

Introdução ao Kotlin



Estrutura da linguagem

- Não utiliza ponto e vírgula;
- Não possui operador ternário tradicional;
- Type Safe e null Safe;
- Inferência de tipo;
- E muito mais!



Aplicabilidade

- Android nativo;
- Mobile multiplataforma (KMM);
- Frontend Web com Kotlin/JS;
- Frameworks:
 - ex: [Kvision](#) e [Fritz2](#)
- Backend com target node.js do Kotlin/JS;



Kotlin: vantagens

- Segura, estruturada, menos verbosa, mais otimizada, grande relevância no mercado;
- Interoperabilidade com o Java;
- Smart casts e null safety;
- Android é Kotlin first;



Resumo

- Kotlin é a linguagem oficial para desenvolvimento Android;
- Em desenvolvimento desde 2011;
- Possui diversas vantagens como null check e type safe;
- É 100% interoperável com Java;
- Também desenvolve frontend e backend com Kotlin/Js e multiplataformas com KMM;



Kotlin + Android

- **2016:** Primeira versão estável do Kotlin é liberada;
- **2017:** Google anuncia suporte a Kotlin para desenvolvimento Android;
- **2018:** Segunda linguagem preferida dos desenvolvedores;
- **2019:** Google anuncia que Kotlin é a nova linguagem oficial para desenvolvimento Android;



Parte 3: Kotlin, sintaxe básica

Desenvolvimento
Android

Tipos de dado

- Boolean
- Char
- Byte
- Short
- Null!

```
1 toByte()
2 toShort()
3 toInt()
4 toLong()
5 toFloat()
6 toDouble()
7 toChar()
```



Tipos de dado

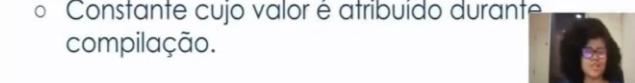
- Int
- Long
- Float
- Double
- Array

- Boolean
- Char
- Byte
- Short
- Null!

```
1 println(Int.MAX_VALUE)
2 println(Float.MAX_VALUE)
3 println(Long.MAX_VALUE)
4 println(Byte.MAX_VALUE)
5 println(Short.MAX_VALUE)
```

Declaração de variável

- **Var** (valor mutável, CamelCase):
 - Variável que pode ter seu valor alterado durante o código;
- **Val** (valor imutável, CamelCase):
 - Variável que terá somente o valor atribuído (similar ao *final* em Java);
- **Const Val** (valor imutável, SNAKE_CASE):
 - Constante cujo valor é atribuído durante compilação.



Declaração de variável

```
1 var currentAge = 22
2
3 val currentAge:Int?
4 currentAge = null ou 22
5
```

**var
(camelCase)**
valor definido e alterado durante a execução

```
1 val currentAge = 22
2
3 val currentAge:Int?
4 currentAge = null ou 22
5
```

**val
(camelCase)**
valor definido durante a execução

```
1 const val MIN_AGE = 16
2 const val MAX_AGE = 68
```

**const val
(SNAKE_CASE)**
valor definido durante a compilação

Possíveis erros

```
1 var currentAge
2 currentAge = 22 //ERRO!
```

Uma variável **não pode** ser declarada **sem tipo e sem atribuição**.

```
1 var currentYear = "Ano"
2 currentYear = 2021 //ERRO!
```

Uma variável com inferência de tipo só receberá valores do mesmo tipo que a sua primeira atribuição.

Operadores Aritméticos

Função	Expressão	Comando	Atribuição
soma	a + b	a.plus(b)	a += b
subtração	a - b	a.minus(b)	a -= b
multiplicação	a * b	a.times(b)	a *= b
divisão	a / b	a.div(b)	a /= b
resto	a % b	a.mod(b)	a %= b

- Os operadores podem ser chamados tanto como expressão quanto como comando. O resultado será o mesmo.
- A função de soma também funciona para concatenar Strings;



Operadores comparativos

Função	Expressão	Comando
maior/menor	a > b ou a < b	a.compareTo(b) > 0 ou a.compareTo(b) < 0
maior/menor igual	a >= b ou a <= b	a.compareTo(b) >= 0 ou a.compareTo(b) <= 0
Igual	a == b	a.equals(b)
diferente	a != b	!(a.equals(b))

- Os comandos **compareTo** retornam os **valores -1 (menor que), 0 (igual) ou 1 (maior)**. Já os **operadores** retornam **valores booleanos**;
- O comando **equals** retorna um **booleano**;



Operadores Lógicos

Função e expressão	Comando
E (&&)	[expressão1] and [expressão2]
Ou ()	[expressão1] or [expressão2]

- Quando utiliza-se o comando, é recomendado colocar a expressão entre parênteses;

Função e expressão
contém (in)
Não contém (in)
range/Faixa de valores (Int..Int)

- Se valor está presente em uma lista ou uma faixa (range) de valores;
- Range cria um intervalo de valores que inicia no primeiro parâmetro e acaba no segundo.

```
1 val numbers = listOf(3,9,0,1,2)
2 print(12 in numbers)
3 //false
4
5 print(12 in 0..20)
6 //true
```



Manipulação de Strings

- Strings possuem diversos métodos associados;
- indexação, concatenação, comparação, formatação;
- Pode ser concatenada com plus/+;
- Também é tratada como um array de Char;

Concatenação

```
1 val name = "Ana"
2 val s = "Olá"
3
4 println(s + name)
5 //imprime OláAna
6 println("${s}, ${name}!")
7 //Imprime Olá, Ana!
8 println("Bem vinda(o), $name!")
9 //imprime Bem vinda(o), Ana!
```

- Para concatenar duas strings o plus/+ pode ser utilizado;
- Para concatenar uma variável a uma String, os símbolos \${} devem ser inseridos;



Empty X Blank

- Métodos de comparação;
- String está vazia, em branco ou é nula ?

```
val s = ""
println(s.isEmpty())
//true
println(s.isBlank())
//true
println(s.isNullOrEmpty() || s.isNullOrEmpty())
//true
```

```
val s = " "
println(s.isEmpty())
//false
println(s.isBlank())
//true
```

Se o tamanho da string (s.length) for 0 está empty e Blank;
Se o tamanho for > 0 mas todos os caracteres são espaços em branco, está blank mas não empty;

Funções ordem superior

- Recebem outra função ou lambda por parâmetro;
- Bastante úteis para a generalização de funções e tratamento de erros;

```
1 val x = calculate(12,4,::sum)
2 val y = calculate(12,4){a,b -> a*b }
```

Indexação

- String como array;
- First(), last(), String.length, String[index];

```
1 val s = "Olá, mundo!"
2
3 println(s[0])
4 println(s.first())
5 //imprime O
6
7 println(s[s.length-1])
8 println(s.last())
9 //imprime !
```



Formatação

Nome	função	Métodos
Capitalização de strings	Alterar a formatação entre letras minúsculas e maiúscula	capitalize(), toUpperCase(), toLowerCase(), and decapitalize()
Remoção de espaços	Remove espaços vazios e caracteres inadequados para impressão	trimEnd(), trimStart(), trim()
Substituição de caracteres	Substituir caracteres por outros	replace(x, y)
formatação	Formatar outros valores para um padrão de string	"padrão \${valor}".format(val)



Funções

- Prefixo **Fun nomeDaFunção(nome:Tipo):TipoRetorno{}**
- Funções anônimas, single-line, inline, extensões, Lambdas, ordem superior;

```
1 private fun getfullname(name:String, lastName:String):String{
2     val fullname = "$name $lastName"
3     return fullname
4 }
5
6 private fun getfullname(name:String, lastName:String):String{
7     return "$name $lastName"
8 }
9
10 private fun getfullname(name:String, lastName:String) = "$name $lastName"
```

simplificando uma função



```
fun main() {
    val x:Int
    val z:Int
    z = calculate(34, 90, ::sum)
    println(z)
}
fun sum(a1:Int,a2:Int) = a1.plus(a2)

fun calculate{n1:Int,n2:Int, operation(Int,Int)->Int}:Int{
    cal result = operation(n1, n2)
    return result
}}
```

Funções single-line

- Prefixo **Fun nomeDaFuncao(nome:Tipo) = retorno;**
- Função de uma única linha;
- Infere o tipo de retorno;

```
10 private fun getFullname(name:String, lastName:String) = "$name $lastName"
```

```
1 private fun getFullname(name:String, lastName:String) =  
2     "$name $lastName"
```

Estruturas de controle

- if/else, when, elvis operator;
- Pode ser utilizado tanto para controle quanto para atribuição;
- Pode ser encadeado com múltiplas estruturas;

```
1 if(expressão){  
2     //bloco de código  
3 } else if (expressão2){  
4     //bloco de código  
5 } else{  
6     //bloco de código  
7 }
```

```
1 when {  
2     a > b -> {}  
3     a != b && a > c -> {}  
4     b == 0 -> {}  
5     else -> {}  
6 }
```

```
1 val a:Int? = null  
2 var number = a ?: 0
```

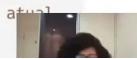


When

- Equivalente ao switch de outras linguagens;
- Aceita tanto valores quanto condicionais;
- Aceita range como case;

```
1 when {  
2     a > b -> {}  
3     a != b && a > c -> {}  
4     b == 0 -> {}  
5     else -> {}  
6 }
```

```
1 when(year) {  
2     -4000.. 475 -> //Antiguidade  
3     476..1452 -> //Medieval  
4     1453..1789 -> //Moderna  
5     currentYear -> //ano atual  
6 }
```



```
//função principal  
fun main() {  
    val grade = (0..10).random()  
    println(grade.getStudentStatus())  
}  
  
fun Int.getStudentStatus():String{  
    println("nota $this")  
    return when(this){  
        in 0..4 -> "Reprovado"  
        in 5..7 -> "mediano"  
        in 8..9-> "Bom"  
        10 -> "Excelente"  
        else -> "Indefinido"  
    }  
}
```

Funções/extensões

- Prefixo **Fun Tipo.nomeDaFuncao();**
- Cria uma função que só pode ser chamada por um tipo específico, cujo o valor pode ser referenciado dentro da função através da palavra **this**;

```
1 fun String.randomCapitalizedLetter() =  
2     this[0..this.length-1].random().toUpperCase()
```

Atribuição

```
1 val maxValue = if (a > b) a else if (a < b) b else b  
2  
3 val minValue = if (a > b){  
4     println("$b é o menor valor")  
5     b  
6 }else if(a < b){  
7     println("$a é o menor valor")  
8     a  
9 }else{  
10     println("os valores são iguais")  
11     b  
12 }
```

- O valor atribuído tem que estar na última linha do bloco;
- A condicional pode não usar chaves se só fizer a atribuição



Elvis Operator

- O mais próximo que a linguagem possui de um operador ternário;
Verifica se um valor é nulo e apresenta uma opção caso seja;
Pode ser encadeado;

```
val a:Int? = null  
val c:Int? = 9  
var number = a?: b?: 0
```

Nesse caso, se o valor de **a** não for nulo, **number** recebe **a**. Se o valor de **a** for nulo e **b** não for nulo, **number** recebe **b**. Se **a** e **b** forem nulos, **number** recebe **0**.

```
File: 2  
fun main() {  
    //val grade =(0..10).random()  
    //println(grade.getStudentStatus())  
    var t:Int  
    var x:Int? = null  
    var w:Int = 10  
    t = x?:w ?: -1  
    println(t)  
}
```

Repetição

- While, do..while, for e forEach;
- Estruturas similares às convencionais em outras linguagens;
- Aceita os comandos **in**, **range**, **until**, **downTo** e **step**

```
while(Condição){  
    do{  
        //bloco  
    } while(Condição)
```

```
for(i in 0..20 step 2){  
    println(i)
```

```
File: 2  
fun main() {  
    var x:Int? = 8  
    var w:Int? = null  
    var t:String? = x?.getStudentStatus()  
    println(t)  
}  
  
fun Int.getStudentStatus():String{  
    println("nota $this")  
    return when(this){  
        in 0..4 -> "Reprovado"  
        in 5..7 -> "Mediano"  
        in 8..9 -> "Bom"  
        10 -> "Excelente"  
        else -> "Indefinido"  
    }  
}
```

For

- **for**(varivelIndexadora **in/until/downTo** faixa de valores/condicional **step** intervalo)
 - **In**: conta do valor inicial até o valor final estabelecido
 - **Until**: conta do valor atual até o valor estabelecido menos 1;
 - **DownTo**: conta de maneira decrescente;
 - **Step**: determina o intervalo da contagem;



For

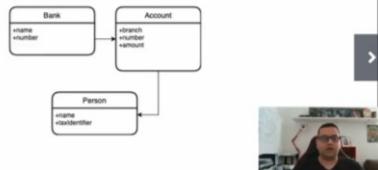
```
for(i in 0..20 step 2){  
    println(i)  
}  
  
<  
  
for(i in 10 downTo 0){  
    println(i)  
}  
  
for(i in 0 until 10){  
    println(i)  
}
```

```
1 var text = "Kotlin"  
2  
3 for (letter in text) {  
4     println(letter)  
5 }
```

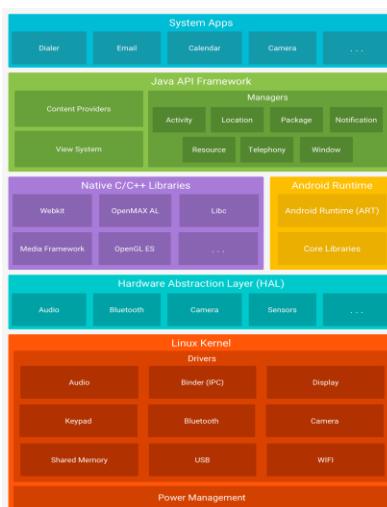
Fundamentos de Orientação a Objetos com Kotlin

Projeto Prático

✓ Aplicação DigiOneBank



Introdução a arquitetura da plataforma Android com Kotlin



[Link para o texto:](#)

<https://developer.android.com/guide/platform?hl=pt-br>

- ✓ Aplicar comunicação segura
- ◀ ✓ Oferecer as permissões corretas
- ✓ Armazenar dados de forma segura
- ✓ Manter os serviços e as dependências atualizados

Segurança com Dados

- ✓ Trabalhe com dados de forma mais segura
- ◀ ✓ Trabalhar com dados de forma mais segura em outras versões do Android

Como se proteger contra ameaças de segurança com SafetyNet

- ✓ Proteger-se contra ameaças de segurança com SafetyNet

Criptografia

- ✓ Especificar um provedor apenas com o sistema Android Keystore
- ◀ ✓ Escolher um algoritmo recomendado
- ✓ Complexidades de implementação
- ✓ Funcionalidades suspensas

Autentique usuários e chaves com biometria

- ✓ Mostrar caixa de diálogo de autenticação biométrica

Resolver problemas encontrados pelo Google Play

- ◀ ✓ Programa App Security Improvement
- ✓ Como funciona?

Aplicações Potencialmente Nocivas (PHAs)

- ✓ Potencialmente nocivo?
- ✓ PHAs desejados pelo usuário
- ✓ Classificações

Seja o primeiro a saber

- ✓ Iniciando um programa de divulgação de vulnerabilidade (**Starting A Vulnerability Disclosure Program**)

<https://developers.google.com/android/play-protect/start-a-vdp>

Exercício final

Ler a documentação do segurança:

<https://developer.android.com/security>

o que é o Android App Bundle

Junte-se a milhares de desenvolvedores e mude ou comece agora



"É um formato de publicação que inclui todo o código e recursos compilados do seu aplicativo e adia a geração e assinatura de APK no Google Play."

Etapas para começar a criar Android App Bundles

- ✓ Iniciar
- ✓ Teste seu pacote de aplicativos
- ✓ Baixe os módulos
- ✓ Uma observação sobre Instant Apps
- ✓ Restrição de tamanho de download compactado
- ✓ Problemas conhecidos
- ✓ O formato Android App Bundle
- ✓ Visão geral dos APKs divididos
- ✓ Recursos adicionais

Etapas para começar a criar Android App Bundles

Iniciar:

- ✓ Baixe o Android Studio 3.2 ou superior - Adicione suporte para Play Feature Delivery
- ✓ Crie um Android App Bundle
- ✓ Teste seu Android App Bundle usando-
- ✓ Inscreva-se no aplicativo Play App Signing
- ✓ Publique seu pacote de aplicativos no Google Play

Etapas para começar a criar Android App Bundles

Baixe os módulos de recursos com a Play Core Library

Etapas para começar a criar Android App Bundles

Teste seu pacote de aplicativos:

- ✓ Teste seu Android App Bundle localmente usando bundletool
- ✓ Compartilhe seu aplicativo com um URL
- ✓ Configure um teste aberto, fechado ou interno

Exercício final

Ler a documentação do Android App Bundle:

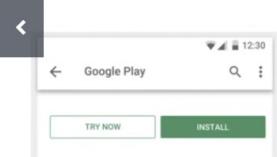
<https://developer.android.com/platform/technology/app-bundle>

Visão geral do Google Play Instant

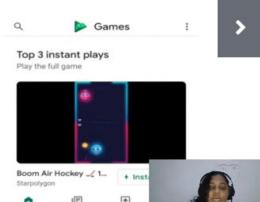
- ✓ O Google Play Instant permite que aplicativos e jogos nativos sejam iniciados em dispositivos com Android 5.0 (API de nível 21)

Como funciona a experiência instantânea

Experiência de "teste" instantânea na Play Store



Experiência de jogo completa "Instant play" no app Play Games



Permita experiências instantâneas reduzindo app

Benefícios, incluindo:

- ✓ Maior envolvimento do usuário ou instalações e sucesso nos negócios
- ✓ Ativando todas as superfícies instantâneas, incluindo o botão "Experimente agora na Play Store"
- ✓ Página inicial "Instant play" apresentada no aplicativo Google Play Games
- ✓ Publique seu pacote de aplicativos no Google Play

Link: <https://developer.android.com/topic/google-play-es/reduce-module-size>



Algumas Considerações

Criar links de aplicativo para sua experiência instantânea:

Link:

<https://developer.android.com/training/app-links/instant-app-links>

Saber mais e Recursos adicionais

Saber mais:

- ✓ [Lista de problemas conhecidos](#)
- ✓ [Componente StackOverflow para aplicativos instantâneos](#)
- ✓ [Histórias de desenvolvedores](#)

Recursos adicionais

Treinamento

- ✓ Google I / O 2018: [o futuro dos aplicativos no Android e no Google Play: modular, instantâneo e dinâmico](#)
- ✓ Google I / O 2018: [como os desenvolvedores de jogos estão proporcionando sucesso](#)



Exercício final

Ler a documentação do Google Play Instant:

<https://developer.android.com/security/instant-apps>

Aplicando conceitos de Coleções, Arrays e Listas

PowerPoint | titula_kotlin_other_collections | Saved to Dropbox | Jether Rodrigues Nascentes

Collections Mutáveis

Kotlin.collection

```

    graph TD
        Iterable["Iterable<out T>"]
        Collection["Collection<out T>"]
        Set["Set<out T>"]
        List["List<out T>"]
        MutableList["MutableList<T>"]
        MutableCollection["MutableCollection<T>"]
        MutableIterable["MutableIterable<T>"]
        MutableSet["MutableSet<T>"]

        Iterable --> Collection
        Collection --> Set
        Collection --> List
        Set --> MutableList
        MutableList --> MutableCollection
        MutableCollection --> MutableIterable
        MutableIterable --> MutableSet
    
```

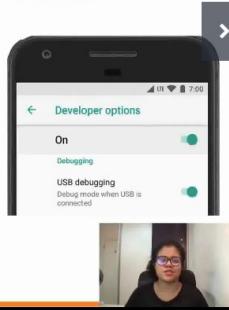
`java.util.ArrayList<T>` `java.util.HashSet<T>`

Melhores práticas para debugging, tratamento de erros e exceções

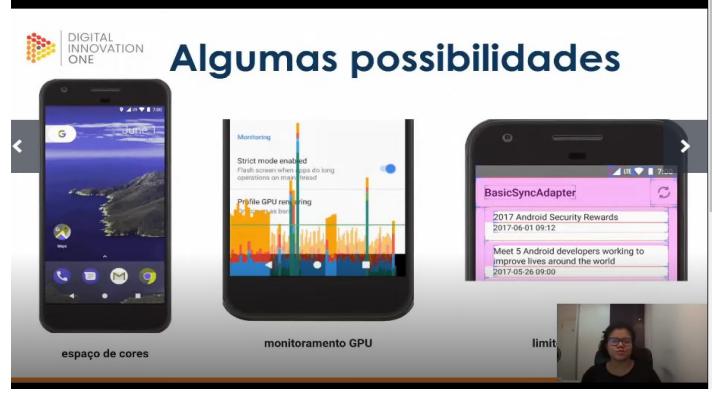


modo desenvolvedor

- Aparelho de "testes";
- Simular configurações diversas do aparelho;
- Debug por USB ou Wifi;
- Funcionalidades de depuração, redes, entrada, desenho e monitoramento;

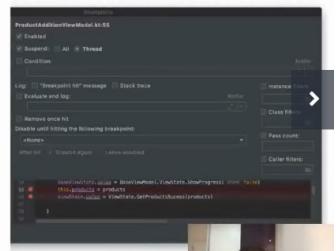


Algumas possibilidades



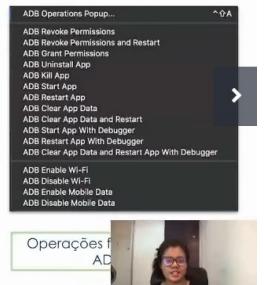
Breakpoint

- Pausa a execução em determinado método ou linha de código;
- Tela de configuração de debug;
- Pausa com condicionais;



ADB

- Android debug bridge;
- Operações que facilitam a manipulação da aplicação no modo debug;
- Pode ser chamada pelo terminal mas possui plugin com principais funcionalidades (ADB Idea);



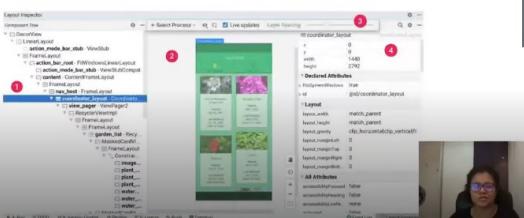
Aba Debug

- Inspeção de variáveis;
- Alteração de valor de variável;
- Step over, step into, resume;
- Abre automaticamente quando pausa em um breakpoint;



Layout inspector

- Permite a depuração dos elementos do layout que estão sendo exibidos na tela;



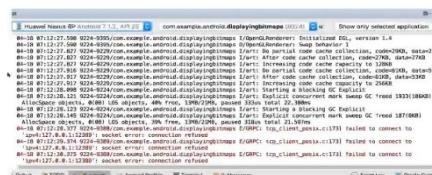
Logcat

- Tipos de log da aplicação;
- Criação de tags;
- Emissão dinâmica de logs;
- Omissão de logs irrelevantes;
- Contextualização para crashes e exceptions;



Estrutura de um Log

- Extraíndo a informação relevante;
- Apontando para o erro dentro do código;



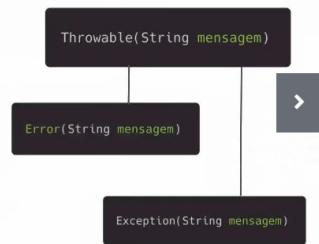
Tipos de log

- DEBUG (Log.d);
- ERROR (Log.e);
- INFO (Log.i);
- VERBOSE (Log.v);
- WARNING (Log.w);



Throwable

- Superclasse de todos os **Eros e Exceptions**;
- **Eros** - podem ocorrer tanto durante a execução quanto na compilação;
- Exception - geralmente quebram (causam crash) na aplicação;



Exceptions

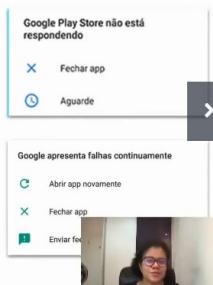
- Subclasse de Throwable;
- Ex:
 - **ActivityNotFoundException**;
 - **NullPointerException**;
 - **IndexOutOfBoundsException**;
 - **ClassCastException**;
 - e muitas outras..
- Para mais informações: [documentação Exceptions](#)

Tratando Exceptions

- Null check;
- Retorno visual do erro;
- Criação de um **ExceptionHandler**;

ANRs

- Aplicação não respondendo;
- Falta de retorno a interação do usuário;
- Bloqueio da UI thread;
- Dialog que permite forçar o fechamento da aplicação;



Como evitar Exceptions

- Mapeamento das exceptions no try/catch;
- Prevenir operações indevidas;
 - (operações durante iteração, casts inválidos, tipos nuláveis sem tratamento e etc)
- Testes unitários;

Projetando o primeiro aplicativo Android usando Kotlin

Manifest

- Informações essenciais para as ferramentas de compilação do Android, para o SO Android e para a Google Play;
- Recebe a declaração de todas as activities, permissões do app etc;
- Arquivo gerado e escrito em XML;
- Declaração por tags;



Activity

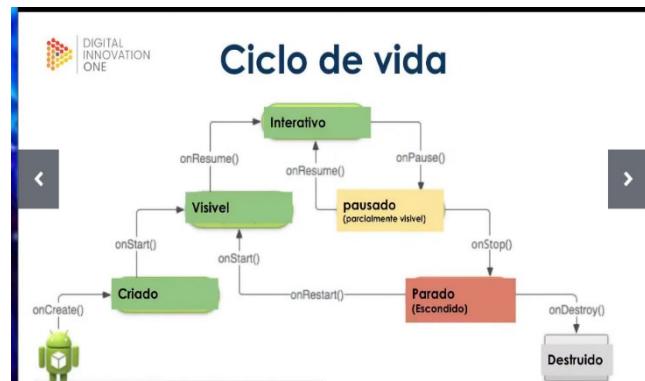
- Um dos componentes principais de um projeto Android;
- Utiliza instâncias dos métodos callback de ciclos de vida (em vez de uma única função main);
- Pode corresponder a uma ou mais telas;

Ciclo de vida

- Informa: criação, interrupção ou retomada de uma tela ou da destruição do processo pelo sistema;
- **onCreate(), onStart(), onResume(), onPause(), onStop() e onDestroy()**;
- Ciclo visível X ciclo invisível;
- Equivalente a várias funções "main()"

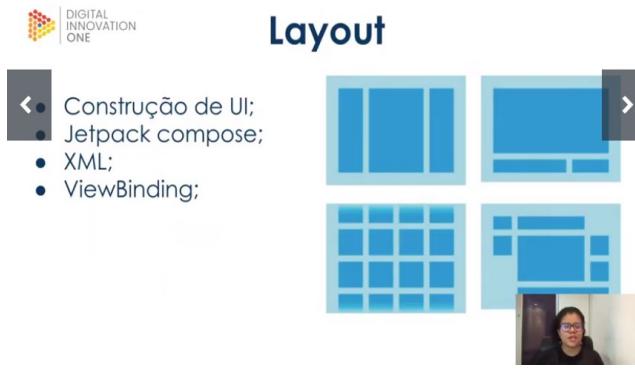
Activity

- Toda activity deve ser "registrada" no manifest;
- O layout da activity define o que será exibido para o usuário;
- Gerencia os ciclos de vida, interações com usuário, abertura de outras telas e aplicações e etc;



Gradle

- Configuração e automatização de build/compilação;
- build.gradle arquivo de configuração;
- Gerencia importações, tipos de build, versão e etc;
- Escrito em Groovy ou Kotlin DSL;



Views

- Todo componente é uma View;
- Toda View é clicável;
- Tag <nomeComponente>;
- id, layout_width, layout_height;
- Views herdam métodos, aparência, estilo e atributos de componentes (botões, imagens, texto e etc) ou podem ser criados de maneira customizada;

Atributos de Views

- Gravity X Layout_gravity;
- margin X padding;
- InputType e personalização de texto;
- Match_parent, wrap_content, dp, sp;
- Visibility, backgroundColor, tint e etc e etc;
- [Sumário dos atributos](#);

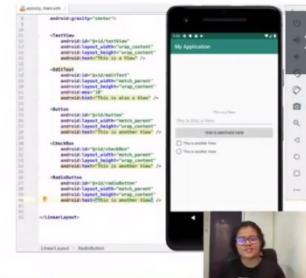
Layout Types

- Relative Layout:**
 - A posição dos itens é relativa a outros itens ou ao layout pai;
 - Permite sobreposição de elementos;
 - Elimina a necessidade de layouts encadeados;
 - Exige a criação de hierarquias complexas;

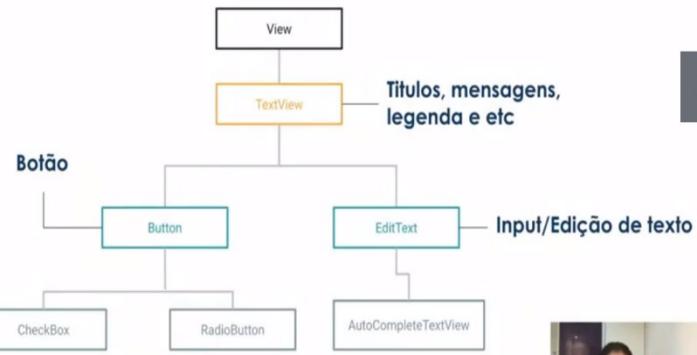
To_end_of:
To_start_of:
above:
below:
align_parent:

Layout XML

- Tipos de layout;
- Parent;
- Componentes;
- Medidas de altura e largura;
- Personalização de atributos;



Exemplos de views



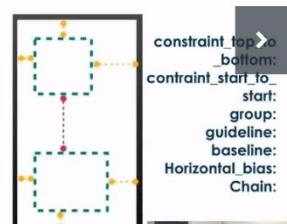
Layout Types

- LinearLayout:**
 - Itens alinhados uns após os outros **verticalmente** ou **horizontalmente**;
 - Elementos não serão sobrepostos;
 - Orientação vertical ou horizontal;



Layout Types

- Constraint Layout:**
 - Baseado em restringir os componentes aos limites do layout ou outros componentes;
 - Focado em otimização;
 - Visa substituir o Linear e Relative no desenvolvimento android;



Outros Layouts Type

- **Absolute Layout;**
 - Posição absoluta x/y;
- **Table Layout;**
 - Formato de tabela;
- **Grid Layout;**
 - Formato "grade" colunas X linhas;
- **Frame Layout;**
 - "moldura", ideal para um único item;



primeiro aplicativo Android usando Kotlin UI

```

Analyze Refactor Build Run Tools VCS Window Help
MainActivity.java MainActivity.kt [imc-app.apk]
activity_main.xml activity_main.xml main_activity.xml main_main.xml styles.xml
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setListeners()
    }

    private fun setListeners() {
        alturaEDT.setOnTextChanged { text ->
            // Toast.makeText(this, text.toString(), Toast.LENGTH_SHORT).show()
        }
        pesoEDT?.onTextChanged { text, _, _, _ ->
            // titleTXT.text = text
            calcularIMC(pesoEDT.text.toString(), alturaEDT.text.toString())
        }
    }

    private fun calcularIMC(peso: String, altura: String) {
        val peso = peso.toFloatOrNull()
        val altura = altura.toFloatOrNull()
        if (peso != null && altura != null) {
            val imc = peso / (altura * altura)
            titleTXT.text = String.format("%.2f", imc)
        }
    }
}

```

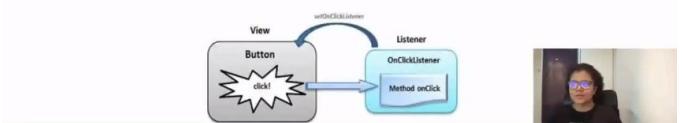
Tratando Inputs de Texto

- **TextWatchers:**
 - Callbacks para o fluxo do input
 - Antes do texto mudar, quando o texto muda, após o texto mudar;
- Métodos:
 - **beforeTextChanged(CharSequence s, int start, int count, int after),**
 - **onTextChanged(CharSequence s, int start, int before, int count),**
 - **afterTextChanged(Editable s)**



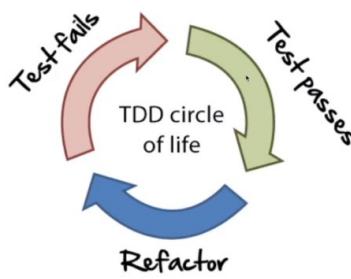
Tratando Cliques

- Toda view possui um método: **OnClickListener {}**
- View → (click do usuário) → OnClickListener
 - Os listeners geralmente são declarados no onCreate;
 - Executam uma ação, pegam algum dado e retornam algum feedback visual para o usuário;



Aplicando TDD e padrões de testes no desenvolvimento de aplicativos Android

TDD



Exibindo resultados

- Atribuir valores a campos de texto, botão e etc;
- Exibir warning dialog;
 - Exibir Toast;
 - Abrir uma nova tela;
- O que você quiser!

Criando um Toast!!!!



Desenvolvimento integrado de aplicações Android



Content Provider

Um Content Provider é um provedor de conteúdo, que permite compartilhar informações de uma aplicação com outras aplicações.



Content Provider

- Permitir compartilhar dados de forma padronizada.
 - Para alterar a forma de persistir os dados, é preciso alterar o provider apenas.
- É o mecanismo-padrão de compartilhamento de dados da aplicação com outros componentes do sistema e com outras aplicações que estejam no mesmo dispositivo.



Working with Notifications

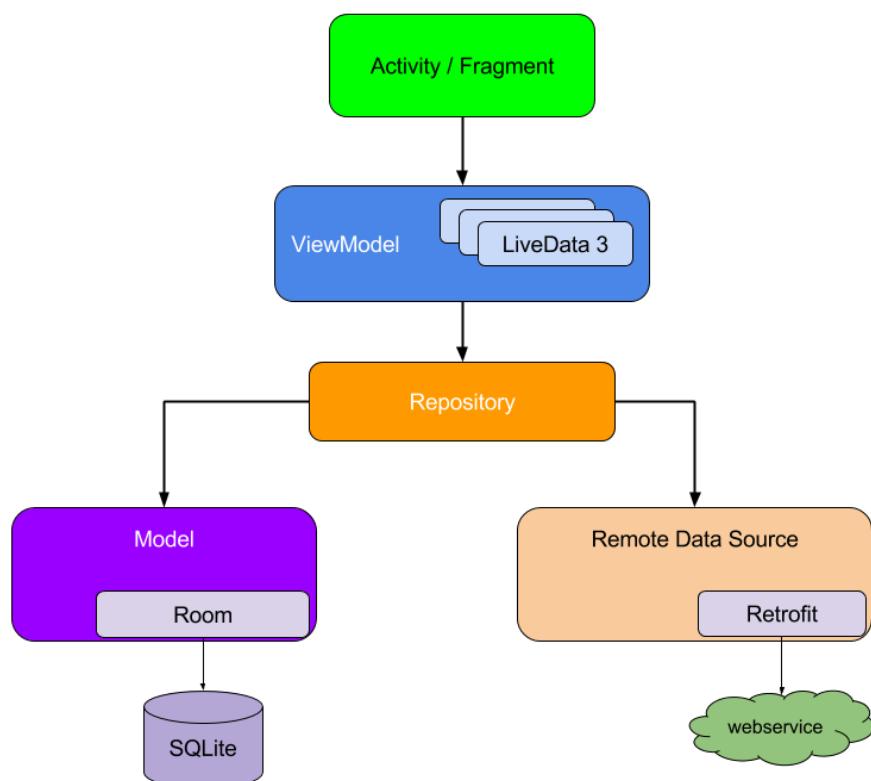
Uma notificação é uma mensagem que Android exibe fora da IU do app, para fornecer ao usuário lembretes, comunicados de outras pessoas ou outras informações oportunas do seu app. Os usuários podem tocar na notificação para abrir seu app ou executar uma ação diretamente da notificação.



Working with API

As API são conjuntos de **instruções e padrões de programação** que servem para fornecer dados e informações relevantes de uma determinada aplicação.

Princípios do desenvolvimento Kotlin com Clean Architecture e MVVM



MVVM

View: Camada onde teremos as partes relacionadas a componentes visuais (Activities/Fragments) e lógica relacionada a estes componentes.

ViewModel: Responsável por ter a lógica de negócios (O viewmodel NÃO deve "conhecer" os componentes de UI).

Repository: Manipula operações de dados com banco de dados e/ou webservice.

**MVVM**

LiveData: Armazena dados observáveis.

MVVM

Data source: É a camada que o repositório irá usar para acessar as fontes de dados locais e/ou web

Fonte: <https://developer.android.com/jetpack/guide?hl=pt-br>

Use case e implementações

Domain - Modelos e regras de negócio.

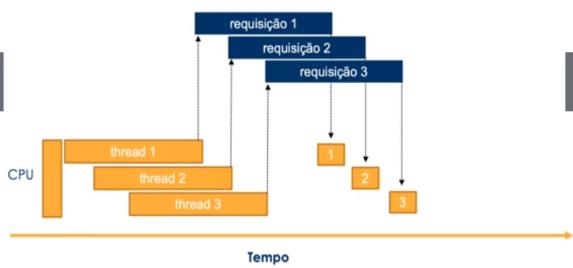
Data - Abstração para acessar o datasource.

Use cases - Transmite as ações do usuário.

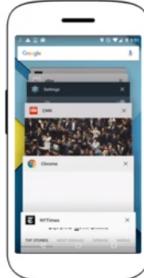
App - Irá conter as implementações das interfaces da camada de dados



Trabalhando com processamento assíncrono no Android

Definição de thread**Visão geral dos processos e threads**

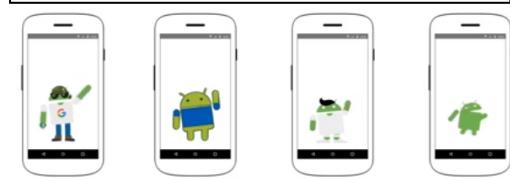
O Android vai lançar uma exceção se nós tentarmos acessar a rede através da Thread Principal!

Multitarefas no Android

Ex.: Ouvir música em segundo plano enquanto verifica os apps recentes

**Modelo de threads no Android**

Podendo rodar em núcleos diferentes

**Funcionamento da thread principal**

60FPS < 17MS
run time computação

**Android ANRs (application not responding)**

Após 5 segs sugere encerrar o app



Executar a tarefa de rede em um thread de execução secundário.

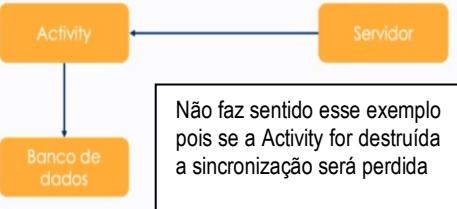


Fazer alterações na interface de usuário.

Parte 2: Performing Background Work with Services

Bootcamp Kotlin Everis

Introdução ao Service



Introdução ao Service



Service e ciclo de vida



O Service é um dos quatro componentes primários descrito no Adroid Manifest.

A diferença principal entre Sevice e a Acitivity é o ciclo de vida dele que não está ligado ao IUI.

Ele funciona o tempo todo no Android de várias maneiras.

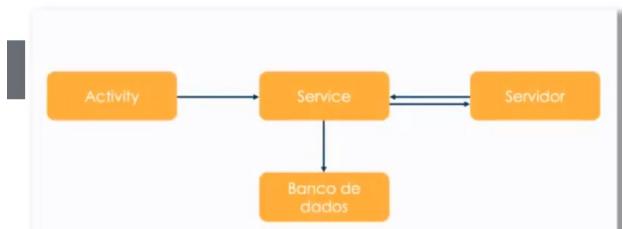
São três de três tipos:

1. Primeiro Plano - interage com os usuários (usados para tocar música, baixar arquivos ou usar o GPS), por ter prioridade extra o android não encerra a service mesmo com problema de memória;
2. Segundo Plano – são services que não são perceptíveis aos usuários (Geralmente tarefas de longa duração), podendo o android interromper a servisse por demanda de memória;
3. Service Vinculado – a função primaria do servisse é criar uma API para ser usado por outros aplicativos utilizarem. O Seu aplicativo inicia e os dois se relacionam através da comunicação entre processos. O SERVIÇO VINCULADO oferece uma interface cliente/servidor que permite os componentes interajam com a interface (enviem e recebam resultados).

Service no Android Framework



Service de segundo plano





Padrões e convenções profissionais de codificação com Kotlin

// Aula 1 – programação em Kotlin

```
fun main() {  
    var idade = 11  
    idade = 37  
    val nome = "Diego"  
    println("Eu sou $nome e tenho $idade anos")  
}
```

//Aula 2 - Operadores

```
fun main() {  
    var saldo = 1000  
    println("Valor do saldo é $saldo")  
}
```

//Aula 3 – Condicionais : IF / ELSE

```
fun main() {  
    val idade = 69  
    if (idade >= 18 && idade <= 70 ){  
        println("Pode dirigir")  
    } else{  
        println(" Não pode dirigir")  
    }  
}
```

//Aula 4 - When

```
fun main() {  
    val cor = "Azul"  
  
    var mensagem = when (cor) {  
        "vermelho" -> "A cor do cavalo de Napoleão não é vermelho e sim branco."  
        "Azul" -> "A cor do cavalo de Napoleão não é azul e sim branco."  
        "Branco" -> "Acertou. A cor do cavalo de Napoleão é branco."  
        else -> "sem cor."  
    }  
    println(mensagem)  
}
```

//Aula 5 – Variaveis com valores nulos

```
fun main() {  
    //sem a interrogação após o tipo, iria dar erro o operador ?: torna aceitável esta variável  
    val mensagem: String? = null  
  
    //O operador !! – é usado quando se garante que a variável ou expressão não é nula  
    //o operador ?: (ou elvis) faz o seguinte - caso seja nulo o valor da esquerda e alterá-lo pelo da direita  
    println("${mensagem ?: "Mensagem genérica."}")  
  
    //mensagem = "mensagem correta"  
    //println("$mensagem")  
  
    /*if(mensagem != null) {  
        println("$mensagem")  
    } else {  
        println("Mensagem genérica.")  
    }*/  
}
```

//Aula 6 – Arrays e Listas

```
fun main() {  
    //O mutableListOf permite fazer alterações na lista tanto adiconar como remover  
    val listaDeFrutas = mutableListOf("Banana", "Laranja")  
  
    //Conforme abaixo aparecerá com o .size a quantidade de elementos do array  
    //println("${listaDeFrutas.size}")  
    //Para imprimir um determinado item coloca-se entre colchetes a posição desejada  
    //Sendo que todo array começa pela posição 0  
    //println("${listaDeFrutas[0]}")  
  
    // o comando .add para adicionar um novo item  
    listaDeFrutas.add("Limão")  
    //Para adicionar o item em uma determinada posição, basta colocar a posição e após a vírgula o item  
    //que será adicionado na lista  
    listaDeFrutas.add(0,"Tomate")  
    //Para remover um item pode colocar o nome do item:  
    //listaDeFrutas.remove("Tomate")  
    //ou ou a posição usando o removeAt:  
    listaDeFrutas.removeAt(0)  
    println("${listaDeFrutas}")  
  
    //Para adicionar um array em outro array:  
    val listaDeFrutasDaModa = listOf("Açaí", "Amora")  
  
    listaDeFrutas.addAll(listaDeFrutasDaModa)  
    println("Juntando Array ${listaDeFrutas}")  
  
    //Tentando mexer no array não mutável  
    //listaDeFrutasDaModa.add("Manga")  
    //println("Tentando adicionar novos elemento no array listOf ${listaDeFrutasDaModa}")
```

```
//Neste caso não dá certo pois list são imutáveis e nesse exemplo apenas poderia ver os itens da lista
```

```
//Com o ArrayList é possível manipular os dados
```

```
//Já com a lista só podemos consultar os dados
```

```
val numeros = arrayListOf<Int>(10, 20, 30)
```

```
numeros.add(40)
```

```
println("$numeros")
```

```
}
```

//Aula 7 – FOR e WHILE

```
fun main() {
```

```
    var numeros = arrayListOf<Int>(10, 20, 30, 40, 50, 60)
```

```
    for (numero in numeros) {
```

```
        //println("$numero")
```

```
}
```

```
//Aqui disse o tamanho do array que será percorrido e mostrado no resultado
```

```
for (index in 2..3) {
```

```
    //println("${numeros[index]}")
```

```
}
```

```
var count = 0
```

```
while (count < 3) {
```

```
    println("${numeros[count]}")
```

```
    count++
```

```
}
```

```
do {
```

```
    println("Agora com Do while: ${numeros[count]}")
```

```
    count++
```

```
} while (count < 3)
```

```
}
```