CURSOS TÉCNICOS

DESENVOLVIMENTO DE APLICATIVOS I

Eixo Informática para Internet

QI Faculdade & Escola Técnica

SUMÁRIO

APRESENTAÇÃO DA DISCIPLINA

Seja bem-vindo (a) à disciplina Desenvolvimento de Aplicativos I

Nessa disciplina iremos estudar sobre o desenvolvimento de aplicações mobile nativo (Android) usando a linguagem de programação mais utilizada no mercado de trabalho.

Kotlin é uma linguagem moderna e flexível, amplamente utilizada por desenvolvedores Android, especialmente em grandes empresas. Ela aumenta a produtividade e garante um código mais seguro para aplicativos. Se você já desenvolve em Java, pode integrar Kotlin facilmente, pois são completamente interoperáveis. A linguagem foi lançada em 2016 e é apoiada pela Google e JetBrains, sendo uma das linguagens oficiais para o Android. A comunidade de Kotlin é engajada e crescente. Nesta formação, você aprenderá a desenvolver aplicativos do zero utilizando Kotlin no Android Studio, implementar funcionalidades e lidar com dados.

Bons estudos!

Núcleo de Ensino Técnico e Profissionalizante – NETeP.

UNIDADE 1

1. FUNDAMENTOS DA LINGUAGEM KOTLIN

Para estas abordagens acerca dos fundamentos da linguagem Kotlin, o foco recai sobre uma linguagem de programação mais moderna denominada Kotlin, que tem como propósito incrementar a produtividade e garantir um código mais seguro para aplicativos.

1.1 O que é Kotlin?

Kotlin é uma linguagem de programação moderna, de código aberto e orientada a objetos, que foi desenvolvida pela JetBrains. Ela foi projetada para ser uma alternativa ao Java, com o objetivo de oferecer uma experiência de desenvolvimento mais produtiva e segura.

Lançada em 2011, a linguagem Kotlin foi projetada para interoperar perfeitamente com o Java, permitindo que desenvolvedores aproveitem a vasta biblioteca e ecossistema existentes do Java. Isso significa que é possível usar Kotlin em projetos existentes que utilizam Java e também utilizar bibliotecas Java em projetos Kotlin sem grandes problemas.

Algumas das principais características e vantagens do Kotlin incluem:

- **Concisão:** o Kotlin reduz a quantidade de código necessário para realizar tarefas comuns, tornando-o mais legível e fácil de manter.
- **Segurança:** a linguagem oferece recursos de segurança de tipo, o que ajuda a evitar erros comuns relacionados à nulos e a reduzir a probabilidade de exceções em tempo de execução.
- **Interoperabilidade:** como mencionado anteriormente, o Kotlin pode ser facilmente integrado a projetos Java, permitindo uma transição suave para a linguagem e a coexistência de ambos.
- **Orientação a objetos:** Kotlin é uma linguagem orientada a objetos, o que significa que é possível aplicar conceitos como herança, polimorfismo e encapsulamento.
- **Funcionalidade:** Kotlin também suporta programação funcional, permitindo o uso de funções de ordem superior, lambdas e outras construções funcionais.
- **Extensibilidade:** é possível estender classes existentes com novas funcionalidades sem a necessidade de modificar o código-fonte original, graças ao recurso de extensões em Kotlin.

- **Ferramentas e suporte**: o Kotlin possui um ecossistema maduro, incluindo suporte para várias IDEs populares, como o IntelliJ IDEA (da JetBrains), bem como plugins para outras ferramentas de desenvolvimento.

Uma importante observação no trabalho e desenvolvimento em qualquer linguagem de programação é a utilização da documentação online desta linguagem. É neste local que você encontrará sempre os tópicos e releases mais atuais da linguagem de programação. O link da documentação do Kotlin se encontra nas referências bibliográficas deste documento

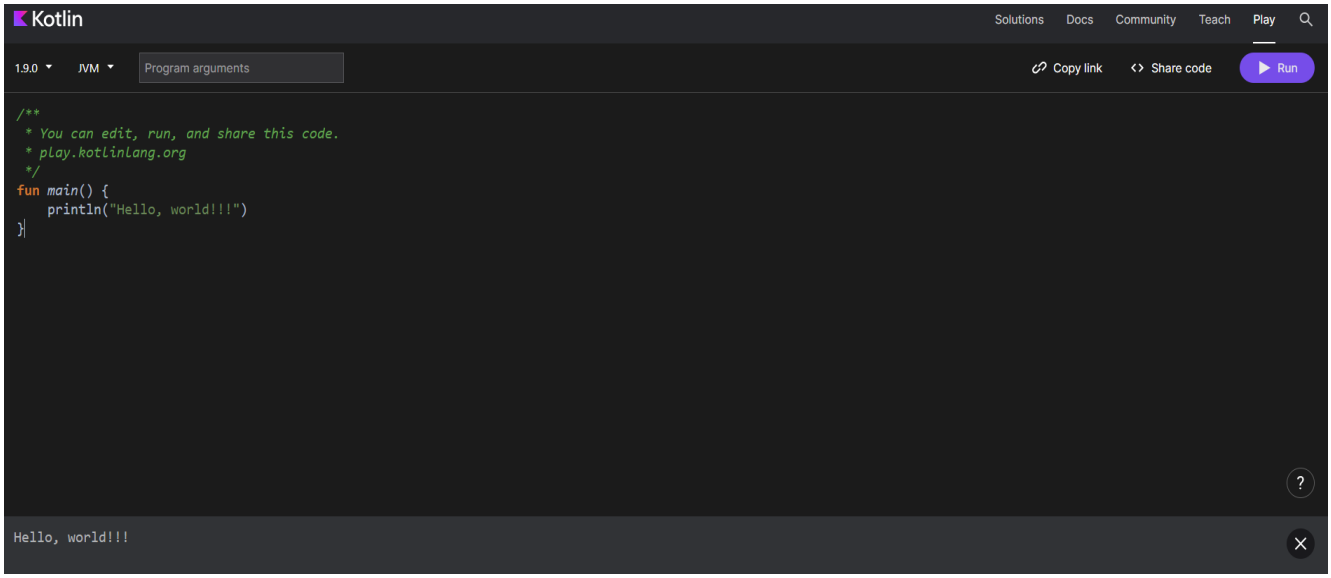
1.2 Primeiros passos

Antes de mais nada, você poderá utilizar duas IDEs para desenvolvimento em Kotlin e são elas:

- **IntelliJ:** para desenvolvimento em Kotlin (download da IDE).
- **Android Studio:** para desenvolvimento **mobile usando Kotlin** ou mesmo Java (download da IDE).

Nota: Para ambas as IDEs será necessário fazer as configurações de ambiente, **salientando que a disciplina é de desenvolvimento de aplicativos, portanto é importante que você siga os processos da documentação que se refere a configuração de ambiente do Android Studio na sua máquina**. Neste link você encontra estas especificações.

Figura 1 - Tela do Playground do Kotlin - (Kotlin Playground)



Neste início de módulo, os exemplos usados neste material serão desenvolvidos usando o **Playground do Kotlin** que você poderá acessar neste endereço: (Kotlin Playground)

1.3 Estrutura do código Kotlin

A estrutura básica de um código em Kotlin é semelhante à de outras linguagens de programação. Os códigos em Kotlin normalmente incluem declarações de funções, classes, variáveis e outras construções que definem a lógica do programa. Aqui está uma visão geral da estrutura típica de um código Kotlin:

a) **Pacotes (Packages)**: Opcionalmente, você pode definir pacotes para organizar seu código em módulos lógicos. Isso é útil para separar e agrupar classes relacionadas. A declaração do pacote é a primeira linha do arquivo, caso exista.

```
package com.example.myapp
```

b) **Imports:** Logo após a declaração do pacote (se houver), você pode adicionar importações para classes e funções externas que serão usadas no arquivo.

```
import kotlin.math.PI
import java.util.*
```

c) **Funções (Functions):** A função main() é o ponto de entrada de um programa Kotlin. Além disso, você pode definir outras funções para encapsular a lógica do programa.

```
fun main() {
    // Variáveis mutáveis
    var idade: Int = 30
    var nome: String = "Ana"
    var altura = 1.75 // Tipo de dado inferido como Double

    idade = 31
    nome = "Mafalda"

    // Variáveis imutáveis
    val pi: Double = 3.14159
    val mensagem = "Olá, mundo!"

    println("$nome tem $idade anos e mede $altura metros.")
    println("O valor de PI é $pi.")
    println(mensagem)
}
```

main define a função principal - ponto de entrada do programa. Todas as instruções dentro dessa função serão executadas

d) **Classes (Classes):** Kotlin é uma linguagem orientada a objetos, e você pode definir classes para encapsular comportamentos e estados relacionados.

```
class Pessoa(val nome: String, var idade: Int) {
    fun aniversario() {
        idade++
    }
}
```

1.4 Tipos e variáveis

Kotlin é uma linguagem que roda na Máquina Virtual Java (JVM) e é utilizada em diversos cenários, desde desenvolvimento Android até aplicações para servidores. Além disso, possui muitos recursos avançados e sintaxe simplificada que permitem escrever código de forma eficiente e segura.

1.4.1 *Os tipos no Kotlin*

Os tipos de dados são um conceito fundamental que determina o tipo de valor que uma variável ou expressão pode conter. Cada valor em um programa Kotlin tem um tipo específico, que pode ser um número inteiro, um número de ponto flutuante, um caractere, uma string, um valor booleano, entre outros.

Os tipos de dados em Kotlin são importantes porque contemplam:

- **Segurança de Tipagem:** Kotlin é uma linguagem de tipagem estática, o que significa que o tipo de cada variável é verificado em tempo de compilação. Isso ajuda a prevenir erros relacionados ao uso incorreto de variáveis, como operações inválidas ou atribuições incompatíveis.
- **Performance:** Os tipos de dados permitem que o compilador aloque o tamanho adequado de memória para cada valor. Isso torna o código mais eficiente, pois evita desperdício de memória e processamento desnecessário.

- **Semântica do Programa:** Os tipos de dados ajudam a expressar a intenção do programa de maneira clara e consistente. Eles fornecem informações sobre os dados que estão sendo manipulados, o que torna o código mais legível e compreensível.

TIPOS	EXEMPLOS
Números inteiros	**Int, Byte, Short e Long**
Números de ponto flutuante	**Float e Double**
Caracteres	**Char**
Booleano	**Boolean**
Texto	**String**
Arrays	**Array**
Coleções	**List, Set e Map**

Além desses tipos básicos, Kotlin também permite a criação de **tipos de dados personalizados**, como classes e enumerações, o que possibilita a definição de abstrações mais complexas para representar dados específicos do seu domínio.

A escolha dos tipos de dados adequados é uma prática importante para escrever código robusto, eficiente e de fácil manutenção em Kotlin.

1.4.2 *As variáveis no Kotlin*

No Kotlin, assim como em outras linguagens de programação, as variáveis são elementos fundamentais para armazenar e manipular dados em um programa. **Uma variável é uma referência a uma posição de memória que contém um valor específico. Esses valores podem ser números, texto, booleanos, objetos ou qualquer outro tipo de dado suportado pela linguagem.**

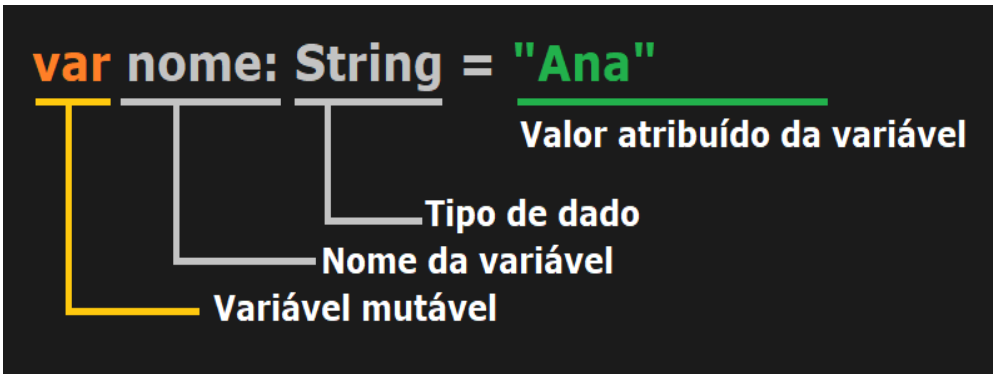
Ao declarar uma variável em Kotlin, você especifica seu nome, opcionalmente o tipo de dado e o valor inicial (se houver). O tipo de dado pode ser explicitamente fornecido ou inferido automaticamente pelo compilador com base no valor inicial atribuído.

1.4.2.1 *P*ontos importantes sobre variáveis em Kotlin

Vejamos alguns pontos importantes acerca das variáveis em Kotlin:

- **Declaração:** as variáveis são declaradas usando as palavras-chave **var ou val**. Onde **var** é usado para declarar variáveis mutáveis (seu valor pode ser alterado), enquanto **val** é usado para declarar variáveis imutáveis (constantes).
- **Tipos de Dados:** Kotlin é uma linguagem com tipagem estática, o que significa que o tipo de dado de uma variável é determinado em tempo de compilação. Os tipos de dados podem ser explícitos ou inferidos.
- **Inicialização:** as variáveis devem ser inicializadas com um valor antes de serem usadas. Para variáveis mutáveis, você pode inicializá-las mais tarde, mas para variáveis imutáveis (declaradas com val), o valor deve ser atribuído durante a declaração ou no construtor, se for uma propriedade de classe.
- **Escopo:** o escopo de uma variável determina onde ela pode ser acessada. Variáveis podem ter escopo local (disponíveis apenas dentro de um bloco ou função) ou escopo de classe (disponíveis em toda a classe).

Figura 2 - var



O que mais sobre Variáveis Mutáveis (var)?

As variáveis declaradas com **var são mutáveis**, o que significa que você pode atribuir valores diferentes a elas após sua inicialização. O uso de var é adequado quando o valor da variável precisa ser alterado ao longo do tempo.

```
// Variáveis mutáveis
var idade: Int = 30
var nome: String = "Ana"
var altura = 1.75 // Tipo de dado inferido como Double

idade = 31
nome = "Mafalda"
```

O que mais sobre Variáveis Imutáveis (val)?

As variáveis declaradas com **val são imutáveis**, ou seja, seu valor não pode ser alterado após a inicialização. Isso é útil quando você deseja garantir que o valor da variável permaneça constante durante a execução do programa.

```
// Variáveis imutáveis
val pi: Double = 3.14159
val mensagem = "Olá, mundo!"
val nome: String = "Maria"
// nome = "João" // Isso causará um erro, pois 'nome' é imutável e não pode ser reatribuído
```

Lembrando que o Kotlin também suporta a **inferência de tipo**, o que significa que você pode declarar uma variável sem especificar o tipo de dado se o compilador puder inferi-lo com base no valor inicial atribuído.

```
//Inferência de tipo
var contador = 0 // O tipo Int é inferido automaticamente
val pi = 3.14159 // O tipo Double é inferido automaticamente
```

1.5 Imprimindo valores no console do Kotlin

O comando **println** em Kotlin é usado para imprimir uma linha de texto ou o valor de uma expressão no console. Ele é uma função built-in (função integrada) na linguagem e é amplamente utilizado para depuração e exibição de informações durante a execução de um programa.

A sintaxe básica da função println é a seguinte:

```
println(valor)
```

Exemplo de código e sua saída, impressa no console do Kotlin.

Código

```
fun main() {
    val nome = "Alice"
    val idade = 30
    val kotlin = "🙂"

    println("Olá, $nome!") // Imprime "Olá, Alice!" usando interpolação de strings
    println("Idade: $idade") // Imprime "Idade: 30"

    val resultado = 10 + 20
    println("O resultado é: $resultado") // Imprime "O resultado é: 30"
     println(kotlin)
}
```

Saída

```
Olá, Alice!
Idade: 30
O resultado é: 30
🙂
```

1.6 Operadores no Kotlin

Operadores são símbolos especiais utilizados para realizar várias operações em variáveis e objetos. Kotlin suporta uma variedade ampla de operadores que facilitam o trabalho com diferentes tipos de dados e estruturas. Abaixo, estão alguns dos operadores mais comuns em Kotlin:

1.6.1 *Operadores aritméticos*

Os operadores aritméticos são usados para que possamos declarar expressões com cálculos que envolvam as quatro operações.

OPERADOR	FUNCIONALIDADE
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo - resto da divisão

1.6.2 *Operadores de atribuição*

Operadores de atribuição são símbolos utilizados para atribuir valores a variáveis. Eles permitem que você atualize ou modifique o valor de uma variável com base em um cálculo ou operação específica.

Em Kotlin, assim como em muitas outras linguagens de programação, os operadores de atribuição são uma forma concisa de realizar ações comuns de atualização de variáveis.

OPERADOR	FUNCIONALIDADE
=	Atribuição simples
+=	Atribuição com soma
-=	Atribuição com subtração
*=	Atribuição com multiplicação
/=	Atribuição com divisão
%=	Atribuição com módulo

1.6.3 *Operadores de comparação*

Comparam valores e/ou strings, são muito utilizados com estruturas condicionais.

OPERADOR	FUNCIONALIDADE
==	Igualdade
!=	Desigualdade
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual

1.6.4 *Operadores lógicos*

OPERADOR	FUNCIONALIDADE
&&	AND (E) lógico
||	OR (OU) lógico
!	NOT (NÃO) lógico

1.6.5 *Operadores de incremento e decremento*

Os operadores de incremento e decremento são utilizados para aumentar ou diminuir o valor de uma variável de forma rápida e resumida.

OPERADOR	FUNCIONALIDADE
++	Incremento em 1
--	Decremento em 1

1.6.6 *Operador elvis*

O operador Elvis (?:) é um operador específico do Kotlin, e seu objetivo principal é fornecer uma maneira concisa de tratar valores nulos em expressões condicionais.

OPERADOR	FUNCIONALIDADE
? :	Operador elvis (null safety), retorna o valor da esquerda se não for nulo, caso contrário, retorna o valor da direita.

1.6.7 *Operadores de intervalo*

O operador de intervalo em Kotlin é utilizado para criar sequências contínuas de valores entre dois pontos específicos.

OPERADOR	FUNCIONALIDADE
..	Operador de intervalo inclusivo
until	Operador de intervalo exclusivo

1.6.8 *Operadores de coleções*

Em Kotlin, os operadores de coleções são funções que permitem realizar transformações, filtragens, reduções e outras operações em coleções de elementos, como listas, conjuntos, arrays e outras estruturas de dados. Esses operadores tornam a manipulação de coleções mais expressiva, concisa e eficiente.

OPERADOR	FUNCIONALIDADE
map	Transforma uma coleção em outra aplicando uma função a cada elemento.
filter	Filtra uma coleção com base em um predicado.
reduce	Combina todos os elementos de uma coleção em um único valor.

1.7 Controle de fluxo e paradigma procedural

Em Kotlin, assim como em outras linguagens de programação, o controle de fluxo e o paradigma procedural são fundamentais para o desenvolvimento de algoritmos e lógica de programação. O Kotlin é uma linguagem de programação moderna, abreviada e segura que suporta esses conceitos de forma eficiente.

1.7.1 *Controle de fluxo em estruturas condicionais*

O controle de fluxo refere-se à maneira como o programa toma decisões e executa instruções com base em determinadas condições. Em Kotlin, você pode utilizar as seguintes estruturas de controle de fluxo:

- **Declarações if-else:**

```
fun main() {
    val idade = 18

    if (idade >= 18) {
        println("É maior de idade")
    } else {
        println("É menor de idade")
    }

}
```

Declaração da val com o valor atribuído → 18

Se (if) a idade for maior igual a 18 (esta é minha condição);

O programa vai imprimir na tela → É maior de idade;

Senão (else), o programa vai imprimir na tela → É menor de idade;

- **Expressões ternárias**

```
fun main() {
    val idade = 18
    val resultado = if (idade >= 18) "É maior de idade!" else "É menor de idade!"
    println(resultado)
}
```

Nesta expressão ternária, estamos simplificando a declaração da estrutura condicional e conseguimos definir o if-else em somente uma linha.

- **Declarações when (substitui o switch-case do Java):**

```
fun main() {
    val diaSemana = 3

    when (diaSemana) {
        1 -> println("Domingo")
        2 -> println("Segunda-feira")
        3 -> println("Terça-feira")
        // ...
        else -> println("Dia inválido")
    }
}
```

Declaração da val com valor atribuído.

Declaração do valor para comparação.

Se o valor comparado está contido numa das opções definidas, o programa irá imprimir na tela o valor correspondente, se não (else) ele irá imprimir que o dia é inválido.

1.7.2 *Paradigma Procedural em Kotlin*

O paradigma procedural é um estilo de programação que se concentra em escrever procedimentos ou funções que contêm um conjunto de instruções, para serem executadas em uma ordem específica. Em Kotlin, você pode escrever funções para dividir a lógica do programa em unidades mais gerenciáveis e reutilizáveis.

Na próxima imagem está um exemplo de como criar uma função procedural em Kotlin:

```
// Função que calcula o fatorial de um número inteiro
fun calcularFatorial(numero: Int): Int {
    if (numero == 0) {
        return 1
    } else {
        return numero * calcularFatorial(numero - 1)
    }
}

fun main() {
    val num = 5
    val resultado = calcularFatorial(num)
    println("O fatorial de $num é $resultado")
}
```

No exemplo acima, a função calcularFatorial é um procedimento que usa uma abordagem recursiva para calcular o número fatorial de um número inteiro.

O Kotlin suporta a programação procedural, mas também oferece recursos para programação orientada a objetos e funcional, permitindo que você escolha o paradigma mais adequado para cada situação.

É importante notar que, embora o paradigma procedural seja viável em Kotlin, a **linguagem é mais conhecida por seu suporte robusto aos paradigmas orientado a objetos e funcional.**

1.8 Laços de repetição

Como em toda a linguagem de programação, os laços de repetição são declarados usando os seguintes recursos: **for, while** e **do-while**.

1.8.1 *for:*

O loop for (para) é usado quando você sabe o número de iterações que deseja executar. **Ele itera sobre uma faixa de valores ou uma coleção de elementos.**

```
fun main() {

    for (i in 1..5) {
        println("Iteração $i")
    }
}
```

Na imagem acima, temos um loop for que irá iterar sobre a faixa de valores de 1 a 5 (inclusive). O operador **..** é usado para criar essa faixa.

Os itens abaixo explicam a funcionalidade do código:

- O loop começa com **i** valendo **1**, que é o primeiro valor da faixa definida.
- O bloco de código dentro do loop é executado, e a instrução **println("Iteração $i")** imprime na tela a mensagem "Iteração" seguida do valor atual de **i**.
- Após executar o bloco, o valor de **i é incrementado automaticamente para o próximo valor na faixa.**
- O loop continua executando o bloco de código para cada **valor dentro da faixa (1, 2, 3, 4, 5)**.
- Quando **i** atinge o valor **6 (o próximo valor após 5 na faixa), o loop é encerrado.**

Outro exemplo!

Repita este código no playground para ver a execução do programa!

```
fun main() {
        val lista = listOf("a", "b", "c", "d", "e")
         for (elemento in lista) {
          println("Elemento: $elemento")
    }
}
```

1.8.2 *while*

O loop while (enquanto) é usado quando você não sabe o número exato de iterações e quer que o bloco de código seja executado enquanto uma condição específica for verdadeira.

```
fun main() {
        var contador = 0
         while (contador < 5) {
         println("Iteração $contador")
         contador++
    }
}
```

Na imagem acima, **temos um loop while que irá executar o bloco de código enquanto a condição contador < 5 for verdadeira. Vamos analisar o que acontece passo a passo:**

- ✓ A variável contador é **inicializada com o valor 0.**
- ✓ O loop entra na primeira iteração e verifica a condição **contador < 5, que é verdadeira porque 0 é menor que 5.**
- ✓ O bloco de código dentro do **loop é executado, e a instrução println("Iteração $contador") imprime na tela a mensagem "Iteração" seguida do valor atual de contador, que é 0.**
- ✓ Após executar o bloco, a **variável contador é incrementada em 1 usando contador++** (operador de incremento), **tornando seu valor igual a 1**.

- ✓ **O loop verifica novamente a condição contador < 5, que é verdadeira porque agora contador é 1.**
- ✓ O bloco de código é executado novamente, **imprimindo "Iteração 1".**
- ✓ Esse processo continua até que o **contador atinja o valor 5.**
- ✓ **Quando contador for igual a 5, a condição contador < 5 será falsa, e o loop será encerrado.**

Importante!

Use este exemplo para praticar na sua IDE ou mesmo no playground do Kotlin!

```
fun main() {
    var dado: Int
    var tentativas = 0

    while (true) {
        dado = (1..6).random()
        tentativas++

        println("Tentativa $tentativas: Resultado do dado -> $dado")

        if (dado == 6) {
            println("Parabéns! Você conseguiu um 6 após $tentativas tentativas.")
            break
        }
    }
}
```

1.8.3 *do-while*

O loop do-while (faça enquanto) é semelhante ao while, **mas ele executa o bloco de código pelo menos uma vez antes de verificar a condição.**

```
fun main() {
    var x = 5
     do {
     println("x: $x")
     x--
     } while (x > 0)
}
```

Neste código, temos um loop **do-while em Kotlin**. Vamos analisar o seu funcionamento?

- ✓ **A variável x** é inicializada com o **valor 5.**
- ✓ O bloco de **código dentro do loop do é executado pela primeira vez, imprimindo "x: 5" na tela.**
- ✓ **Em seguida,** a variável x é decrementada em 1 **usando x--, e seu valor agora é 4.**
- ✓ O loop **while verifica a condição x > 0, que é verdadeira porque x é 4.**
- ✓ **Como a condição é verdadeira,** o loop do-while continua executando o bloco de código.
- ✓ **O processo se repete: o valor de x é impresso ("x: 4"), x é decrementado (agora x é 3)** e a condição é verificada novamente.
- ✓ Esse processo continua até que **x seja igual a 0. Quando x se torna 0, a condição x > 0 se torna falsa, e o loop é encerrado.**

Importante!

Use este exemplo para praticar na sua IDE ou mesmo no playground do Kotlin!

```
fun main() {
    var numero: Int
    do {
        print("Digite um número maior que 10: ")
        numero = readLine()?.toIntOrNull() ?: 0
    } while (numero <= 10)

    println("Você digitou o número $numero, que é maior que 10.")
}
```

2. REFERÊNCIAS

Múltiplos autores: **KOTLIN DOCS**. 2023. Disponível em: <https://kotlinlang.org/docs/home.html>. Acesso em: 17 de julho de 2023

Múltiplos autores: **JETBRAINS**. 2022. Disponível em: <https://www.jetbrains.com/pt-br/>. Acesso em: 18 de julho de 2023

Múltiplos autores: **DOCUMENTAÇÃO ANDROID STUDIO**. 2023. Disponível em: <https://developer.android.com/studio>. Acesso em: 18 de julho de 2023