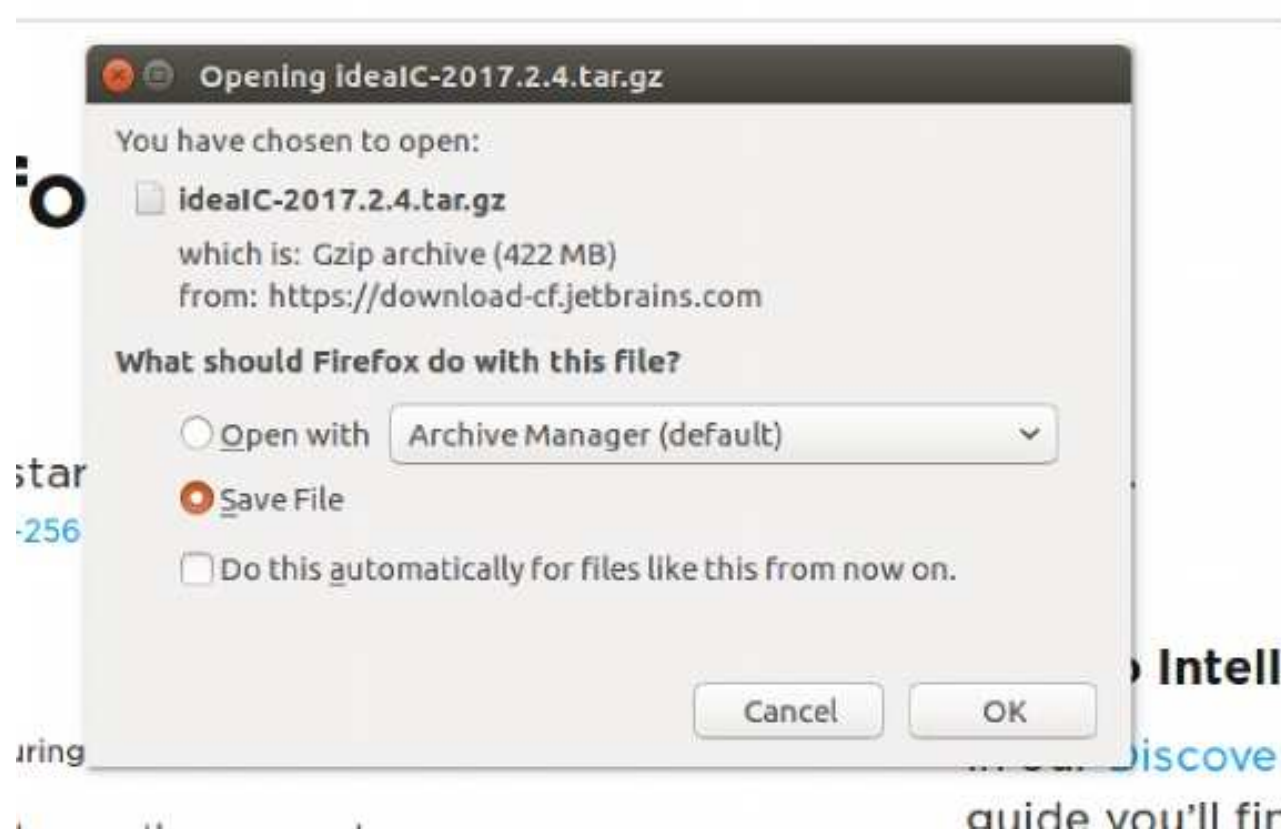


Figura 46. Instalação do IntelliJ (Parte 2 de 11)

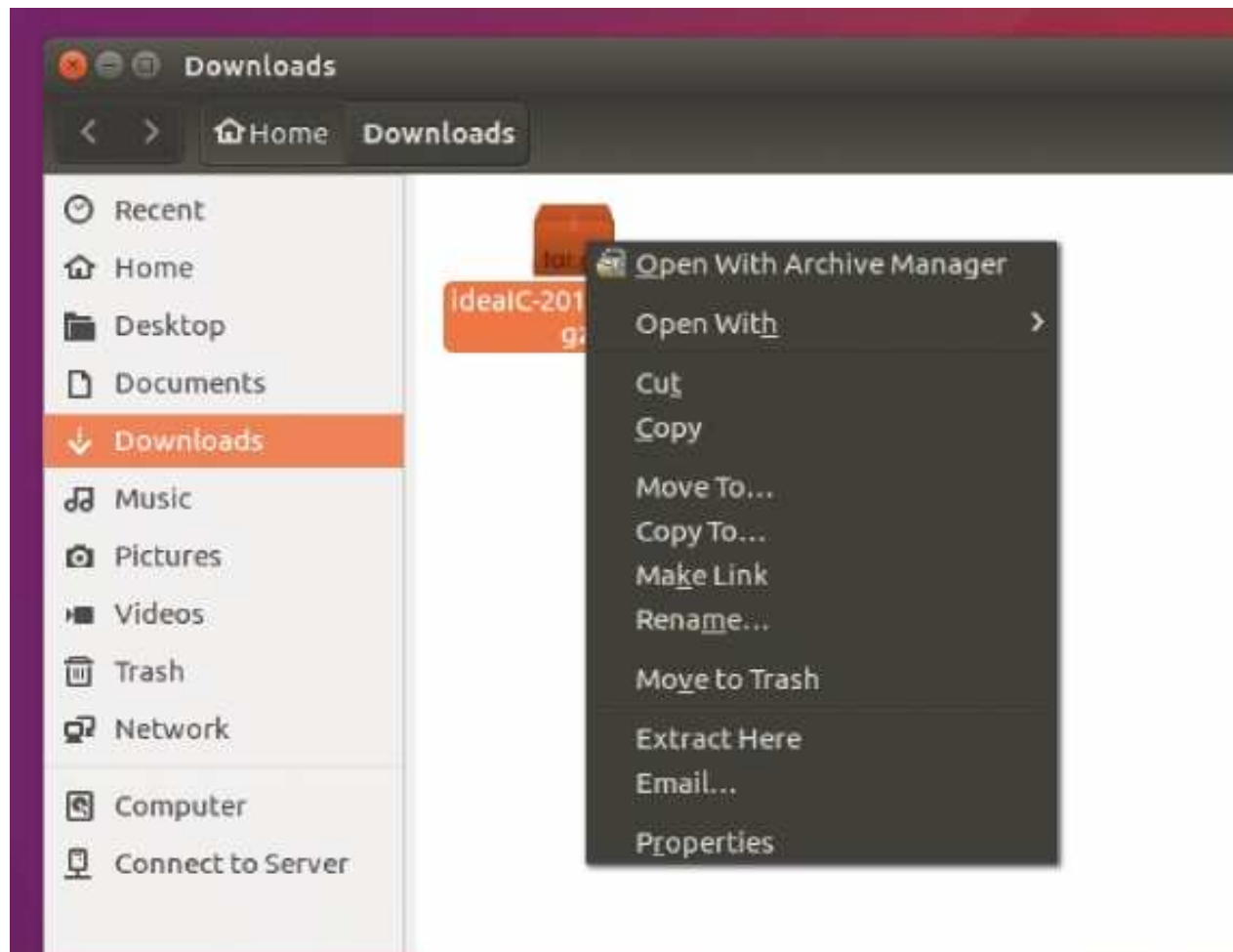


Figura 47. Instalação do IntelliJ (Parte 3 de 11)

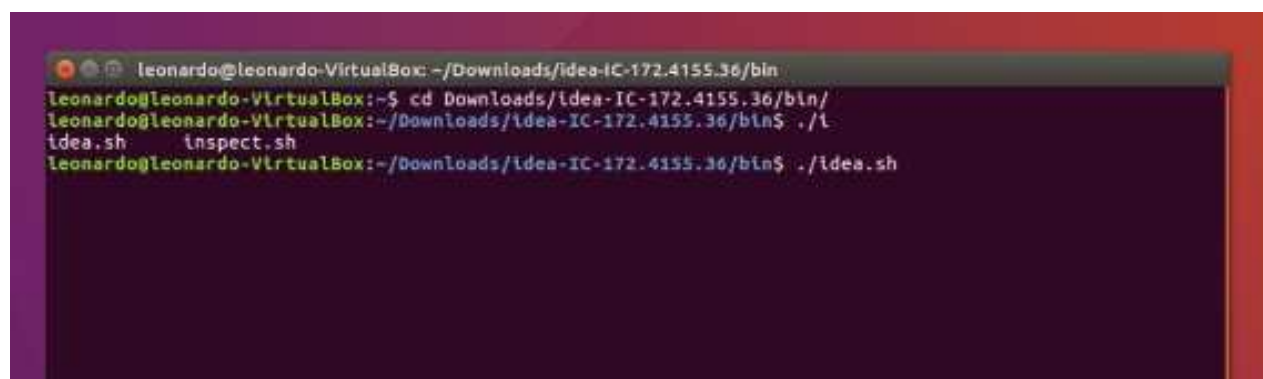


Figura 48. Instalação do IntelliJ (Parte 4 de 11)

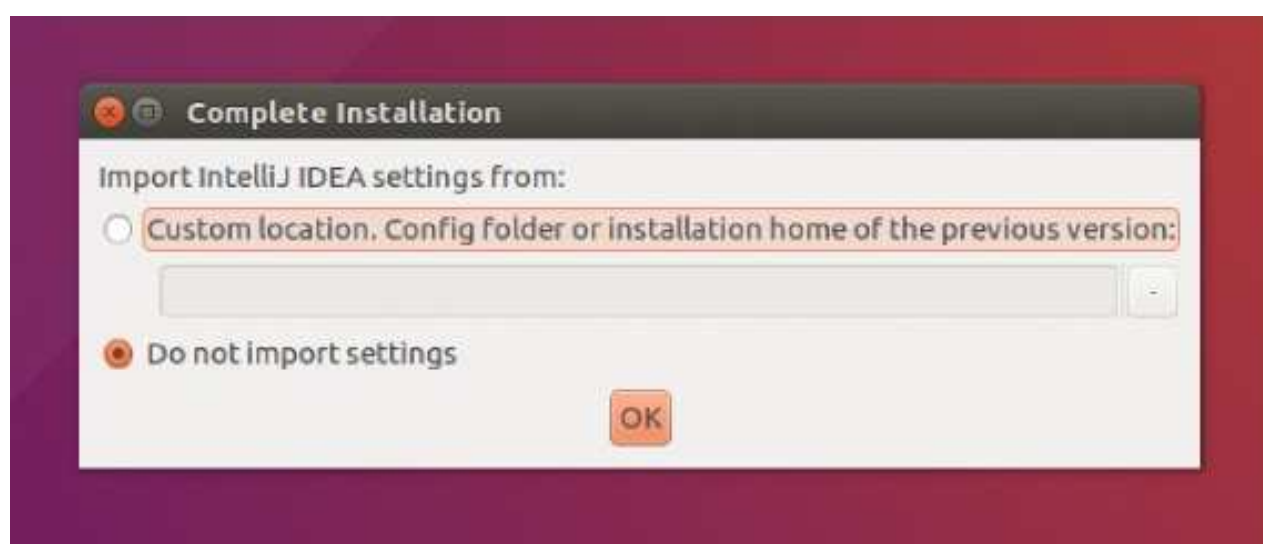


Figura 49. Instalação do IntelliJ (Parte 5 de 11)



Figura 50. Instalação do IntelliJ (Parte 6 de 11)

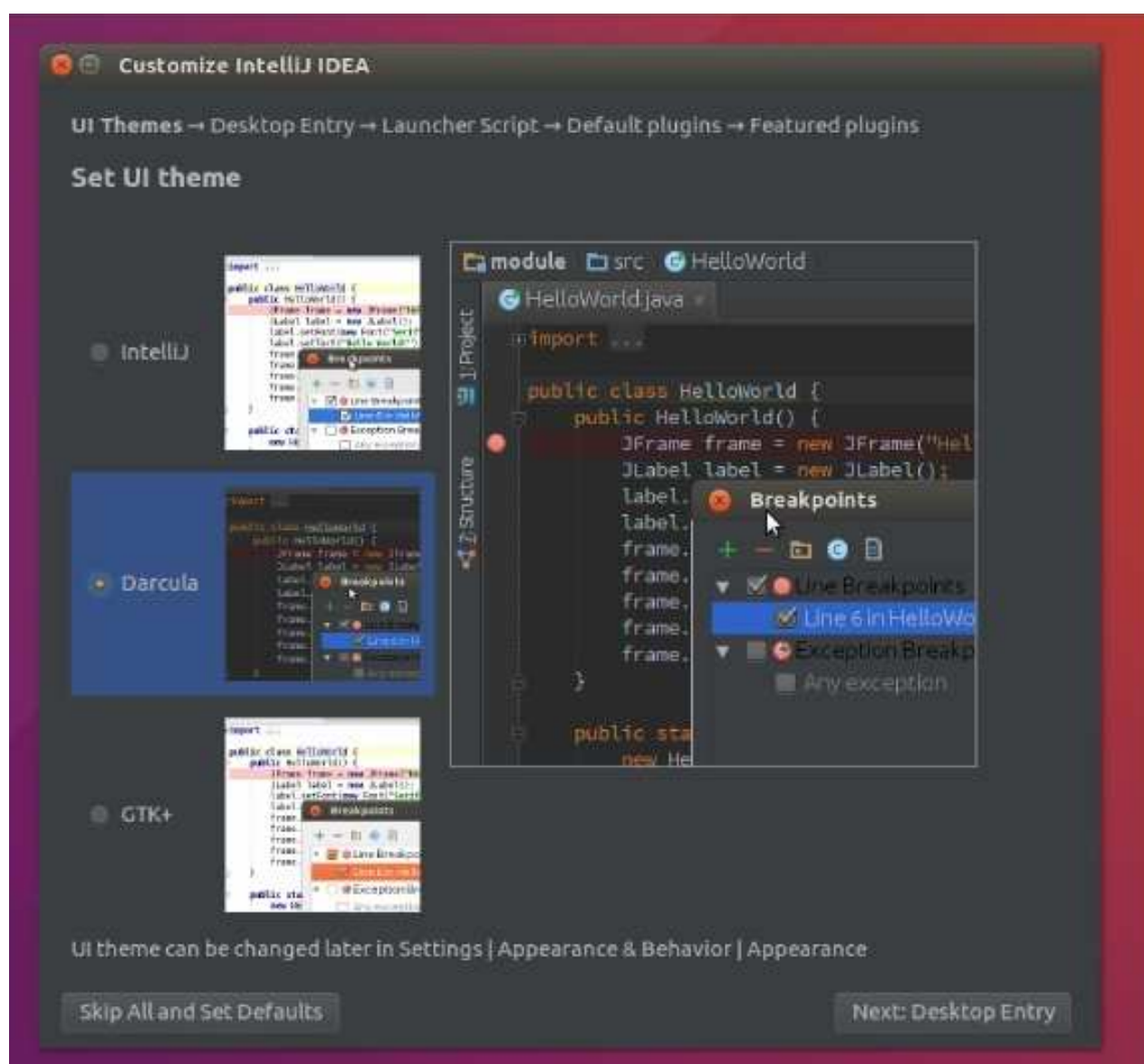


Figura 51. Instalação do IntelliJ (Parte 7 de 11)

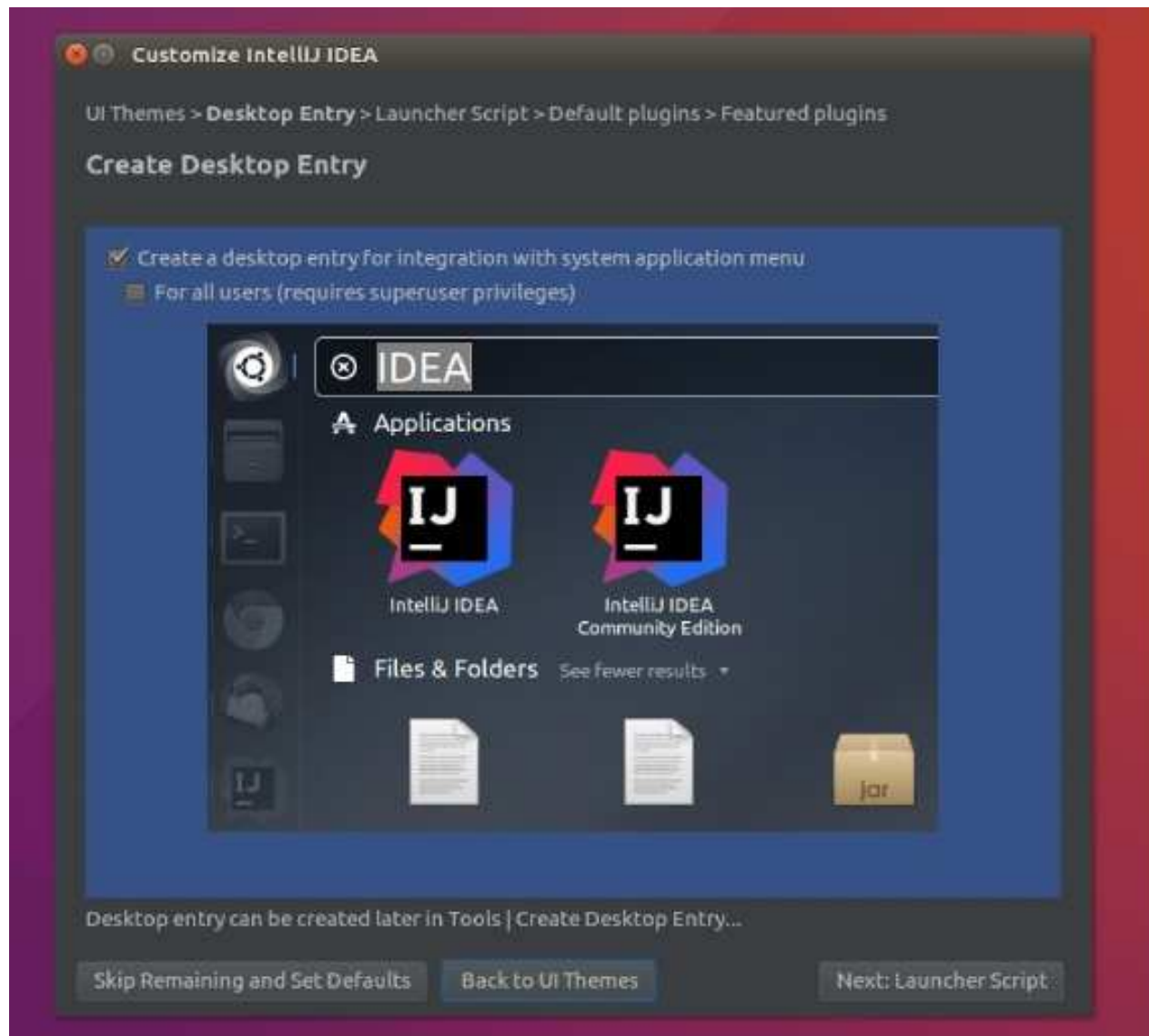


Figura 52. Instalação do IntelliJ (Parte 8 de 11)

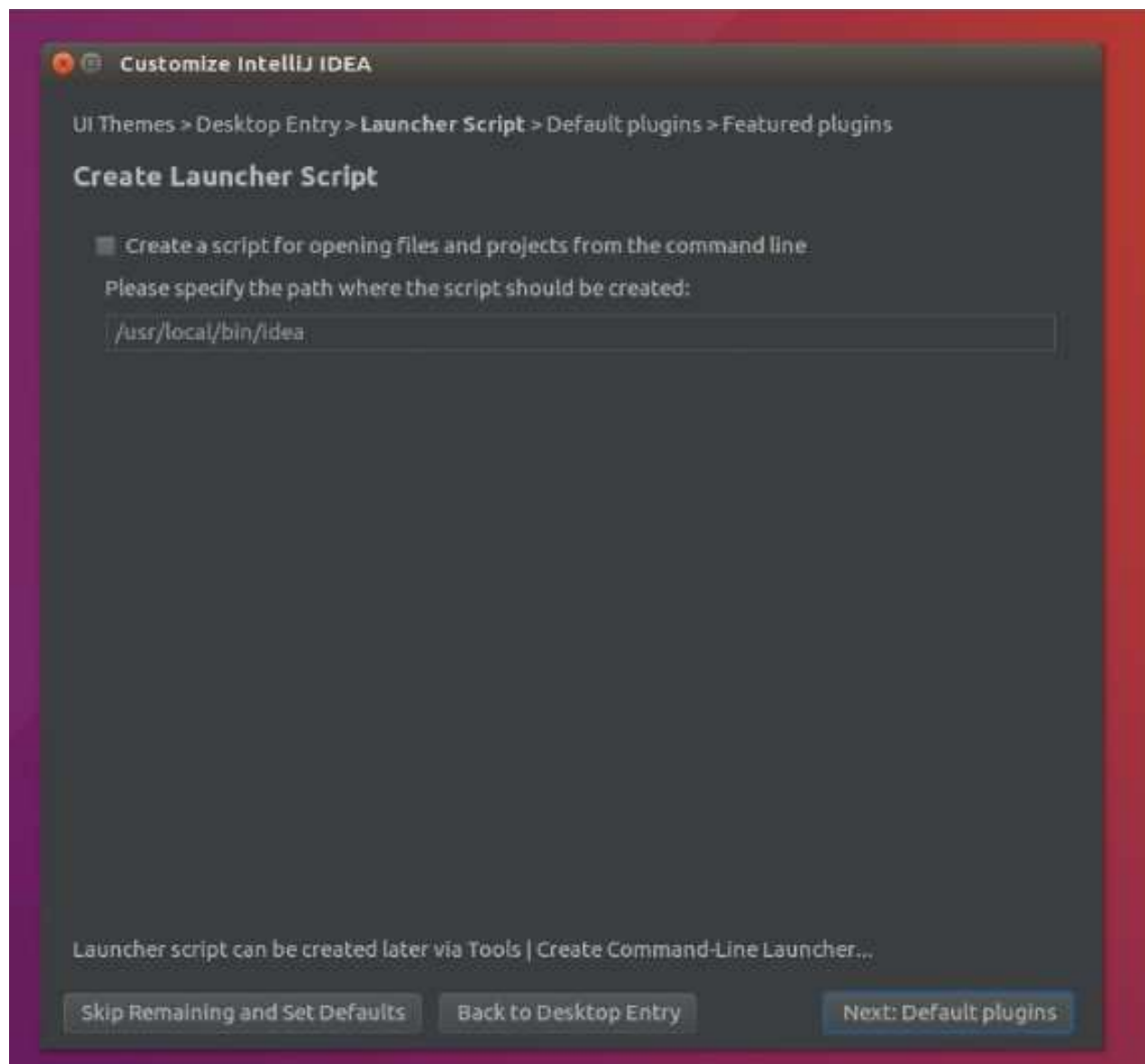


Figura 53. Instalação do IntelliJ (Parte 9 de 11)

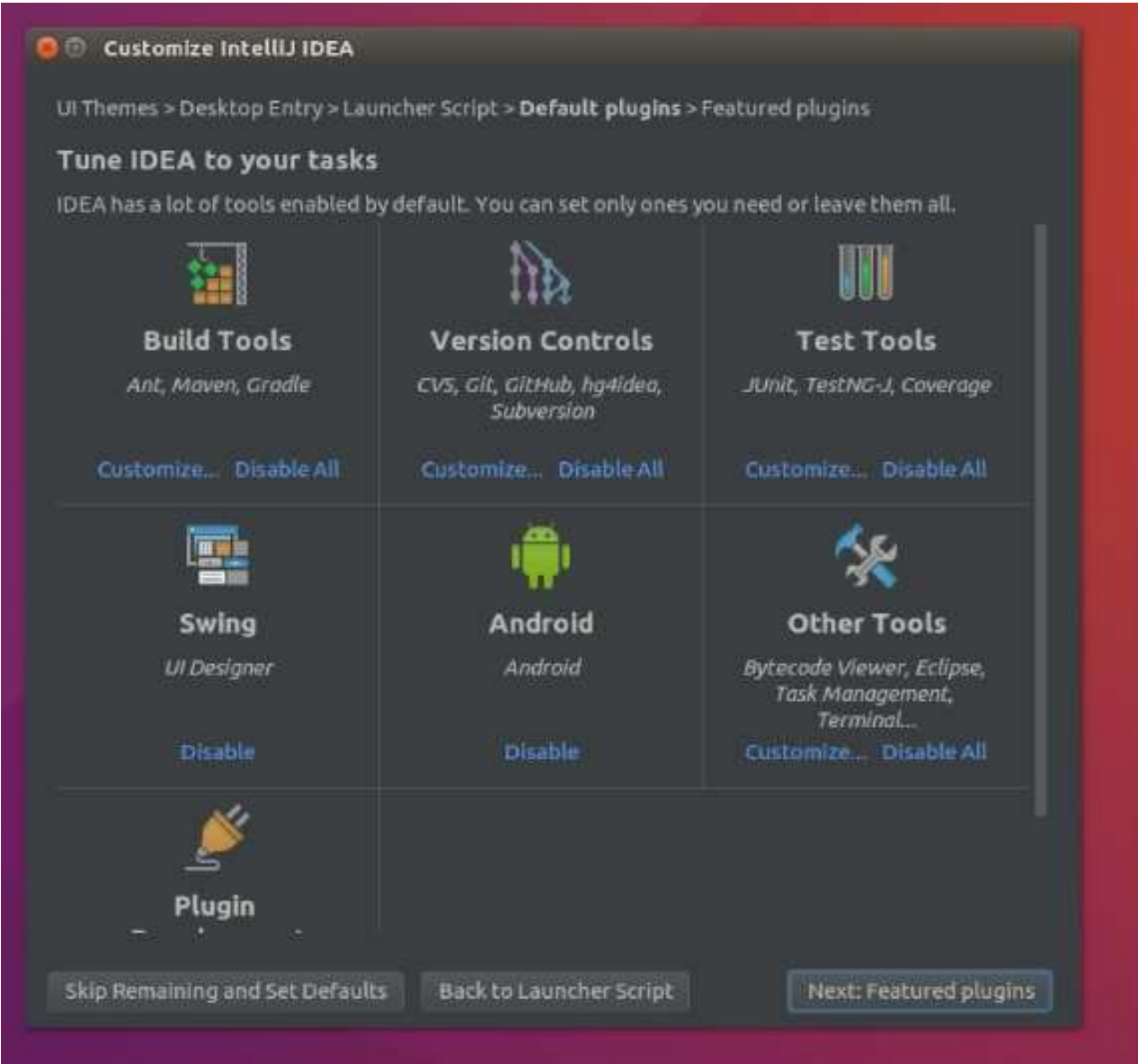


Figura 54. Instalação do IntelliJ (Parte 10 de 11)

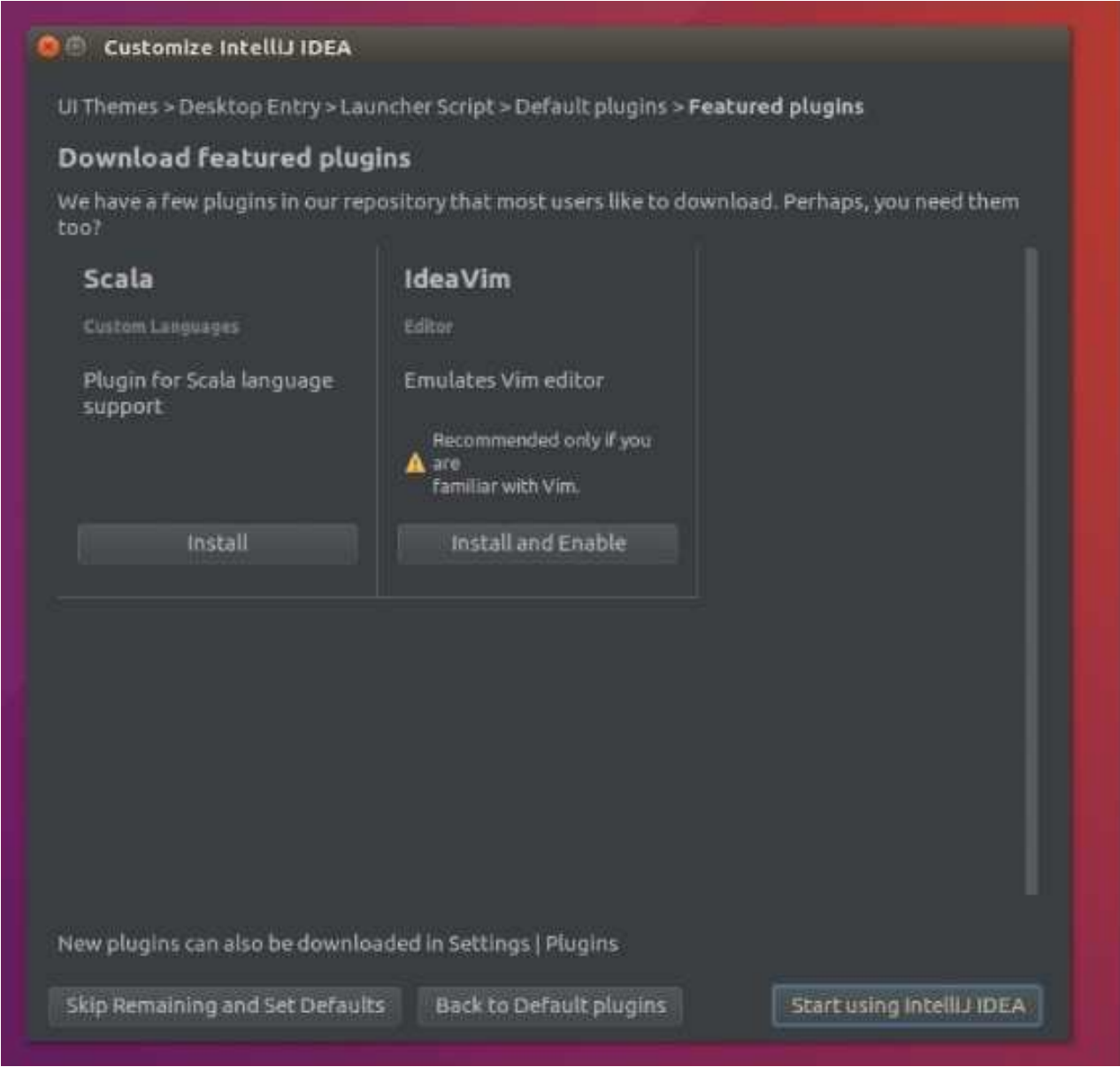


Figura 55. Instalação do IntelliJ (Parte 11 de 11)

3. Conceitos Básicos

3.1. Primeiro Programa em Kotlin



Apesar de ser um exercício bastante simples é possível identificar alguns padrões importantes da linguagem:

- os blocos de código são delimitados por "{}"
- as sentenças de código podem terminar ou não com ";"

Listagem 4 - Primeiro Programa em Kotlin

fundamentos/PrimeiroPrograma.kt

```
package fundamentos

fun main(args: Array<String>) {
    print("Primeiro")
    println(" programa!");
}
```

3.2. Comentários



A ferramenta de geração da documentação feita em código Kotlin é chamada de **Dokka**. <https://github.com/Kotlin/dokka>

Listagem 5 - Comentários em Kotlin

fundamentos/Comentarios.kt

```
package fundamentos

/**
 * Só para dizer que o Kotlin suporta o KDoc... :)
 *
 * @param args lista de parâmetros passados por linha de comando
 */
fun main(args: Array<String>) {

    // Comentário de uma linha
    println("Opa")

    /*
     Mais de
     um linha
    */
    println("Sério?")

    /*
     * Mais de
     * um linha
     * com estilo
     */
    println("Legal!")
}
```

3.3. Variáveis e Constantes

Existem duas palavras reservadas para declarar variáveis:

- **val** - referência **imutável** (constante)
- **var** - referência **mutável** (variável)

3.3.1. Variáveis (var)



Kotlin é uma linguagem **fortemente tipada**, mas é possível não especificar o tipo quando uma atribuição for feita na inicialização.

Nesse cenário dizemos que o tipo foi **inferido**.

Listagem 6 - Variáveis em Kotlin

fundamentos/Var1.kt

```
package fundamentos

fun main(args: Array<String>) {
    var a: Int
    var b = 2 // Tipo inferido

    a = 10

    print(a + b)
}
```

3.3.2. Cuidado com as inferências



O código do próximo exemplo gera um erro:

Kotlin: This variable must either have a type annotation or be initialized

Listagem 7 - Cuidado com as inferências (Parte 1)

fundamentos/CuidadoInferencia1.kt

```
package fundamentos

fun main(args: Array<String>) {
    var a // Erro!
    var b = 2

    a = 10

    print(a + b)
}
```



O código do próximo exemplo gera um erro:

Kotlin: The floating-point literal does not conform to the expected type Int

Listagem 8 - Cuidado com as inferências (Parte 2)

fundamentos/CuidadoInferencia2.kt

```
package fundamentos

fun main(args: Array<String>) {
    var a = 1
    var b = 2

    a = 2.3 // Erro!

    print(a + b)
}
```

3.3.3. Constantes (val)



O código do próximo exemplo gera um erro:

Kotlin: Val cannot be reassigned

Listagem 9 - Constantes em Kotlin

fundamentos/Const.kt

```
package fundamentos

fun main(args: Array<String>) {
    val a: Int = 1
    val b = 2 // Tipo inferido

    a = a + b // Erro!
    print(a)
}
```

3.3.4. Constantes Java

Listagem 10 - Constantes vindas do Java

fundamentos/ConstJava.kt

```
package fundamentos

fun main(args: Array<String>) {
    val raio = 4.5
    print(raio * raio * Math.PI)
}
```

3.4. Tipos Básicos



Tudo em Kotlin é Objeto, inclusive os tipos básicos!

3.4.1. Numéricos

Tabela 1. Tipos Numéricos

Tipos	Tamanho (bits)
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

Listagem 11 - Tipos Básicos

fundamentos/TiposBasicos.kt

```
package fundamentos

fun main(args: Array<String>) {
    // Tipos Numéricos Inteiros
    val num1: Byte = 127
    val num2: Short = 32767
    val num3: Int = 2_147_483_647
    val num4: Long = 9_223_372_036_854_775_807 // Long.MAX_VALUE

    // Tipos Numéricos Reais
    val num5: Float = 3.14F
    val num6: Double = 3.14

    // Tipo Caractere
    val char: Char = '?' // Outros exemplos... '1', 'g', ' '

    // Tipo boolean
    val boolean: Boolean = true // ou false

    println(listOf(num1, num2, num3, num4, num5, num6, char, boolean))

    println(2 is Int)
    println(2147483648 is Long)
    println(1.0 is Double)

    // Tudo é objeto
    println(10.dec())
}
```

3.5. Função



Haverá um **capítulo** exclusivamente dedicado para **função**, então o objetivo dessa seção é mostrar o básico sobre o assunto.



Figura 56. Função Com Parâmetros e Com Retorno



Figura 57. Função **Sem** Parâmetros e **Com** Retorno



Figura 58. Função **Com** Parâmetros e **Sem** Retorno



Figura 59. Função **Sem** Parâmetros e **Sem** Retorno

Listagem 12 - Função Sem Retorno (Mais ou Menos)

fundamentos/FuncaoSemRetorno.kt

```
package fundamentos

fun main(args: Array<String>) {
    imprimirSoma(4, 5)
}

fun imprimirSoma(a: Int, b: Int) {
    println(a + b)
}
```


Listagem 13 - Função Com Retorno

fundamentos/FuncaoComRetorno.kt

```
package fundamentos

fun main(args: Array<String>) {
    println(soma(2, 3))
    println(soma(11))
}

fun soma(a: Int, b: Int = 1): Int {
    return a + b
}
```

3.6. Template Strings

Listagem 14 - Usando Template String

fundamentos/TemplateString1.kt

```
package fundamentos

fun main(args: Array<String>) {
    val aprovados = listOf("João", "Maria", "Pedro")
    print("O primeiro colocado foi ${aprovados[0]}")
}
```

Listagem 15 - Usando Template String

fundamentos/TemplateString2.kt

```
package fundamentos

fun main(args: Array<String>) {
    val bomHumor = true
    print("Hoje estou ${if (bomHumor) "feliz" else "chateado"}")
}
```

3.7. Notação Ponto

Listagem 16 - Notação Ponto

fundamentos/NotacaoPonto.kt

```
package fundamentos

fun main(args: Array<String>) {

    var a : Int = 33.dec()
    var b : String = a.toString()

    println("Int: $a")
    println("Primeiro char da string b é: ${b.first()}")

}
```

3.8. Null Safety

3.8.1. Operador Chamada Segura (Safe Call Operator)

Listagem 17 - Tipos Básicos são Objetos

fundamentos/ChamadaSegura.kt

```
package fundamentos

fun main(args: Array<String>) {
    print(10.dec())

    var a : Int? = null // safe call operator
    println(a?.dec())
}
```

3.8.2. Operador Elvis



Curiosidade:

Esse operador tem esse nome porque o seu símbolo parece com o emoticon que representa o cantor Elvis Presley **?:j**

Listagem 18 - Operador Elvis

fundamentos/OperadorElvis.kt

```
package fundamentos

fun main(args: Array<String>) {
    val opcional : String? = null
    val obrigatorio : String = opcional ?: "Valor Padrão"

    println(obrigatorio)
}
```

3.8.3. Forçando NullPointerException



O código do próximo exemplo gera um erro:

Exception in thread "main" kotlin.KotlinNullPointerException

Listagem 19 - Forçando Exceção de NullPointerException

fundamentos/ForcandoExcecaoNullPointerException.kt

```
package fundamentos

fun main(args: Array<String>) {
    var a: Int? = null
    println(a?.inc())

    println("Hora do erro...")
    println(a!!.inc())
}
```

3.9. Import

Listagem 20 - Exemplos de Import

fundamentos/pacoteA/Mistura.kt

```
package fundamentos.pacoteA

fun simplesFuncao(texto: String): String {
    return "Texto = $texto"
}

class Coisa(val nome: String)

enum class FaceMoeda { CARA, COROA }
```

fundamentos/pacoteB/Matematica.kt

```
package fundamentos.pacoteB

fun soma(a: Int, b: Int): Int {
    return a + b
}

fun subtracao(a: Int, b: Int): Int {
    return a - b
}
```

fundamentos/TesteImport.kt

```
package fundamentos

import fundamentos.pacoteA.simplesFuncao as funcaoSimples
import fundamentos.pacoteA.Coisa
import fundamentos.pacoteA.FaceMoeda.CARA
import fundamentos.pacoteB.*

fun main(args: Array<String>) {
    kotlin.io.println(funcaoSimples("Ok"))

    val coisa = Coisa("Bola")
    println(coisa.nome)

    println(CARA)

    println("${soma(2, 3)} ${subtracao(4, 6)}")
}
```




Lista de pacotes importados por padrão:

- kotlin.*
- kotlin.annotation.*
- kotlin.collections.*
- kotlin.comparisons.* (desde 1.1)
- kotlin.io.*
- kotlin.ranges.*
- kotlin.sequences.*
- kotlin.text.*

Quando executado na **JVM** são adicionados também:

- java.lang.*
- kotlin.jvm.*

3.10. Estruturas de Controle

3.10.1. If

Listagem 21 - Utilizando a estrutura If

fundamentos/controles/If.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    val nota : Double = 8.3

    if(nota >= 7.0) {
        println("Aprovado")
    }
}
```

3.10.2. If/Else

Listagem 22 - Utilizando a estrutura If/Else

fundamentos/controles/IfElse.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    val nota : Double = 8.3

    if (nota >= 7.0) {
        println("Aprovado!!")
    } else {
        println("Reprovado!!")
    }
}
```

Listagem 23 - Variáveis com If/Else(Validar)

fundamentos/controles/IfElse2.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    val num1: Int = 2
    val num2: Int = 3

    val maiorValor = if (num1 > num2) {
        println("processando...")
        num1
    } else {
        println("processando...")
        num2
    }

    println("O maior valor é $maiorValor")
}
```

3.10.3. If/Else If/Else

Listagem 24 - *If em cascata*

fundamentos/controles/IfElseIf.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    val nota: Double = 9.3

    // Usando operador range
    if (nota in 9..10) {
        println("Fantástico")
    } else if (nota in 7..8) {
        println("Parabéns")
    } else if (nota in 4..6) {
        println("Tem como recuperar")
    } else if (nota in 0..3) {
        println("Te vejo no próximo semestre")
    } else {
        println("Nota inválida")
    }
}
```



Use *when* ao invés de *if* em cascata

Listagem 25 - *Refatorando para When*

fundamentos/controles/RefatorandoParaWhen.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    val nota = 10

    when(nota) {
        10, 9 -> print("Fantástico")
        8, 7 -> print("Parabéns")
        6, 5, 4 -> print("Tem como recuperar")
        in 3..0 -> print("Te vejo próximo semestre")
        else -> print("Nota inválida")
    }
}
```

3.11. While

Listagem 26 - *While(Validar)*

fundamentos/controles/While1.kt

```
package fundamentos.controles

fun main(args: Array<String>) {

    var opcao: Int = 0

    while(opcao != -1) {
        val line = readLine() ?: "0"
        opcao = line.toIntOrNull() ?: 0

        println("Você escolheu a opção $opcao")
    }

    println("Até a próxima!")
}
```

Listagem 27 - *While*

fundamentos/controles/While2.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    var contador : Int = 1

    while(contador <= 10) {
        println(contador)
        contador++
    }
}
```

3.11.1. For

Listagem 28 - Itera sobre um Intervalo Fixo

fundamentos/controles/For1.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    for(i in 1..10) {
        println(i)
    }
}
```

Listagem 29 - Itera sobre um Intervalo Fixo (Decrescente)

fundamentos/controles/For2.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    for(i in 10 downTo 1) {
        println("i = $i")
    }
}
```

Listagem 30 - Itera sobre um Intervalo Fixo (Com Passo)

fundamentos/controles/For3.kt

```
package fundamentos.controles

fun main(args: Array<String>) {

    for(i in 0..100 step 5) {
        println(i)
    }

    for(i in 100 downTo 0 step 5) {
        println(i)
    }
}
```


Listagem 31 - Itera com Acesso ao Índice

fundamentos/controles/For4.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    val alunos = arrayListOf("André", "Carla", "Marcos")

    for ((indice, aluno) in alunos.withIndex()) {

        println("$indice - $aluno \n" )
    }
}
```

3.11.2. Do/While

Listagem 32 - Do/While

fundamentos/controles/DoWhile.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    var opcao: Int = 0

    do {
        val line = readLine() ?: "0"
        opcao = line.toIntOrNull() ?: 0
        println("Você escolheu a opção $opcao")
    } while (opcao != -1)

    println("Até a próxima!")
}
```

3.11.3. Break

Listagem 33 - Break

fundamentos/controles/Break1.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    for (i in 1..10){
        if(i == 5){
            break
        }
        println("Atual: $i")
    }
}
```

Listagem 34 - Break com Label

fundamentos/controles/Break2.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    loop@for (i in 1..15) {
        for (j in 1..15) {
            if (i == 2 && j == 9) break@loop
            println("$i $j")
        }
    }
    println("Fim!")
}
```

3.11.4. Continue

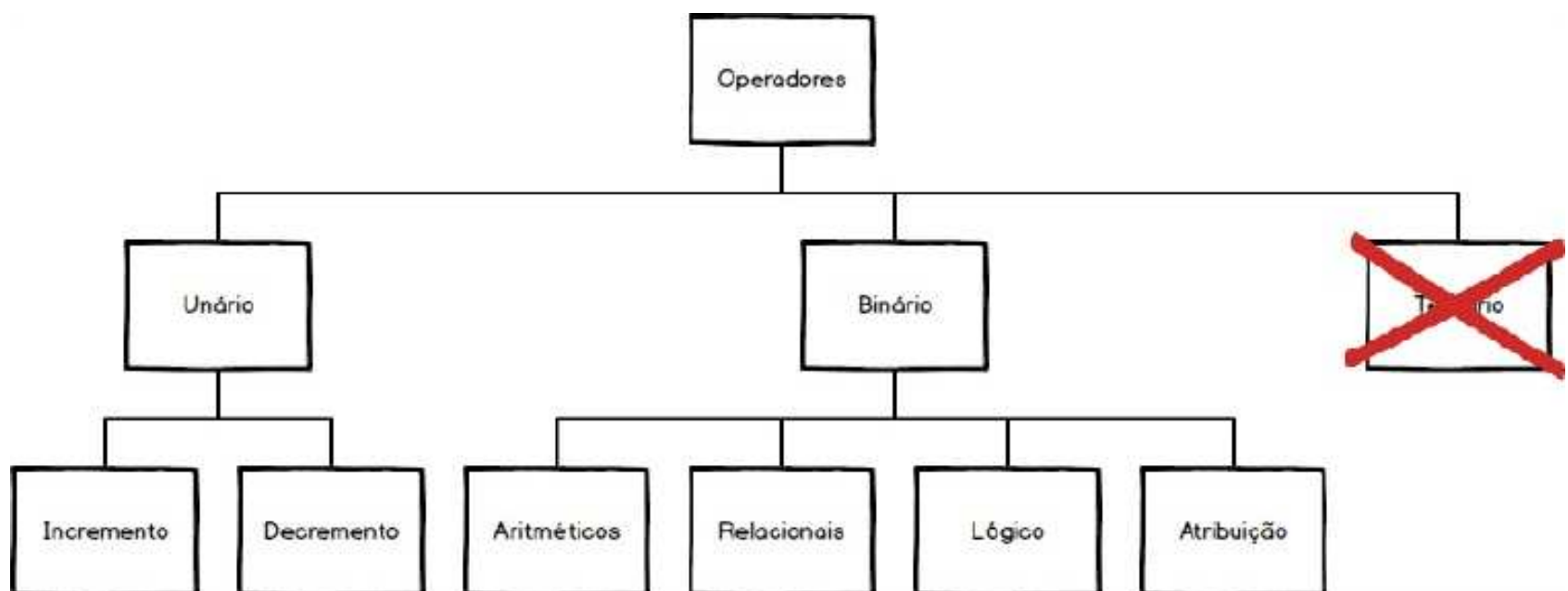
Listagem 35 - Continue

fundamentos/controles/Continue.kt

```
package fundamentos.controles

fun main(args: Array<String>) {
    for (i in 1..10){
        if(i == 5){
            continue
        }
        println("Atual: $i")
    }
}
```

3.12. Operadores



3.12.1. Operadores Binários - Atribuição

Listagem 36 - Operadores de Atribuição

fundamentos/operadores/Atribuicao.kt

```
package fundamentos.operadores

fun main(args: Array<String>) {
    var a : Int = 7
    var b : Int = 3

    b += a // b = b + a
    b -= 4 // b = b - 4
    b *= 2 // b = b * 2
    b /= 2 // b = b / 2
    b %= 2
}
```

Listagem 37 - Destructuring

fundamentos/operadores/OperadorDestructuring.kt

```
package fundamentos.operadores

data class Carro(val marca: String, val modelo: String)

fun main(args: Array<String>) {
    val (marca, modelo) = Carro("Ford", "Fusion")
    println("$marca $modelo")

    val (marido, mulher) = listOf("João", "Maria")
    println("$marido e $mulher")

    val (_, _, terceiroLugar) = listOf("Kimi", "Hamilton", "Alonso")
    println("$terceiroLugar terminou em terceiro lugar.")
}
```

3.12.2. Operadores Binários - Aritméticos

Listagem 38 - Operadores Aritméticos

fundamentos/operadores/Aritmeticos.kt

```
package fundamentos.operadores

fun main(args: Array<String>) {
    val (v1, v2, v3, v4) = listOf(3, 5, 1, 15)

    val soma = v1 + v2 + v3 + v4
    val subtracao = v4 - v2
    val divisao = v4 / v1
    val multiplicacao = v1 * v2
    val modulo = v1 % 2

    println("$soma $subtracao $divisao $multiplicacao $modulo")
}
```

3.12.3. Operadores Binários - Relacionais



Mais detalhes em <https://kotlinlang.org/docs/reference/equality.html>

Listagem 39 - Operadores Relacionais

fundamentos/operadores/Relacionais.kt

```
package fundamentos.operadores

fun main(args: Array<String>) {
    println("Banana" === "Banana")
    println(3 !== 2)
    println(3 < 2)
    println(3 > 2)
    println(3 <= 2)
    println(3 >= 2)

    val d1 = Date(0)
    val d2 = Date(0)

    // Igualdade referencial
    println("Resultado com '===' ${d1 === d2}")

    // Igualdade estrutural
    println("Resultado com '==' ${d1 == d2}")
    println(d1?.equals(d2) ?: (d2 === null))
}
```


3.12.4. Operadores Binários - Lógicos

V && V = V	V V = V	V xor V = F
V && F = F	V F = V	V xor F = V
F && V = F	F V = V	F xor V = V
F && F = F	F F = F	F xor F = F

Figura 60. Tabelas Verdades

Listagem 40 - Operadores Lógicos

fundamentos/operadores/Logicos.kt

```
package fundamentos.operadores

fun main(args: Array<String>) {
    val executouTrabalho1 : Boolean = true
    val executouTrabalho2 : Boolean = true

    val comprouSorvete : Boolean = executouTrabalho1 || executouTrabalho2
    val comprouTv50 : Boolean = executouTrabalho1 && executouTrabalho2
    val comprouTv32 : Boolean = executouTrabalho1 xor executouTrabalho2

    println(comprouSorvete)
    println(comprouTv50)
    println(comprouTv32)
}
```

3.12.5. Operadores Unários

Listagem 41 - Operadores Unários

fundamentos/operadores/Unarios.kt

```
package fundamentos.operadores

fun main(args: Array<String>) {
    var num1 : Int = 1
    var num2 : Int = 2

    num1++
    println(num1)
    --num1
    println(num1)

    // Incremento e decremento
    println(++num1 == num2--)
    println(num1 == num2)
}
```

3.12.6. Não Existe Operador Ternário, mas...

Listagem 42 - If como Ternários

fundamentos/operadores/Ternario1.kt

```
package fundamentos.operadores

fun main(args: Array<String>) {
    val nota : Double = 6.0
    val resultado : String = if(nota >= 7) "Aprovado" else "Reprovado"
    println(resultado)
}
```

Listagem 43 - *If como Ternários*

fundamentos/operadores/Ternario2.kt

```
package fundamentos.operadores

fun obterResultado(nota : Double) : String = if(nota >= 7.0) "Passou" else
"Reprovou"

fun main(args: Array<String>) {
    print(obterResultado(8.3))
}
```

3.13. Conversão

3.13.1. Conversão Número/String

Listagem 44 - *Conversão entre Número e String*

fundamentos/ConversaoNumeroString.kt

```
package fundamentos

fun main(args: Array<String>) {
    val a = 1

    // Número para String
    println(a.toString() + 1)

    // String para Número
    println("1.9".toDouble() + 3)
    println("Teste".toIntOrNull())
    println("Teste".toIntOrNull() ?: 0)
    println("1".toInt() + 3)
}
```

3.13.2. Conversões (Cast)



<https://kotlinlang.org/docs/reference/typecasts.html>

Listagem 45 - Checagem de Tipo

fundamentos/ChecagemDeTipo.kt

```
package fundamentos

fun main(args: Array<String>) {
    val valor = "abc"

    if (valor is String) {
        println(valor)
    } else if (valor !is String) {
        println("Não é uma String")
    }
}
```

Listagem 46 - Smart Cast

fundamentos/SmartCast.kt

```
package fundamentos

fun main(args: Array<String>) {
    souEsperto("Ola")
    souEsperto(9)

    souEsperto2("Opa")
    souEsperto2(7)
    souEsperto2(true)
}

fun souEsperto(x: Any) {
    if (x is String) {
        println(x.toUpperCase())
    } else if (x is Int) {
        println(x.plus(3))
    }
}

fun souEsperto2(x: Any) {
    when(x) {
        is String -> println(x.toUpperCase())
        is Int -> println(x.plus(3))
        else -> println("Repense a sua vida!")
    }
}
```

Listagem 47 - Operador Cast

fundamentos/OperadorCast.kt

```
package fundamentos

fun main(args: Array<String>) {
    val notas = arrayOf(9.6, 3.8, 7.2, 5.5, 4.1)

    for(nota in notas) {
        imprimirConceito(nota.toInt())
    }
}

fun imprimirConceito(nota: Any) {
    when(nota as? Int) {
        10, 9 -> println("A")
        8, 7 -> println("B")
        6, 5 -> println("C")
        4, 3 -> println("D")
        2, 1, 0 -> println("E")
        else -> println("Nota inválida")
    }
}
```


4. Funções

4.1. Funções estáticas *top-level*

Listagem 48 - Função Fora de Classe

funcoes/TopLevel.kt

```
package funcoes

fun main(args: Array<String>) {
    print("O menor valor é ${min(3, 4)}")
}

fun min(a: Int, b: Int): Int {
    return if (a < b) a else b
}
```

É importante lembrar que o código em *Kotlin* é compilado para *Byte Code* (quando executado na *VM*) e como o *Java* é totalmente centrado em classe, **não suporta esse conceito** de função *top-level*. A forma de compatibilizar isso é converter uma função *top-level* para uma função estática de uma classe.

O Exemplo anterior ficaria assim:



```
package funcoes;

public class TopLevelKt {
    public static int min(...) { ... }
}
```

Listagem 49 - Função Fora de Classe (Refatoração)

funcoes/TopLevel.kt

```
package funcoes

fun main(args: Array<String>) {
    print("O menor valor é ${min(3, 4)}")
}

fun min(a: Int, b: Int): Int = if (a < b) a else b
```

4.2. Argumentos Nomeados



O uso de argumentos nomeados além de tornar a chamada à função **mais legível**, permite que os parâmetros sejam passados numa **ordem diferente** do que foi declarado na função.

Listagem 50 - Usando Argumentos Nomeados

funcoes/ArgsNomeados.kt

```
package funcoes

fun main(args: Array<String>) {
    println(relacaoDeTrabalho("João", "Maria"))
    println(relacaoDeTrabalho(funcionario = "João", chefe = "Maria"))
}

fun relacaoDeTrabalho(chefe: String, funcionario: String): String {
    return "$funcionario é subordinado(a) à $chefe."
}
```

4.3. Parâmetros são Imutáveis



A partir da versão do *Kotlin* M5.1 foi removido o suporte para parâmetros mutáveis



O código do próximo exemplo gera um erro:

Kotlin: Val cannot be reassigned

Listagem 51 - Parâmetros Imutáveis

funcoes/ParametrosImutaveis.kt

```
package funcoes

fun incremento(num: Int) {
    num++ // Erro!
}

fun main(args: Array<String>) {
    incremento(3)
}
```

4.4. Parâmetro Padrão



O exemplo além de mostrar o uso de parâmetro com valor padrão, mostra o uso combinado de **parâmetros padrões** e **argumentos nomeados**.

Listagem 52 - Usando Parâmetro Padrão

funcoes/ParamsPadroes.kt

```
package funcoes

fun main(args: Array<String>) {
    println(potencia(2, 3))
    println(potencia(10))
    println(potencia(base = 10))
    println(potencia(expoente = 8))
}

fun potencia(base: Int = 2, expoente: Int = 1): Int {
    return Math.pow(base.toDouble(), expoente.toDouble()).toInt()
}
```

4.5. Múltiplos Retornos Com Destructuring



Múltiplos retornos pode ser simulado com o operador Destructuring

Listagem 53 - Múltiplos Retornos

funcoes/MultiplosRetornos.kt

```
package funcoes

import java.util.*

data class Horario(val hora: Int, val minuto: Int, val segundo: Int)

fun agora(): Horario {
    val agora = Calendar.getInstance()
    with(agora) {
        return Horario(get(Calendar.HOUR), get(Calendar.MINUTE),
get(Calendar.SECOND))
    }
}

fun main(args: Array<String>) {
    val (h, m, s) = agora()
    println("$h:$m:$s")
}
```

4.6. Adicionando Métodos em Classes Existentes

Listagem 54 - Obter Segundo Elemento de uma Lista

funcoes/SegundoElementoList.kt

```
package funcoes

fun main(args: Array<String>) {
    val list = listOf("João", "Maria", "Pedro")
    print(list.secondOrNull())
}

fun <E> List<E>.secondOrNull(): E? =
    if(this.size >= 2) this.get(1) else null
```

4.7. Chamando Função em Java

Listagem 55 - Chamando uma Função Kotlin a partir do Java

funcoes/ChamandoFuncaoKotlin.java

```
package funcoes;

import kotlin.collections.CollectionsKt;

import java.util.ArrayList;

public class ChamandoFuncaoKotlin {

    public static void main(String[] args) {
        ArrayList<String> list =
            CollectionsKt.arrayListOf("João", "Maria", "Pedro");
        System.out.println(SegundoElementoListKt.secondOrNull(list));
    }
}
```

4.8. Argumentos Variáveis (*varargs*)

Listagem 56 - Uso de Argumentos Variáveis

funcoes/VarArgs.kt

```
package funcoes

fun main(args: Array<String>) {
    for(n in ordenar(38, 3, 456, 8, 51, 1, 88, 73)) {
        print("$n ")
    }
}

fun ordenar(vararg numeros: Int): IntArray {
    return numeros.sortedArray()
}
```

4.9. Funções Infix



<https://kotlinlang.org/docs/reference/functions.html#infix-notation>

Listagem 57 - Função Infix

funcoes/FuncaoInfix.kt

```
package funcoes

class Produto(val nome: String, val preco: Double)

infix fun Produto.maisCaroQue(produto: Produto): Boolean =
    this.preco > produto.preco

fun main(args: Array<String>) {
    val p1 = Produto("Ipad", 2349.00)
    val p2 = Produto(preco = 3.49, nome = "Borracha")

    println(p1 maisCaroQue p2)
    println(p2.maisCaroQue(p1))
}
```

4.10. Funções Sempre Retornam um Valor



<https://kotlinlang.org/docs/reference/functions.html#unit-returning-functions>

Listagem 58 - Todas as Funções São Equivalentes

Exemplo 1

```
fun imprimeMaior(a: Int, b: Int) {  
    println(Math.max(a, b))  
}
```

Exemplo 2

```
fun imprimeMaior(a: Int, b: Int): Unit {  
    println(Math.max(a, b))  
}
```

Exemplo 3

```
fun imprimeMaior(a: Int, b: Int): Unit {  
    println(Math.max(a, b))  
    return  
}
```

Exemplo 4

```
fun imprimeMaior(a: Int, b: Int): Unit {  
    println(Math.max(a, b))  
    return Unit  
}
```

Exemplo 5

```
fun imprimeMaior(a: Int, b: Int) {  
    println(Math.max(a, b))  
    return Unit  
}
```

===

Listagem 59 - Passando Função como Parametro 1

funcoes/FuncaoComoParam1.kt

```
package funcoes

class Operacoes {
    fun somar(a: Int, b: Int): Int {
        return a + b
    }
}

fun somar(a: Int, b: Int): Int {
    return a + b
}

fun calc(a: Int, b: Int, funcao: (Int, Int) -> Int) : Int {
    return funcao(a, b)
}

fun main(args: Array<String>) {
    println(calc(2, 3, Operacoes()::somar))
    println(calc(2, 3, ::somar))
}
```

Listagem 60 - Passando Função como Parametro 2

funcoes/FuncaoComoParam2.kt

```
package funcoes

fun <E> filtrar(lista: List<E>, filtro: (E) -> Boolean): List<E> {
    val listaFiltrada = ArrayList<E>()
    for(e in lista) {
        if(filtro(e)) {
            listaFiltrada.add(e)
        }
    }
    return listaFiltrada
}

fun comTresLetras(nome: String): Boolean {
    return nome.length == 3
}

fun main(args: Array<String>) {
    val nomes = listOf("Ana", "Pedro", "Bia", "Gui", "Rebeca")
    println(filtrar(nomes, ::comTresLetras))
}
```

4.11. Inline Function

Listagem 61- *Inline Function*

funcoes/Inline1.kt

```
package funcoes

inline fun transacao(funcao: () -> Unit) {
    println("abrindo transação...")
    try {
        funcao()
    } finally {
        println("fechando transação")
    }
}

fun main(args: Array<String>) {
    transacao {
        println("Executando SQL 1...")
        println("Executando SQL 2...")
        println("Executando SQL 3...")
    }
}
```

Listagem 62- *Inline Function*

funcoes/Inline2.kt

```
package funcoes

inline fun <T> executarComLog(nomeFuncao: String, funcao: () -> T): T {
    println("Entrando no método $nomeFuncao...")
    try {
        return funcao()
    } finally {
        println("Método $nomeFuncao finalizado...")
    }
}

fun somar(a: Int, b: Int): Int {
    return a + b
}

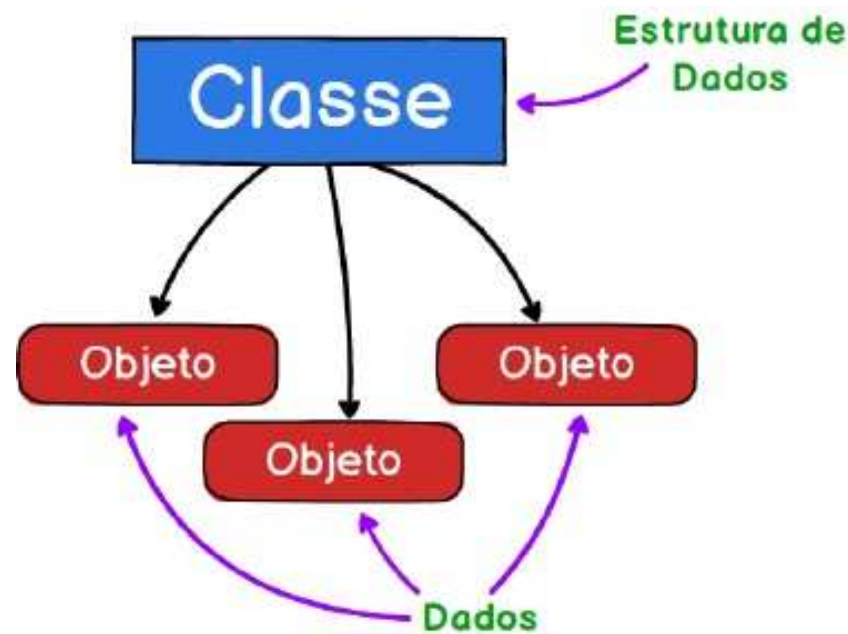
fun main(args: Array<String>) {

    val resultado = executarComLog("somar") {
        somar(4, 5)
    }

    println(resultado)
}
```


5. Classes e Objetos

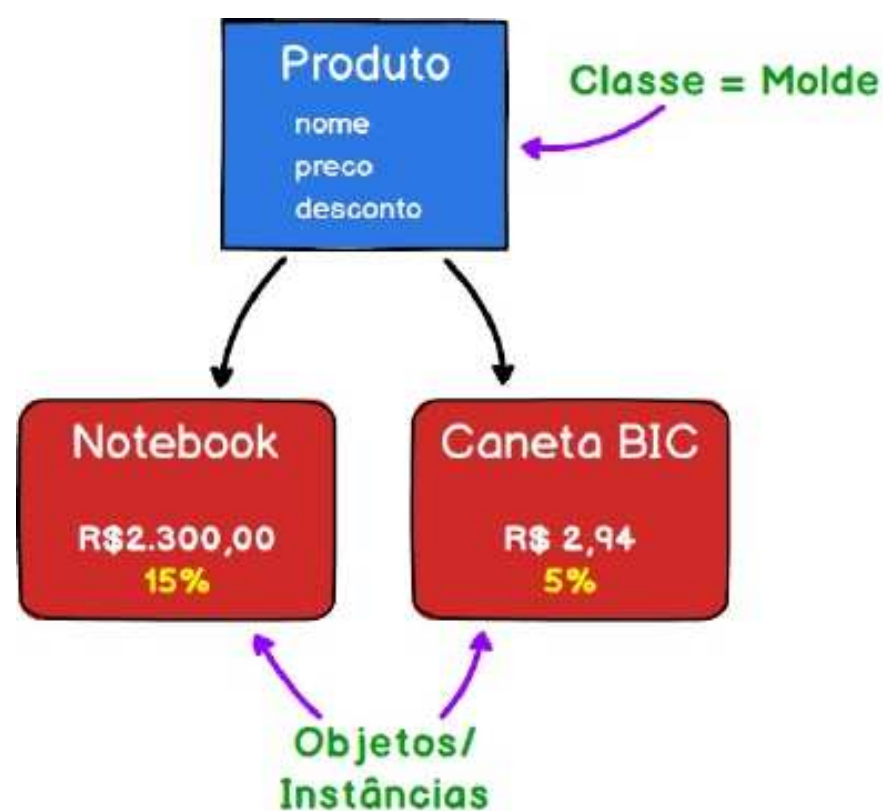
5.1. Classe vs Objeto



Alguns fatos importantes:

- Define um **tipo** (estrutura de dados)
- Representa uma **abstração** (Simplificação)
- Encapsulam **atributos** e **comportamentos**
- **NÃO** suportam métodos estáticos
- Objeto == Instância

Exemplo de relação entre classe e objeto:



5.2. Exemplos Básicos

Listagem 63 - Exemplo Básico de uma Classe

classes/ClasseBasica1.kt

```
package classes

class Cliente {
    var nome: String = ""
}

fun main(args: Array<String>) {
    val cliente = Cliente()
    cliente.nome = "João"
    print("O cliente é ${cliente.nome}")
}
```

Listagem 64 - Exemplo Básico de uma Classe (Estilo Kotlin)

classes/ClasseBasica2.kt

```
package classes

class Pessoa1(var nome: String)
class Pessoa2(val nome: String)
class Pessoa3(nomeInicial: String) {
    val nome: String = nomeInicial
}

fun main(args: Array<String>) {
    val p1 = Pessoa1(nome = "João")
    p1.nome = "Guilherme"
    println("${p1.nome} sabe programar!")

    val p2 = Pessoa2("Maria")
    val p3 = Pessoa3(nomeInicial = "Pedro")
    println("${p2.nome} e ${p3.nome} são legais!")
}
```

5.3. Membros

Listagem 65 - *Membros de uma Classe*

classes/Membros1.kt

```
package classes
```

```
class Data(var dia: Int, var mes: Int, var ano: Int) {  
    fun formatar(): String {  
        return "$dia/$mes/$ano"  
    }  
}
```

```
fun main(args: Array<String>) {  
    var nascimento: Data = Data(dia = 11, mes = 10, ano = 2003)  
  
    println("${nascimento.dia}/${nascimento.mes}/${nascimento.ano}")  
    with(nascimento) { println("${dia}/${mes}/${ano}") }  
  
    nascimento.mes = 11  
    println(nascimento.formatar())  
}
```

Listagem 66 - Membros de uma Classe

classes/Membros2.kt

```
package classes

class Calculadora {
    private var resultado: Int = 0

    fun somar(vararg valores: Int): Calculadora {
        valores.forEach { resultado += it }
        return this
    }

    fun multiplicar(valor: Int): Calculadora {
        resultado *= valor
        return this
    }

    fun limpar(): Calculadora {
        resultado = 0
        return this
    }

    fun print(): Calculadora {
        println(resultado)
        return this
    }

    fun obterResultado(): Int {
        return resultado
    }
}

fun main(args: Array<String>) {
    val calculadora = Calculadora()
    calculadora.somar(1, 2, 3).multiplicar(3).print()
    calculadora.somar(7, 10).print().limpar()

    println(calculadora.obterResultado())
}
```

5.4. Tipos de Variáveis/Constantes

Listagem 67 - Tipos de Variáveis/Constantes em Kotlin

classes/TiposVariaveis.kt

```
package classes

val diretamenteNoArquivo = "Bom dia"

fun topLevel() {
    val local = "Fulano!"
    println("$diretamenteNoArquivo $local")
}

class Coisa {
    var variavelDeInstancia: String = "Boa noite"

    companion object {
        @JvmStatic val constanteDeClasse = "Ciclano"
    }

    fun fazer() {
        val local: Int = 7

        if (local > 3) {
            val variavelDeBloco = "Beltrano"
            println("$variavelDeInstancia, $constanteDeClasse, $local e $variavelDeBloco!")
        }
    }
}

fun main(args: Array<String>) {
    topLevel()
    Coisa().fazer()
    println(Coisa.constanteDeClasse)
}
```

5.5. Data Classe

Listagem 68 - Class x Data Class

classes/ClassVsDataClass.kt

```
package classes

class Geladeira(val marca: String, val litros: Int)
data class Televisao(val marca: String, val polegadas: Int)

fun main(args: Array<String>) {
    val g1 = Geladeira("Brastemp", 320)
    val g2 = Geladeira("Brastemp", 320)

    println(g1 == g2) // equals

    val tv1 = Televisao("Samsung", 32)
    val tv2 = Televisao("Samsung", 32)

    println(tv1 == tv2) // equals
    println(tv1 === tv2)
    println(tv1.toString())
    println(tv1.copy())
    println(tv1.copy(polegadas = 42))

    // Destructuring em data class
    val (marca, pol) = tv1
    println("$marca $pol'")
}
```

5.6. Construtor

Listagem 69 - Construtor (Estilo Java)

classes/Construtor1.kt

```
package classe

class Filme {
    val nome: String
    val anoLancamento: Int
    val genero: String

    constructor(nome: String, anoLancamento: Int, genero: String) {
        this.nome = nome
        this.anoLancamento = anoLancamento
        this.genero = genero
    }
}

fun main(args: Array<String>) {
    val filme = Filme("O Poderoso Chefão", 1972, "Drama")
    println("${filme.nome} foi lançado em ${filme.anoLancamento}.")
}
```

Listagem 70 - Construtor (Estilo Kotlin)

classes/Construtor2.kt

```
package classes

class Filme2(val nome: String, val anoLancamento: Int, val genero: String)

fun main(args: Array<String>) {
    val filme = Filme2("Monstros S.A", 2001, "Comédia")
    println("${filme.nome} foi lançado em ${filme.anoLancamento}.")
}
```

Listagem 71 - *Bloco init*

classes/BlocoInit.kt

```
package classes

class Filme3(nome: String, anoLancamento: Int, genero: String) {
    val nome: String
    val anoLancamento: Int
    val genero: String

    init {
        this.nome = nome
        this.anoLancamento = anoLancamento
        this.genero = genero
    }
}

fun main(args: Array<String>) {
    val filme = Filme3("Os Incríveis", 2004, "Ação")
    println("${filme.nome} foi lançado em ${filme.anoLancamento}.")
}
```

5.7. Getters & Setters

Listagem 72 - Classe com Getters & Setters

classes/GettersSetters.kt

```
package classe

class Cliente {
    constructor(nome: String) {
        this.nome = nome
    }

    var nome: String
    get() = "Meu nome é ${field}"
    set(value) {
        field = value.takeIf { value.isNotEmpty() } ?: "Anônimo"
    }
}

fun main(args: Array<String>) {
    val c1 = Cliente("")
    println(c1.nome)

    val c2 = Cliente("Pedro")
    println(c2.nome)
}
```

Listagem 73 - Getters Com Valor Calculado

classes/GettersCalculados.kt

```
package classe

class Produto(var nome: String, var preco: Double, var desconto: Double, var
ativo: Boolean) {
    val inativo: Boolean get() = !ativo
    val precoComDesconto: Double get() = preco * (1 - desconto)
}

fun main(args: Array<String>) {
    val p1 = Produto("iPad", 2349.90, 0.20, ativo = true)
    println(p1.precoComDesconto)

    val p2 = Produto("Galaxy Note 7", 2699.49, 0.50, ativo = false)
    println("${p2.nome}\n\tDe: R$ ${p2.preco} \n\tPor: R$
${p2.precoComDesconto}")
    if(p2.inativo) {
        p2.preco = 0.0
        println("Depois de inativo: R$ ${p2.precoComDesconto}")
    }
}
```

5.8. Membros Classe vs Instância

Listagem 74 - *Membros Classe vs Instância*

classes/MembrosClasseVsInstancia.kt

```
package classes

class ItemDePedido(val nome: String, var preco: Double) {
    companion object {
        fun create(nome: String, preco: Double) = ItemDePedido(nome, preco)
        @JvmStatic var desconto: Double = 0.0
    }

    fun precoComDesconto(): Double {
        return preco - preco * desconto
    }
}

fun main(args: Array<String>) {
    var item1 = ItemDePedido.create("TV 50 Polegadas", 2989.90)
    var item2 = ItemDePedido("Liquidificador", 200.0)
    ItemDePedido.desconto = 0.10

    println(item1.precoComDesconto())
    println(item2.precoComDesconto())
}
```

5.9. Passagem por Referência

Listagem 75 - Passagem por Referência

classes/PassagemReferencia.kt

```
package classes

// Erro!! Kotlin: Val cannot be reassigned
// fun porReferencia(velociade: Int) {
//     velocidade++
// }

fun porReferencia(carro: Carro) {
    carro.velocidade++
}

data class Carro(var marca: String, var modelo: String, var velocidade: Int = 0)

fun main(args: Array<String>) {
    var carro1 = Carro("Ford", "Fusion")

    var carro2 = carro1
    carro2.modelo = "Edge"
    println(carro1)

    carro1 = Carro("Audi", "A4")

    porReferencia(carro1)
    porReferencia(carro2)

    println(carro1)
    println(carro2)
}
```

5.10. Enum

Listagem 76 - Enum Básico

classes/Enum1.kt

```
package classes

enum class DiaSemana {
    DOMINGO, SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO
}

fun main(args: Array<String>) {
    print("O melhor dia da semana é ${DiaSemana.SABADO}.")
}
```

Listagem 77 - Enum com Atributos

classes/Enum2.kt

```
package classes

enum class DiaSemana2 (
    val id: Int, val nome: String, val util: Boolean
) {
    DOMINGO(1, "Domingo", false),
    SEGUNDA(2, "Segunda", true),
    TERCA(3, "Terça", true),
    QUARTA(4, "Quarta", true),
    QUINTA(5, "Quinta", true),
    SEXTA(6, "Sexta", true),
    SABADO(7, "Sábado", false)
}

fun main(args: Array<String>) {
    for (dia in DiaSemana2.values()) {
        println("${dia.nome} é um dia ${if(dia.util) "útil" else "do fim de semana"}.")
    }
}
```

#include::this.adoc[]

6. Programando com Lambdas



<http://beust.com/weblog/2015/10/30/exploring-the-kotlin-standard-library/>

6.1. Minhas Lambdas

6.1.1. Exemplo Simples

Listagem 78 - Criando um Lambda simples

lambdas/Lambda1.kt

```
package lambdas

fun main(args: Array<String>) {
    val soma = {x : Int, y : Int -> x + y}
    println(soma(4, 6))
}
```

6.1.2. Por que Lambda?

Listagem 79 - Motivação Para Usar Lambda

lambdas/Lambda2.kt

```
package lambdas

interface Operacao {
    fun executar(a: Int, b: Int): Int
}

class Multiplicacao: Operacao {
    override fun executar(a: Int, b: Int): Int {
        return a * b
    }
}

class Calculadora {
    fun calcular(a: Int, b: Int): Int {
        return a + b
    }

    fun calcular(a: Int, b: Int, operacao: Operacao): Int {
        return operacao.executar(a, b)
    }

    fun calcular(a: Int, b: Int, operacao: (Int, Int) -> Int): Int {
        return operacao(a, b)
    }
}

fun main(args: Array<String>) {
    val calculadora = Calculadora()
    val resultado1 = calculadora.calcular(3, 4)
    val resultado2 = calculadora.calcular(3, 4, Multiplicacao())
    val subtracao = { a: Int, b: Int -> a - b }
    val resultado3 = calculadora.calcular(3, 4, subtracao)

    println("$resultado1 $resultado2 $resultado3")
}
```

6.2. Lambdas Encontradas na API

6.2.1. SortedBy

Listagem 80 - Ordenando nomes

lambdas/SortedBy.kt

```
package lambdas

fun main(args: Array<String>) {
    val nomes = arrayListOf("Renata", "Bernardo", "Willian", "Andreia", "Caio")
    val ordenados = nomes.sortedBy { it.reversed() }

    println(ordenados)
}
```

6.2.2. Filter

Listagem 81 - Obter os alunos aprovados

lambdas/Filter.kt

```
package lambdas

data class Aluno(val nome: String, val nota: Double)

fun main(args: Array<String>) {
    val alunos = arrayListOf(
        Aluno("Pedro", 7.4),
        Aluno("Artur", 8.0),
        Aluno("Rafael", 9.7),
        Aluno("Ricardo", 5.7)
    )

    val aprovados = alunos.filter { it.nota > 7.0 }.sortedBy { it.nome }
    println(aprovados)
}
```

6.2.3. Map

Listagem 82 - Lista Maiúscula

lambdas/Map1.kt

```
package lambdas

fun main(args: Array<String>) {
    val alunos = arrayListOf("Pedro", "Tiago", "Jonas")
    alunos.map { it.toUpperCase() }.apply { print(this) }
}
```

Listagem 83 - Média dos Preços do Material Escolar

lambdas/Map2.kt

```
package lambdas

class Produto(val nome: String, val preco: Double)

val materialEscolar = listOf(
    Produto("Fichário escolar", 21.90),
    Produto("Lápis de cor", 11.90),
    Produto("Borracha bicolor", 0.70),
    Produto("Apontador com depósito", 1.80)
)

fun main(args: Array<String>) {
    val totalizar = { total: Double, atual: Double -> total + atual }
    val precoTotal = materialEscolar.map { it.preco }.reduce(totalizar)

    println("O preço médio é R$ ${precoTotal / materialEscolar.size}.")
}
```

6.2.4. TakeIf

Listagem 84 - *Lambdas em Strings*

lambdas/TakeIf.kt

```
package lambdas

fun main(args: Array<String>) {
    println("Digite sua mensagem: ")
    val entrada = readLine()

    val mensagem = entrada.takeIf { it != "" } ?: "Sem mensagem"
    println(mensagem)
}
```

6.2.5. Apply

Listagem 85 - *Apply*

lambdas/Apply.kt

```
package lambdas

class Calculadora {
    var resultado = 0

    fun soma(valor1: Int, valor2: Int) {
        resultado += valor1 + valor2
    }

    fun add(valor: Int) {
        resultado += valor
    }
}

fun main(args: Array<String>) {
    val calculadora = Calculadora()

    calculadora.apply { soma(4, 5) }.apply { add(5) }.apply { println(resultado)
}

    calculadora.apply {
        soma(4, 5)
        add(5)
        print(resultado)
    }

    // inline function
    with(calculadora) {
        soma(4, 5)
        add(5)
        print(resultado)
    }
}
```

6.2.6. Let

Listagem 86 - Utilizando Let

lambdas/Let.kt

```
package classes

fun main(args: Array<String>) {
    val listWithNulls: List<String?> = listOf("A", null, "B", null, "C", null)

    for (item in listWithNulls) {
        item?.let {
            println(it)
        }
    }
}
```

6.2.7. Run

Listagem 87 - Utilizando Run

lambdas/Run.kt

```
package lambdas

data class Casa(var endereco : String = "", var num : Int = 0)

fun main(args: Array<String>) {
    var casa = Casa()

    casa.run {
        endereco = "Rua ABC"
        num = 1544
    }

    println(casa)
}
```

7. Arrays e Coleções

7.1. Arrays e Collections

7.2. Arrays

Listagem 88 - *Array*

collections/Arrays.kt

```
package collections

fun main(args: Array<String>) {
    val numeros = Array<Int>(10) { i -> i * 10 }

    for(numero in numeros) {
        println(numero)
    }

    println(numeros.get(1))
    println(numeros.size)

    numeros.set(1, 1234)
    println(numeros[1])

    numeros[1] = 2345
    println(numeros[1])
}
```

7.3. Array List

Listagem 89 - *Array List*

collections/ArrayListStrings.kt

```
package collections

fun main(args: Array<String>) {
    val strings = arrayListOf("Carro", "Moto", "Barco", "Avião")

    for(item in strings) {
        println(item.toUpperCase())
    }
}
```

Listagem 90 - *Array List com Numeros*

collections/ArrayListInt.kt

```
package collections

fun main(args: Array<String>) {
    val pares = arrayListOf(2, 4, 6)
    val impares = intArrayOf(1, 3, 5)

    for(n in impares.union(pares).sorted()) {
        print("$n ")
    }
}
```

Listagem 91 - *Array List Heterogêneo*

collections/ArrayListMix.kt

```
package collections

fun main(args: Array<String>) {
    val listaMix = arrayListOf("Rafael", true, 1, 3.14, 'p')

    for(item in listaMix) {
        if(item is String) { // smart cast
            println(item.toUpperCase())
        } else {
            println(item)
        }
    }
}
```

Listagem 92 - *União de Array Lists*

collections/ArrayListUniao.kt

```
package collections

fun main(args: Array<String>) {
    val numeros = arrayListOf(1, 2, 3, 4, 5)
    val strings = arrayListOf("Rafael", "Pedro", "Leandro", "Gustavo")
    val uniao = numeros + strings

    for(item in uniao) {
        println(item)
    }
}
```

Listagem 93 - *Array List com Objetos*

collections/ArrayListObjetos.kt

```
package collections

data class Frutas (var nome : String, var preco: Double)

fun main(args: Array<String>) {
    var frutas = arrayListOf(Frutas("Banana", 1.50), Frutas("Morango", 3.20))

    for(fruta in frutas) {
        println("${fruta.nome} - R$ ${fruta.preco}")
    }
}
```

7.4. For (each)

Listagem 94 - *For (each)*

collections/ExemploFor1.kt

```
package collections

fun main(args: Array<String>) {
    val alunos = arrayListOf("Amanda", "André", "Bernardo", "Carlos")

    for(aluno in alunos) {
        println(aluno)
    }

    for((index, aluno) in alunos.withIndex()) {
        println("$index - $aluno")
    }
}
```

Listagem 95 - *For (each) Lambda*

collections/ExemploFor2.kt

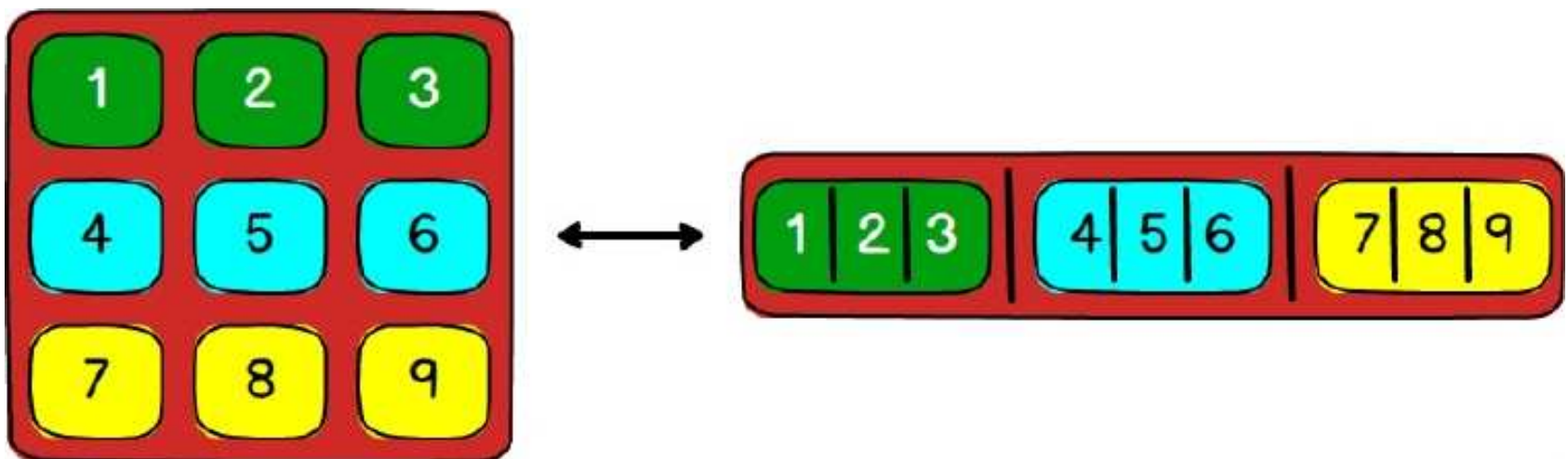
```
package collections

fun main(args: Array<String>) {
    val alunos = arrayListOf("Amanda", "André", "Bernardo", "Carlos")

    alunos.forEach {
        println(it)
    }
}
```

7.5. Matriz

Na prática, matriz é um array de arrays, então podemos visualizar melhor na imagem a seguir:



Listagem 96 - Matriz

collections/Matriz.kt

```
package collections
```

```
fun main(args: Array<String>) {  
    val matriz = Array(3) { arrayOfNulls<Int>(3) }  
  
    matriz[0][0] = 1  
    matriz[0][1] = 2  
    matriz[0][2] = 3  
    matriz[1][0] = 4  
    matriz[1][1] = 5  
    matriz[1][2] = 6  
    matriz[2][0] = 7  
    matriz[2][1] = 8  
    matriz[2][2] = 9  
  
    for ((linha, linhaArray) in matriz.withIndex()) {  
        for ((coluna, valor) in linhaArray.withIndex()) {  
            println("[${linha}][${coluna}] - ${matriz[linha][coluna]} == $valor")  
        }  
    }  
  
    matriz.forEach {  
        it.forEach {  
            println(it)  
        }  
    }  
}
```

7.6. Set

Listagem 97 - Set Heterogêneo

collections/ConjuntoBaguncado.kt

```
package collections

fun Any.print() = println(this)

fun main(args: Array<String>) {
    val conjunto = hashSetOf(3, 'a', "texto", true, 3.14)

    // conjunto.get(1)

    conjunto.add(3).print()
    conjunto.add(10).print()

    conjunto.size.print()

    conjunto.remove("a").print()
    conjunto.remove('a').print()

    conjunto.contains('a').print()
    conjunto.contains("Texto").print()
    conjunto.contains("texto").print()

    val nums = setOf(1, 2, 3) // somente leitura
    // nums.add(4)

    (conjunto + nums).print()
    (conjunto - nums).print()

    conjunto.intersect(nums).print() // não muda o conjunto
    conjunto.retainAll(nums) // muda o conjunto
    conjunto.print()

    conjunto.clear()
    conjunto.isEmpty().print()
}
```

Listagem 98 - Set Homogêneo

collections/ConjuntoComportado.kt

```
package collections
```

```
fun main(args: Array<String>) {
    val aprovados = HashSetOf("João", "Maria", "Pedro", "Ana", "Joana")
    // aprovados.add(1)

    println("Sem ordem...")
    for (aprovado in aprovados) {
        aprovado.print()
    }

    val aprovadosNaOrdem1 = linkedSetOf("João", "Maria", "Pedro", "Ana", "Joana")

    println("\nLinked...")
    for (aprovado in aprovadosNaOrdem1) {
        aprovado.print()
    }

    val aprovadosNaOrdem2 = sortedSetOf("João", "Maria", "Pedro", "Ana", "Joana")

    println("\nSorted...")
    for (aprovado in aprovadosNaOrdem2) {
        aprovado.print()
    }

    // Ordem maluca...
    aprovados.sortedBy { it.substring(1) }.print()
}
```

```
#include::list.adoc[]
```

```
#include::queue.adoc[]
```

```
#include::stack.adoc[]
```

7.7. HashMap

Listagem 99 - Hash Map

collections/Map1.kt

```
package collections

fun main(args: Array<String>) {
    var map = HashMap<Long, String>()

    map.put(10020030040, "João")
    map.put(30040050060, "Maria")
    map.put(60070080090, "Pedro")

    map.put(60070080090, "Pedro Filho")

    for (par in map) {
        println(par)
    }

    for (nome in map.entries) {
        println(nome)
    }

    for (cpf in map.keys) {
        println(cpf)
    }

    for ((cpf, nome) in map) {
        println("$nome (CPF: $cpf)")
    }

    map.size.print()
    map.get(30040050060).print()
    map.contains(30040050060).print()
    map.remove(30040050060)?.print()
    map.clear()
    map.isEmpty().print()
}
```

Listagem 100 - Hash Map Inicializado

collections/Map2.kt

```
package collections

fun main(args: Array<String>) {
    var map = hashMapOf(1 to "Gui", 2 to "Rebeca", 3 to "Cibalena")

    for ((id, nome) in map) {
        println("$id) $nome")
    }
}
```

7.8. Hashcode & Equals

Listagem 101 - Entendendo o Hashcode & Equals

collections/HashCodeEquals.kt

```
package collections

class Objeto(val nome: String, val descricao: String) {

    override fun hashCode(): Int {
        return nome.length
    }

    override fun equals(other: Any?): Boolean {
        if(other is Objeto) {
            return nome.equals(other.nome, ignoreCase = true)
        } else {
            return false
        }
    }
}

fun main(args: Array<String>) {
    val conjunto = HashSetOf(
        Objeto("Cadeira", "..."), // Hashcode = 7
        Objeto("Mesa", "..."), // Hashcode = 4
        Objeto("Prato", "..."), // Hashcode = 6
        Objeto("Colher", "...") // Hashcode = 6
    )

    conjunto.contains(Objeto("prato", "???")).print()
}
```

8. Orientação a Objetos

8.1. Os Pilares da OO

8.2. Encapsulamento

Listagem 102 - Encapsulamento

oo/encapsulamento/Encapsulamento.kt

```
package oo.encapsulamento

private val dentroDoArquivo = 1
// protected val protegidoNaoSuportadoAqui = 1
internal val dentroDoProjeto = 1
val publico = 1

private fun dentroDoArquivo() = 1
// protected fun protegidoNaoSuportadoAqui() = 1
internal fun dentroDoProjeto() = 1
fun publico() = 1

open class Capsula {
    private val dentroDoObjeto = 1
    protected val vaiPorHeranca = 1
    internal val dentroDoProjeto = 1
    val publico = 1

    private fun dentroDoObjeto() = 1
    protected fun vaiPorHeranca() = 1
    internal fun dentroDoProjeto() = 1
    fun publico() = 1
}

class CapsulaFilha : Capsula() {
    fun verificarAcesso() {
        // println(Capsula().dentroDoObjeto)
        // println(Capsula().vaiPorHeranca)
        println(vaiPorHeranca)
        println(dentroDoProjeto)
        println(publico)
    }
}

fun verificarAcesso() {
    // println(Capsula().dentroDoObjeto())
    // println(Capsula().vaiPorHeranca())
    println(Capsula().dentroDoProjeto())
    println(Capsula().publico())
}
```

8.3. Herança

Listagem 103 - Exercício sobre Herança (Parte 1)

oo/heranca/Carro.kt

```
package oo.heranca

open class Carro(val velocidadeMaxima: Int = 200, var velocidadeAtual: Int = 0) {

    protected fun alterarVelocidadeEm(velocidade: Int) {
        val novaVelocidade = velocidadeAtual + velocidade
        velocidadeAtual = when(novaVelocidade) {
            in 0..velocidadeMaxima -> novaVelocidade
            in velocidadeMaxima + 1..Int.MAX_VALUE -> velocidadeMaxima
            else -> 0
        }
    }

    open fun acelerar() {
        alterarVelocidadeEm(5)
    }

    open fun frear() {
        alterarVelocidadeEm(-5)
    }

    override fun toString(): String {
        return "A velocidade atual é $velocidadeAtual Km/h."
    }
}
```

Listagem 104 - Exercício sobre Herança (Parte 2)

oo/heranca/Esportivo.kt

```
package oo.heranca

interface Esportivo {
    var turbo: Boolean

    fun ligarTurbo() {
        turbo = true
    }

    fun desligarTurbo() {
        turbo = false
    }
}
```

Listagem 105 - Exercício sobre Herança (Parte 3)

oo/heranca/Ferrari.kt

```
package oo.heranca

class Ferrari: Carro(velocidadeMaxima = 350), Esportivo {

    override var turbo: Boolean = false
        get() = field
        set(value) { field = value }

    override fun acelerar() {
        super.alterarVelocidadeEm(if(turbo) 50 else 25)
    }

    override fun frear() {
        super.alterarVelocidadeEm(-25)
    }
}
```

Listagem 106 - Exercício sobre Herança (Parte 4)

oo/heranca/Teste.kt

```
package oo.heranca

fun main(args: Array<String>) {
    val carro = Ferrari()
    carro.acelerar()
    println(carro)

    carro.ligarTurbo()
    carro.acelerar()
    println(carro)

    carro.frear()
    carro.frear()
    println(carro)
}
```

8.3.1. Usando this() & super()

Listagem 107 - Usando *this()* & *super()*

oo/heranca/Heranca2.kt

```
package oo.heranca

open class Animal(val nome: String)

class Cachorro : Animal {
    private val altura: Double

    constructor(nome: String) : this(nome, 0.0)

    constructor(nome: String, altura: Double) : super(nome) {
        this.altura = altura
    }

    override fun toString(): String = "$nome tem $altura cm de altura."
}

fun main(args: Array<String>) {
    val dogAlemao = Cachorro("Spyke", 84.3)
    val yorkshire = Cachorro("Lady Di")

    println(dogAlemao)
    println(yorkshire)
}
```

8.4. Polimorfismo

Listagem 108 - Sem Polimorfismo

oo/polimorfismo/SemPolimorfismo.kt

```
package oo.polimorfismo

class Feijao(val peso: Double)
class Arroz(val peso: Double)

class Pessoa(var peso: Double) {
    fun comer(feijao: Feijao) {
        peso += feijao.peso
    }

    fun comer(arroz: Arroz) {
        peso += arroz.peso
    }
}

fun main(args: Array<String>) {
    val feijao = Feijao(0.3)
    val arroz = Arroz(0.3)

    val joao = Pessoa(80.5)
    joao.comer(feijao)
    joao.comer(arroz)

    println(joao.peso)
}
```

Listagem 109 - Usando Polimorfismo

oo/heranca/ComPolimorfismo.kt

```
package oo.polimorfismo

open class Comida(val peso: Double)
class Feijao(peso: Double) : Comida(peso)
class Arroz(peso: Double) : Comida(peso)
class Ovo(peso: Double) : Comida(peso)

class Pessoa(var peso: Double) {
    fun comer(comida: Comida) {
        peso += comida.peso
    }
}

fun main(args: Array<String>) {
    val feijao = Feijao(0.3)
    val arroz = Arroz(0.3)
    val ovo = Ovo(0.2)

    val joao = Pessoa(80.5)
    joao.comer(feijao)
    joao.comer(arroz)
    joao.comer(ovo)

    println(joao.peso)
}
```


9. Tópicos Avançados

9.1. Recursividade

Listagem 110 - *Recursão*

avancado/Recurso.kt

```
package avancado

fun fatorial(num: Int): Int = when(num) {
    in 0..1 -> 1
    in 2..Int.MAX_VALUE -> num * fatorial(num - 1)
    else -> throw IllegalArgumentException("Número negativo")
}

fun main(args: Array<String>) {
    println("Resultado: ${fatorial(5)}")
}
```

9.2. Genéricos



<https://kotlinlang.org/docs/reference/generics.html>

Listagem 111 - Genericos

avancado/Genericos.kt

```
package avancado

class Caixa<T>(val objeto: T) {
    private val objetos = ArrayList<T>()

    init {
        adicionar(objeto)
    }

    fun adicionar(novoObjeto: T) {
        objetos.add(novoObjeto)
    }

    override fun toString(): String = objetos.toString()
}

fun main(args: Array<String>) {
    val materialEscolar = Caixa("Caneta")
    materialEscolar.adicionar("Lapis")
    materialEscolar.adicionar("Borracha")
    println(materialEscolar)

    val numeros = Caixa(objeto = 1)
    numeros.adicionar(3)
    println(numeros)
}
```

9.3. Sobrecarga de Operadores

Listagem 112 - Sobrecarga de Operadores

avancado/SobrecargaDeOperadores.kt

```
package avancado

data class Ponto(val x: Int, val y: Int) {
    operator fun plus(other: Ponto): Ponto = Ponto(x + other.x, y + other.y)
    operator fun unaryMinus(): Ponto = Ponto(-x, -y)
}

fun main(args: Array<String>) {
    val ponto1 = Ponto(10, 20)
    val ponto2 = Ponto(10, 20)

    println(-ponto1)
    println(ponto1 + ponto2)
}
```

9.4. Anotação & Reflexão

Listagem 113 - Uso de Anotações e Reflexão

avancado/AnotacaoComReflexao.kt

```
package avancado

annotation class Positivo
annotation class NaoVazio

class Pessoa(id: Int, nome: String) {
    @Positivo
    var id: Int = id

    @NaoVazio
    var nome: String = nome
}

fun getValue(objeto: Any, atributo: String): Any {
    val field = objeto.javaClass.getDeclaredField(atributo)
    val isAccessibleOriginal = field.isAccessible

    field.isAccessible = true
    val value = field.get(objeto)
    field.isAccessible = isAccessibleOriginal

    return value
}
```

```

fun validar(objeto: Any): List<String> {
    val msgs = ArrayList<String>()
    objeto::class.members.forEach { member ->
        member.annotations.forEach { annotation ->
            val value = getValue(objeto, member.name)
            when (annotation) {
                is Positivo ->
                    if (value !is Int || value <= 0) {
                        msgs.add("O valor '$value' não é um número positivo!")
                    }
                is NaoVazio ->
                    if (value !is String || value.trim().isEmpty()) {
                        msgs.add("O valor '$value' não é uma string válida!")
                    }
            }
        }
    }
    return msgs
}

fun main(args: Array<String>) {
    val obj1 = Pessoa(1, "Chico")
    println(validar(obj1))

    val obj2 = Pessoa(-1, "")
    println(validar(obj2))
}

```

10. Conclusão

10.1. Próximos Passos

Appendix A: Tabela de Códigos

- Listagem 4 - *Primeiro Programa em Kotlin*
- Listagem 6 - *Variáveis em Kotlin*
- Listagem 7 - *Cuidado com as inferências (Parte 1)*
- Listagem 8 - *Cuidado com as inferências (Parte 2)*
- Listagem 9 - *Constantes em Kotlin*
- Listagem 10 - *Constantes vindas do Java*