UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

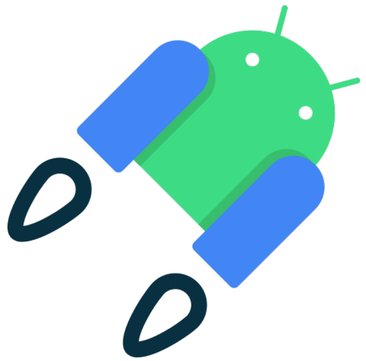
Navegação - parte 2

QXD0102 - Desenvolvimento para Dispositivos Móveis

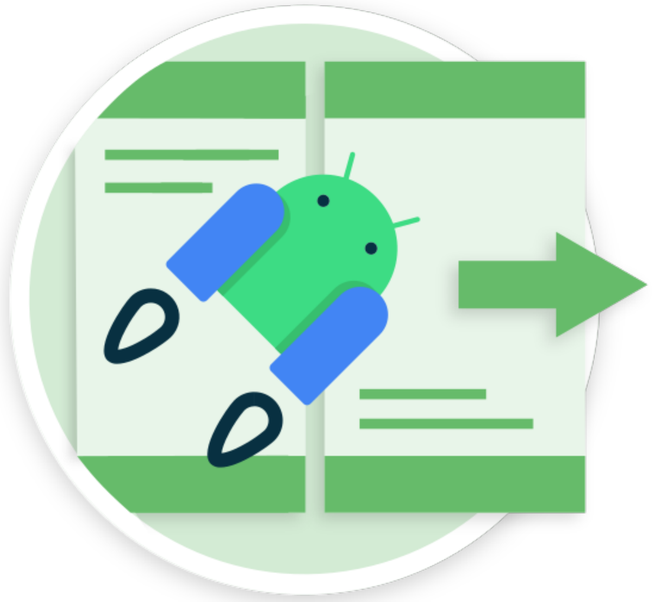
Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Conteúdo

- Fragments
- Navigation Component
- Architecture Components
- Exemplo avançado de navegação

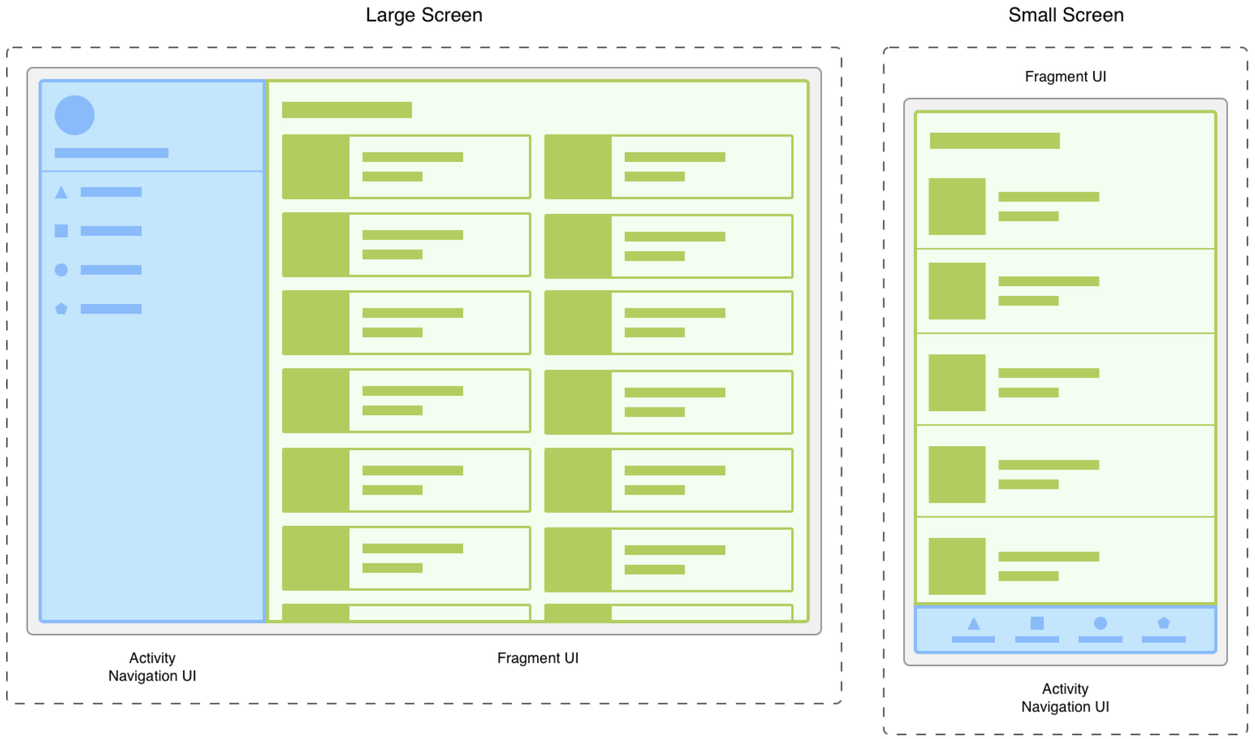


Fragments



Fragments

- Um fragment é uma parte reutilizável da IU que pode ser reutilizada e incorporada em uma ou mais activities
- É possível exibir vários fragmentos de uma vez em uma única tela



Fragments

- Possuem um ciclo de vida e podem responder à entrada do usuário
- Está sempre contido na hierarquia de views de uma activity quando é exibido na tela
- É possível que vários fragmentos sejam hospedados simultaneamente por uma única activity
- Cada fragment gerencia o próprio ciclo de vida separado

Fragments

Ciclo de vida dos fragments

- Assim como as activities, os fragmentos podem ser inicializados e removidos da memória e, ao longo da existência deles, aparecer, desaparecer e reaparecer na tela
- Também possuem um ciclo de vida com vários estados e oferecem diversos métodos que podem ser modificados para responder às transições entre eles
- O ciclo de vida do fragmento tem cinco estados, representado pela enumeração Lifecycle.State
  - INITIALIZED
  - CREATED
  - STARTED
  - RESUMED
  - DESTROYED

Fragments

Ciclo de vida dos fragments

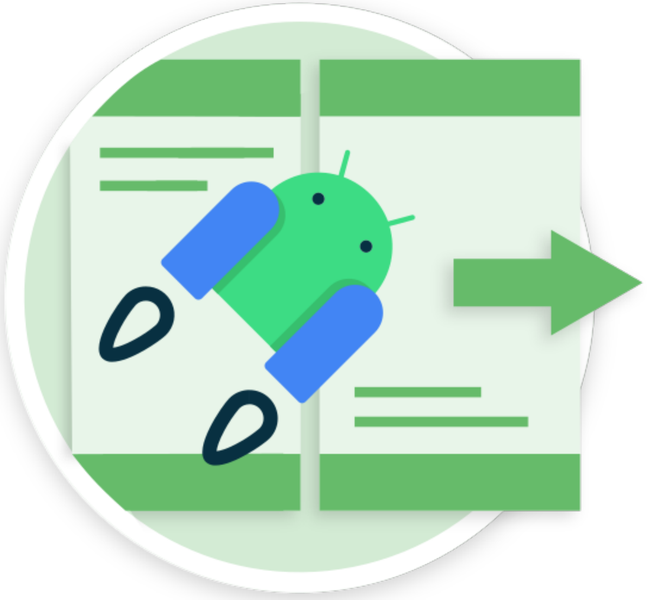
- onCreate(): o fragmento foi instanciado e está no estado CREATED. No entanto, a view correspondente ainda não foi criada
- onCreateView(): é neste método que você infla o layout
- onViewCreated(): é chamado depois que a view é criada. Nesse método, você normalmente vincula views específicas a propriedades chamando findViewById()
- onStart(): o fragmento entra no estado STARTED
- onResume(): o fragmento entra no estado RESUMED e agora tem foco
- onPause(): o fragmento entra novamente no estado STARTED. A IU está visível para o usuário
- onStop(): o fragmento entra novamente no estado CREATED. O objeto é instanciado, mas não é mais exibido na tela
- onDestroyView(): chamado logo antes do fragmento entrar no estado DESTROYED. A view já foi removida da memória, mas o objeto do fragmento ainda existe
- onDestroy(): o fragmento entra no estado DESTROYED

Lifecycle State	Callback
CREATED	
STARTED	
RESUMED	
STARTED	
CREATED	
DESTROYED	

Fragments

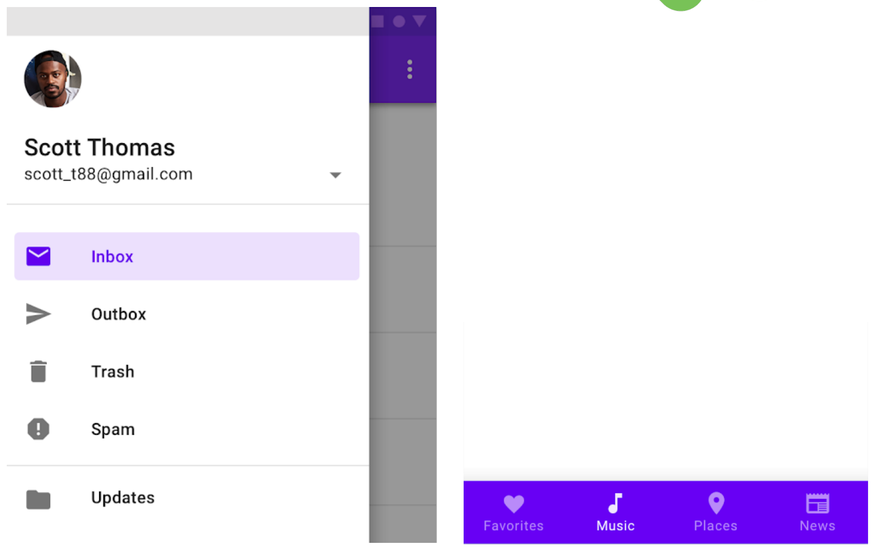
- Os estados do ciclo de vida e os métodos de callback são muito parecidos com os usados em activities.
- Há uma diferença no método onCreate()
  - Em activities, você precisa usar esse método para inflar o layout e vincular as views
  - Em fragments, o onCreate() é chamado antes da criação da views
    - Portanto, não é possível inflar o layout nele
  - O processo de inflar deve ser feito no onCreateView()
  - Em seguida, após a criação das views, o método onViewCreated() é chamado
    - Neste momento deve-se vincular views

Navigation Component



Navigation Component

- É parte do Android Jetpack
- Facilita a implementação de uma navegação até padrões mais complexos:
  - app bars e navigation drawer
- Garante ao usuário uma experiência consistente e previsível devido a sua aderência a uma série de princípios



Navigation Component

É composto de 3 partes

- Navigation Graph
  - Arquivo de recurso XML que centraliza todas as informações referentes a navegação
  - Inclui todas as áreas de conteúdo do app, *destinations*, bem como todos os possíveis caminhos que podem ser seguidos dentro do app

Navigation Component

É composto de 3 partes

- NavHost
  - Um contêiner que mostra os *destinations* do gráfico de navegação
  - Por padrão o NavHost implementa a classe NavHostFragment
    - Mostra *destinations* que são fragments

Navigation Component

É composto de 3 partes

- NavController
  - Um objeto que gerencia a navegação da app junto com o NavHost
  - Orquestra a troca do conteúdo dos destinos no NavHost

Navigation Component

- A medida que você navega no app, você informa ao NavController que você quer navegar seguindo um caminho específico ou diretamente o destino desejado.
- O NavController mostra o a destino apropriado no NavHost.

Navigation Component

Benefícios

- Gerencia as transações de Fragments
- Gerencia as ações de *back* e *up* corretamente por default
- Prover de maneira padronizadas recursos para animações e transições
- Implementa e gerencia o uso de *deep linking*
- Inclui padrões de navegação, como navigation drawer e bottom navigation
- Safe Args, um plugin do Gradle que prover type safety durante a navegação entre destino com passagem de dados
- Suporte a ViewModel, é possível compartilhar dados relacionados a interface entre destinos

Navigation Component

- Mais informações
  - Primeiros passos com o componente Navigation
  - Codelab na qual a prática foi inspirada

Architecture Components

Architecture Components

Introdução

- Na maioria dos casos, aplicações desktop têm um único ponto de entrada e são executados como um único processo monolítico
- Por outro lado, um app Android contém vários componentes, incluindo activities, fragments, services, content providers e broadcast receivers.
  - A maioria desses componentes é declarada no AndroidManifest
  - O Android usa esse arquivo para decidir como integrar seu app à experiência geral do usuário do dispositivo
  - Precisam se adaptar a diferentes fluxos de trabalho e tarefas controlados pelo usuário

Architecture Components

Introdução

Exemplo: Compartilhar uma foto no seu app de rede social favorito

1. O app aciona um intent de câmera. Em seguida, o Android abre um app de câmera para lidar com a solicitação.
   - Nesse momento, o usuário sai do app de rede social
2. O app de câmera pode acionar outros intents, como a abertura do seletor de arquivos, que pode iniciar mais um app
3. Por fim, o usuário retorna ao app de rede social e compartilha a foto

Architecture Components

Introdução

Atenção

- A qualquer momento durante o processo, o usuário pode ser interrompido
  - Ex: por uma chamada telefônica ou notificação
  - Depois de lidar com essa interrupção, o usuário espera retomar o processo de compartilhamento de fotos
  - Esse comportamento de alternância de apps é comum em dispositivos móveis
- Devido aos limitados recursos dos dispositivos móveis limitados, o sistema operacional pode interromper os processos de alguns apps a qualquer momento para dar espaço a outros novos

Architecture Components

Introdução

Como esses eventos não estão sob seu controle, **não armazene nenhum dado ou estado de app nos componentes do seu app e não permita que os componentes dele dependam uns dos outros**

Architecture Components

Princípios arquitetônicos

Separação de conceitos (Separation of Concerns)

- É um erro comum escrever todo o código em Activity ou Fragment
  - Essas classes baseadas em IU devem conter apenas a lógica que processa as interações entre a IU e o sistema operacional
  - Ao manter essas classes o mais enxutas possível, você pode evitar muitos problemas relacionados ao ciclo de vida
- Activity e Fragments representam o contrato entre o SO Android e seu app
  - Podem ser destruídas a qualquer momento com base nas interações do usuário ou devido a condições do sistema, como pouca memória

Architecture Components

Princípios arquitetônicos

Para oferecer uma experiência do usuário satisfatória e uma experiência de manutenção de app mais gerenciável, **o melhor a se fazer é minimizar sua dependência delas**

Architecture Components

Princípios arquitetônicos

Basear a IU em um modelo

- É importante que a IU seja baseada em um modelo, de preferência um que seja persistente
- Modelos são componentes responsáveis por manipular os dados de um app
  - São independentes dos objetos View e componentes do app
  - Não são afetados pelo ciclo de vida do app e pelas preocupações associadas

Architecture Components

Princípios arquitetônicos

- A persistência é ideal pelos seguintes motivos:
  - Seus usuários não perderão dados se o SO Android destruir seu app para liberar recursos
  - Seu app continuará funcionando se uma conexão de rede estiver lenta ou indisponível

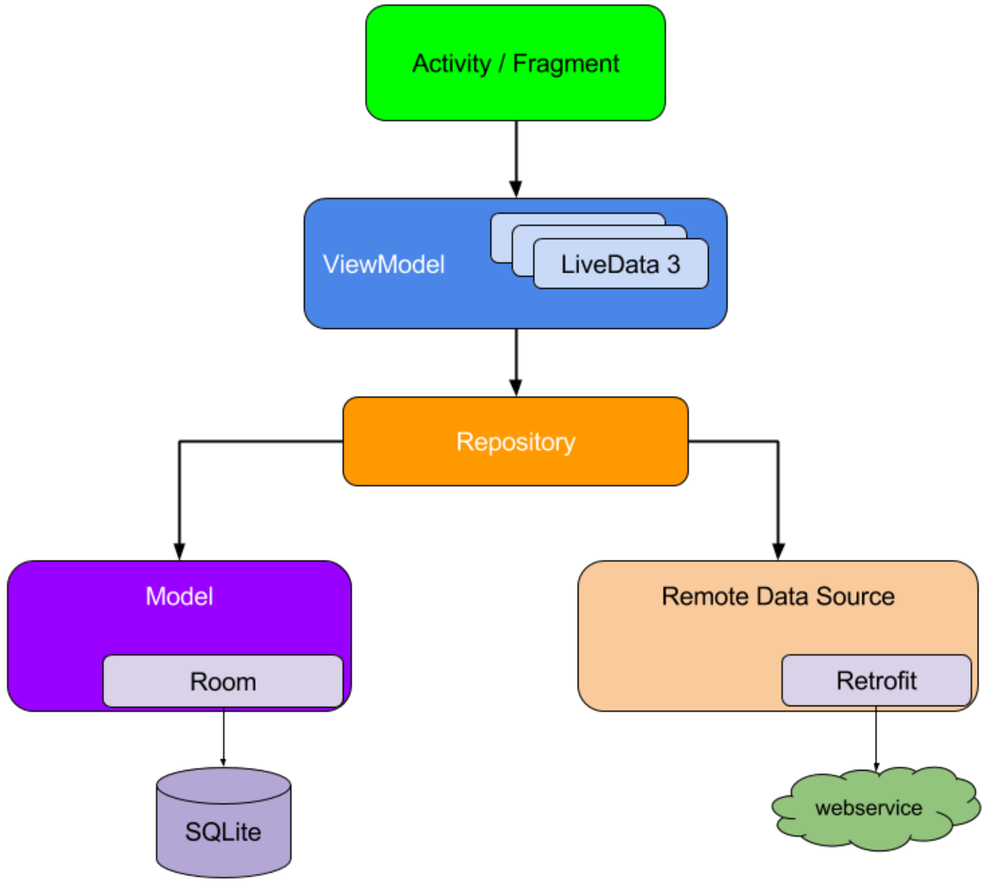
Architecture Components

Princípios arquitetônicos

Baseando seu app em classes de modelo com responsabilidade bem definida de gerenciamento dos dados, ele se torna mais testável e consistente

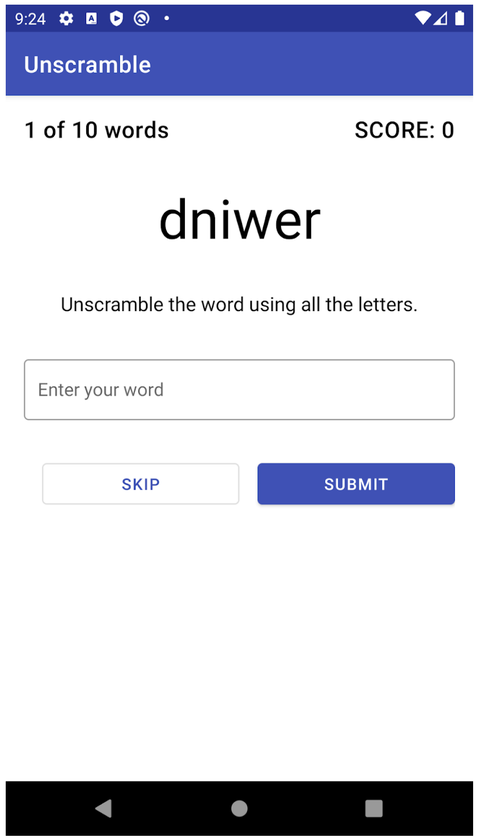
Architecture Components

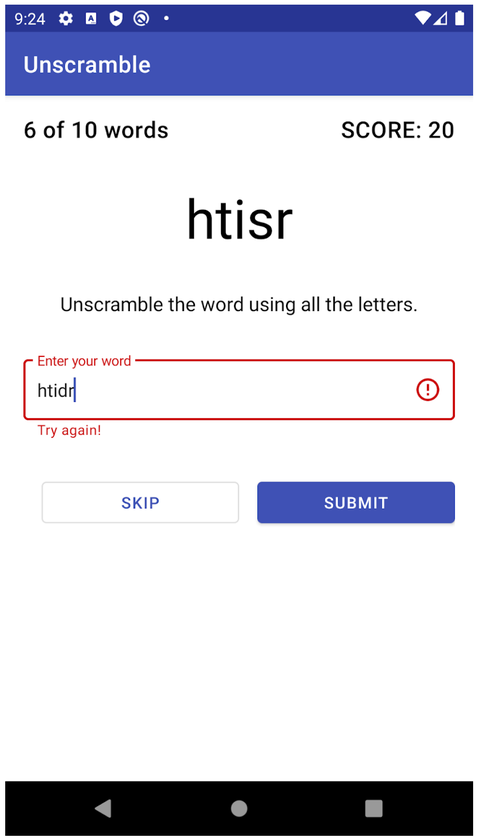
Visão geral

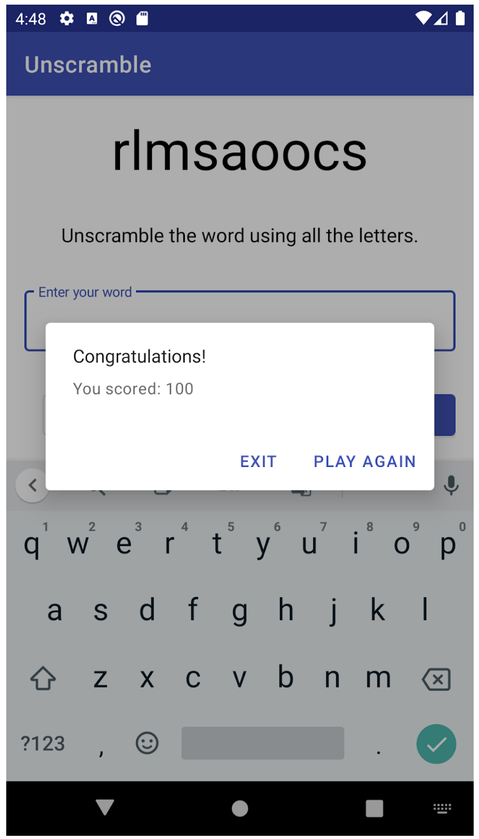


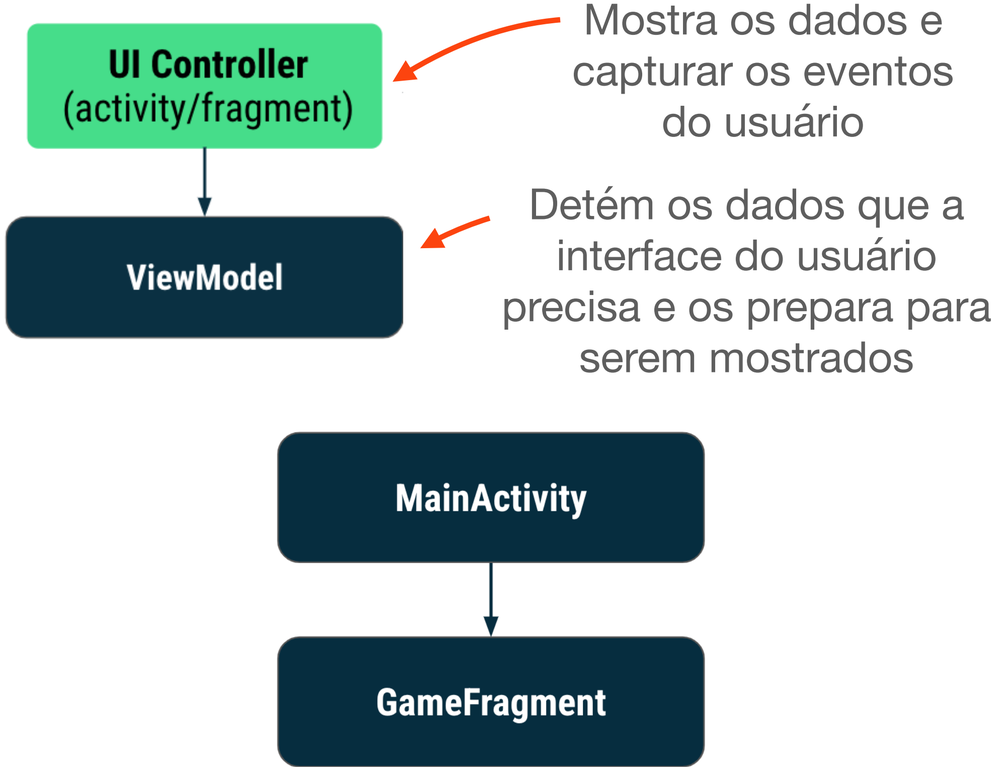
- Cada componente depende apenas daquele que está um nível abaixo de si
  - Activities e Fragments dependem apenas de um ViewModel
- O repositório é a única classe que depende de várias outras classes
  - Depende de um modelo de dados persistente e de uma fonte de dados de back-end remota

Architecture Components



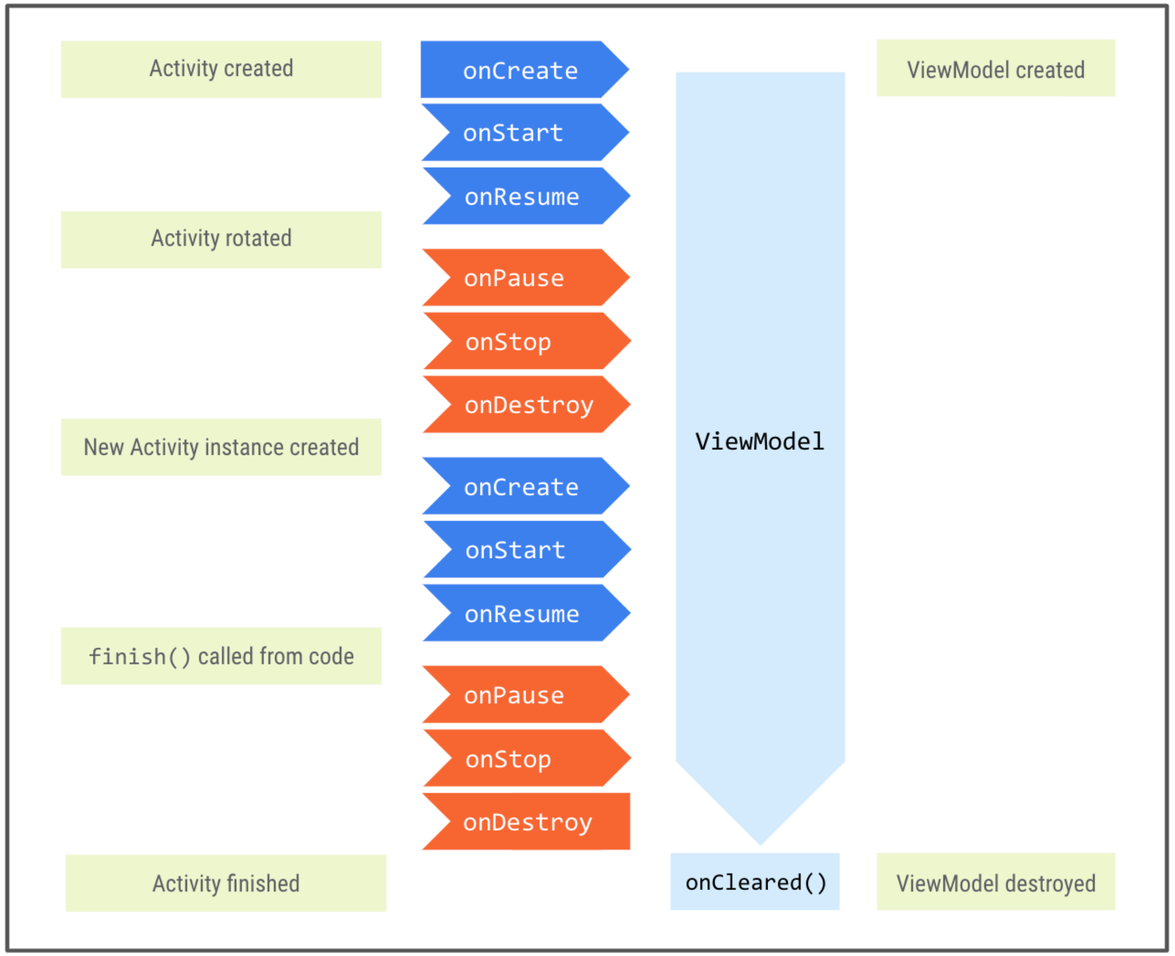






Architecture Components

Ciclo de vida de ViewModels



Architecture Components

LiveData

LiveData

- É uma classe armazenadora de dados observáveis
- É um wrapper que pode ser usado com qualquer tipo de dados
- É observável, o que significa que um observador é notificado quando os dados contidos no objeto LiveData mudam.
- Ao você anexar um observador ao LiveData, ele estará associado a um LifecycleOwner (geralmente uma activity ou fragment)
    - O LiveData só atualiza observadores que estão em um estado de ciclo de vida ativo, como STARTED ou RESUMED

Por hoje é só