[MB-KT-004]

Técnicas de Programação (kotlin)

Técnica de Programação - Kotlin

#1	Introdução à Programação Funcional: **Porque utilizar Programação Funcional?**
#2	Entendimento sobre corrotinas e como implementar (Parte 1): **Trabalhando com fluxo de dados**
#3	Entendimento sobre corrotinas e como implementar (Parte 2): **Trabalhando com fluxo de dados assíncronos**
#4	Extension: **Entendendo funções de extensão**
#5	Regex: **Manipulando dados com regex**

Técnica de Programação - Kotlin

#6	Boas práticas em Kotlin (Parte 1): **Criando código convencional**
#7	Boas práticas em Kotlin (Parte 2): **Criando código convencional**
#8	Revisão de tudo que foi visto nas aulas anteriores
#9	Aula dedicada à devolutiva da avaliação por rubrica / autoavaliação.

Vamos falar sobre Corrotina ?

Vamos falar sobre Corrotina ?

CO-RROTINA
CO
ROTINA

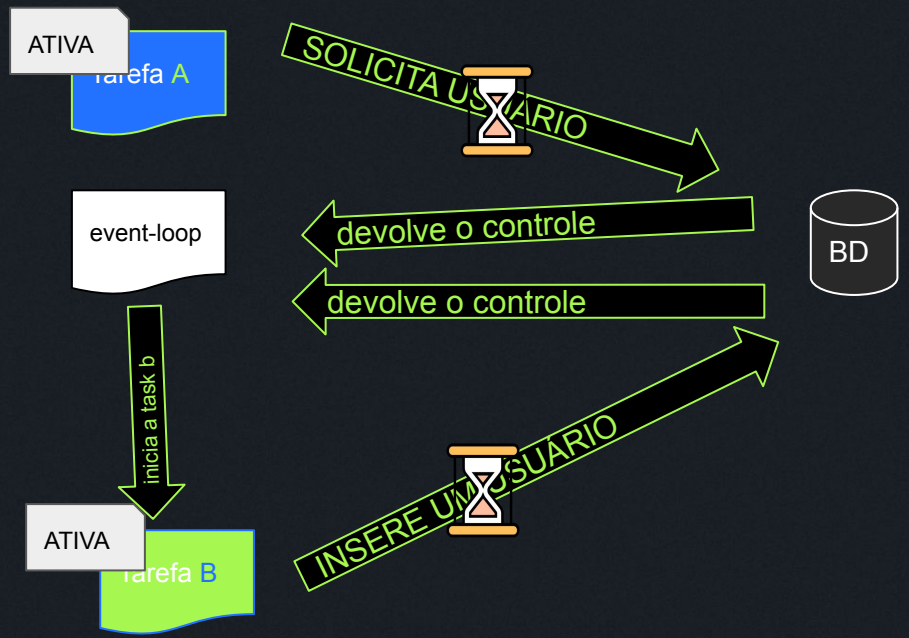
O que vem à sua cabeça quando se ouve o termo CORROTINA?

Vamos falar sobre Corrotina ?

Mas antes, precisamos entender alguns conceitos importantes . . .

Programação Concorrente

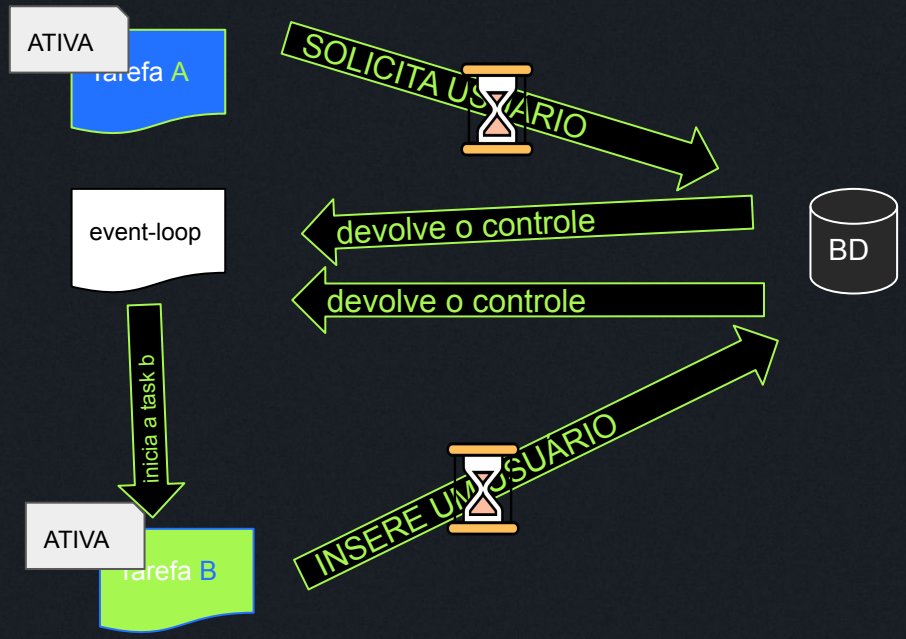
Define um conceito onde diferentes tarefas estão ativas ao mesmo tempo e, muitas vezes, compartilhando os mesmos recursos, mas não necessariamente executando ao mesmo tempo.



As tarefas só continuam após o banco responder, e nesse caso, o banco está sofrendo pela concorrência da leitura e da escrita. Por isso esse código é concorrente, as tarefas concorrem por um recurso, mesmo executando em momentos distintos.

ada

Programação Concorrente



Normalmente, concorrer pelo acesso não tem problema, mas quando ocorre uma escrita por uma das tarefas, uma delas pode falhar ao tentar ler algo que não existe mais, mas que existia quando ela começou a ler.

Nisso, ocorrem falhas com coleções regulares, dando lugar as implementações que suportam concorrência.

Paralelismo

Antes de entender o que é paralelismo, precisamos entender, o que é **THREAD**

Paralelismo

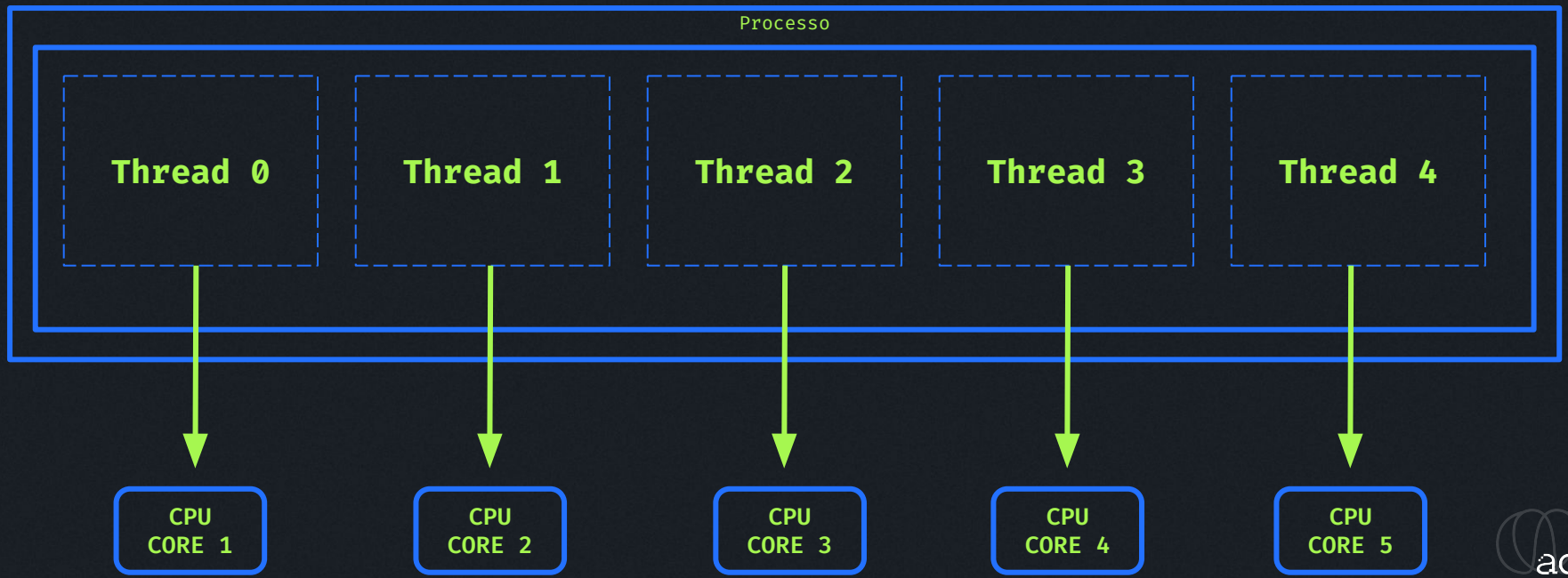
Antes de entender o que é paralelismo, precisamos entender, o que é **THREAD**

Thread também é uma parte do código que pode ser executada independentemente do programa principal.

Por exemplo, um programa pode ter uma thread aberta aguardando um evento específico para ocorrer ou executando um trabalho separado, permitindo que o programa principal execute outras tarefas.

Paralelismo

É o conceito de duas tarefas executarem exatamente ao mesmo tempo, algo apenas possível em processadores com mais de um núcleo.



ada

Programação não-blocante

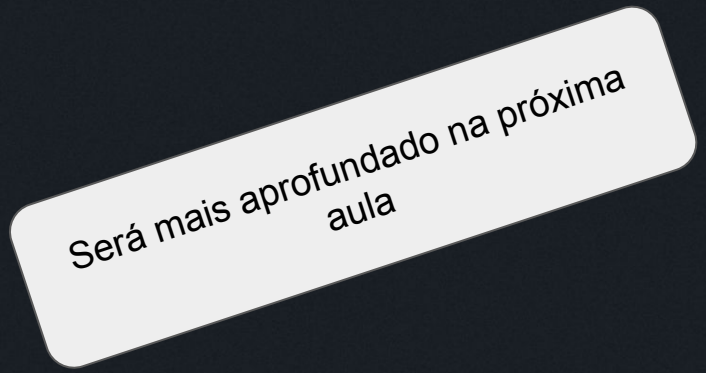
É a ideia é que nenhum código possa bloquear a execução de outras tarefas.

Ao utilizar programação reativa, ou Coroutines, tarefas que antes bloqueavam, podem utilizar construções que as permitem entregar o controle da execução e retomar essa execução quando o recurso estiver pronto.

Com a programação não-bloqueante, tarefas que bloqueariam a execução, desperdiçando tempo de processamento, não o fazem mais, e permite que outras tarefas que irão gastar o tempo de CPU fazendo algo útil, o façam sem interferir na capacidade das demais tarefas de esperar por algum recurso.

Programação assíncrona

Se trata de uma tarefa que executa em algum momento no futuro, ou seja, é uma tarefa que pode rodar em background enquanto o fluxo regular da aplicação segue normalmente.



Agora

Finalmente . . .

Vamos as Corrotinas

O que são Corrotina?

Coroutines são componentes da programação que permitem a suspensão e o resumo da execução de uma sub rotina. Em outras palavras, permitem que a execução de uma função “pare” em um ponto do código e continue desse mesmo ponto quando forem requisitadas para o fazer.

Ela é uma feature do Kotlin na qual possibilita escrever códigos assíncronos mais facilmente e de maneira sequencial, sem usar o padrão de Callback (o famoso Callback Hell). Coroutines está disponível desde o Kotlin 1.1 como experimental, ou a partir do Kotlin 1.3 como versão estável.

O que são Corrotina?

O conceito de coroutines não é exclusivo do Kotlin, algumas linguagens como Clojure e Rust implementam através de bibliotecas terceiras, outras linguagens como Kotlin e Go implementam de maneira nativa. Também encontramos coroutines expressas com um nome mais “artístico” e específico de acordo da linguagem como: Goroutines (Go) e Cloroutine (Clojure).

O que são Corrotina?

Já sabemos o que são Corrotinas e o que são Threads . . .
mas, qual a diferença?

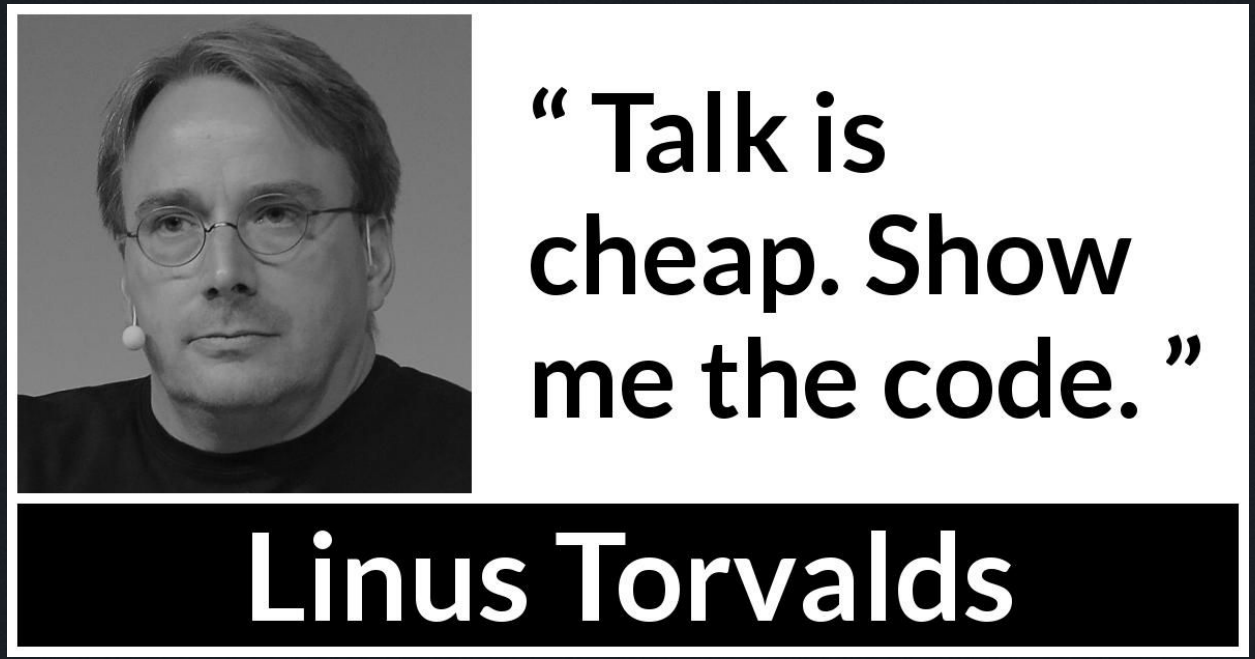
A própria documentação do Kotlin fala que você pode pensar em coroutines como uma light-weight thread, tradução livre, thread leve

Várias coroutines podem ser criadas na mesma threads, em grandes quantidades.

Criar 100 mil corrotinas 👍
Criar mil Threads? muitos Out of Memory

O que são Corrotina?

Criar 100 mil corrotinas 👍



O que são Corrotina?

Em Resumo . . .

Comparado a Thread, as corrotinas tem:
- Menor Custo na criação
- Manor custo na troca de contexto

e . . .

São muito mais eficientes, várias corrotinas podem rodar usando apenas uma única thread

Antes de entender um pouco mais no código . . .

Quando é absolutamente necessário chamar uma corrotina a partir de funções regulares, temos que utilizar o runBlocking.

Ela irá bloquear a Thread atual até que a Corrotina complete.

O que são Corrotina?

Primeiro exemplo **PRÁTICO**

“ Talk is cheap. Show me the code. ”

Linus Torvalds

O que são Corrotina?

Corrotinas x Threads

“ Talk is cheap. Show me the code. ”

Linus Torvalds

O que são Corrotina?

Existem dois tipos de Coroutines: stackful e stackless, Kotlin implementa a última.

VAMOS A UMA BREVE ATIVIDADE DE PESQUISA ?

Faça uma breve pesquisa, e traga em poucas palavras sobre os tipos de corrotinas Stackful e stackless, e qual o kotlin usa.

Poste sua resposta na atividade que está no class, e no fim do tempo, um aluno nos conte o que achou :D

15min

ada

Funções Suspensas

Suspend functions

Funções suspend (declaradas como suspend fun) são funções que podem ser suspensas sem bloquear a thread. Ou seja, uma função suspend pode ser pausada e resumida sem bloquear a thread atual.

Função Blocante	Função Suspensa
Só libera a thread na qual está executando após terminar a execução de tudo	Pode pausar durante sua execução para que outra função possa executar na mesma thread

Funçao Launch

A função launch cria uma corrotina e a inicia em background e sem bloquear a thread a qual ela está associada. Caso não seja passado qual contexto onde a corrotina irá executar, ela herdará o contexto de onde ela está sendo inicializada.

Funções Suspensas

Suspend functions

“ Talk is cheap. Show me the code. ”

Linus Torvalds

Dispatchers ("contexto")

Para executar uma coroutine em um contexto específico, como uma operação que faz acesso ao disco ou que realiza uma série de processamentos matemáticos, existem os Dispatchers e a função withContext

Todas Coroutines executam em um certo “contexto”, esse contexto armazena dois valores importantes, o Job representando a execução, e o agendador utilizado para despachar a execução.

Dispatchers ("contexto")

Os contextos podem ser utilizados para armazenar informações a serem compartilhadas com todas outras Coroutines, que tenham sido iniciadas nesse mesmo contexto, ou seja, elas herdam os contextos, criando uma hierarquia entre eles.

Agendadores

Existem 4 principais agendadores de Coroutines: Default, IO, Main e Unconfined.

Dispatchers ("contexto")

Default

É o agendador padrão utilizado quando nenhum é especificado, principalmente nas funções launch e async. Esse agendador utiliza um pool de Threads compartilhado, com número de Threads equivalente ao número de núcleos do processador, ou no mínimo duas Threads (quando o sistema reporta que tem apenas um núcleo).

Dispatchers ("contexto")

Default

“ Talk is cheap. Show me the code. ”

Linus Torvalds

Dispatchers ("contexto")

IO

Esse, por sua vez, é utilizado para delegar tarefas de I/O, como tarefas de leitura e escrita de arquivos, comunicações de rede, comunicação entre processos, e assim por diante.

Tarefas despachadas para o agendador IO escalam para até 64 Threads, ou o número de núcleos disponíveis (qual for maior), e esse valor pode ser configurado por meio de propriedades da aplicação.

Dispatchers ("contexto")

IO

“ Talk is cheap. Show me the code. ”

Linus Torvalds

Dispatchers ("contexto")

Main

Agendador para enviar tarefas para a mesma Thread responsável pela UI, a ativação desse agendador depende da presença de certas dependências em sub-módulos do Kotlinx Coroutines, como a dependência do Android, JavaFx ou Swing.

Normalmente, esse agendador é single-threaded (pois as UIs costumam rodar em apenas uma Thread).

Dispatchers ("contexto")

Main

“ Talk is cheap. Show me the code. ”

Linus Torvalds

Dispatchers ("contexto")

Unconfined

Recomendado para operações que não precisam de uma thread específica. É recomendado usar quando não consome tempo de CPU nem atualizar dados compartilhados (como a interface de usuário), confinados em uma thread específica.

A coroutine que usa esse Dispatcher é executada na mesma thread de quem a chamou, mas só se mantém nessa thread até o primeiro ponto de suspensão (primeira suspend fun). Depois de suspendida, é resumida na thread.

Dispatchers ("contexto")

Unconfined

“ Talk is cheap. Show me the code. ”

Linus Torvalds

Links Uteis

- https://jhrl.medium.com/kotlin-coroutines-uma-abordagem-diferente-para-programa%C3%A7%C3%A3o-concorrente-85b16573c209#:~:text=em%20sua%20ess%C3%AAncia•–,Programa%C3%A7%C3%A3o%20concorrente,necessariamente%20executando%20ao%20mesmo%20tempo.
- https://medium.com/ifood-tech/kotlin-coroutines-e6d048a59c40
- https://www.youtube.com/watch?v=xNBMNKjpJzM
- https://kotlinlang.org/docs/coroutines-basics.html#coroutines-are-light-weight
- https://coderef.com.br/kotlin-coroutines-o-que-s%C3%A3o-f9375c314480
- https://medium.com/android-dev-br/utilizando-kotlin-coroutines-no-android-c73fcda71e27
- https://medium.com/@tguizelini/android-mvvm-retrofit-coroutines-s2-7f09bce0ad0c
- https://medium.com/@bhavnathacker14/deep-dive-into-dispatchers-for-kotlin-coroutines-f38527bde94c
- https://stackoverflow.com/questions/59039991/difference-between-usage-of-dispatcher-io-and-default
-