

Fundamentos de Desenvolvimento Android

MIT em Desenvolvimento Mobile - 2020

Ciclo de Vida de Activity

Ciclo de Vida de Activity

O ciclo de vida da Activity é composto dos diferentes estados pelos quais uma tela pode passar, desde o momento em que é inicializada pela primeira vez até quando é finalmente destruída e sua memória recuperada pelo sistema.

Conforme o usuário inicia seu aplicativo, navega entre atividades, navega dentro e fora de seu aplicativo e sai dele, ocorrem mudanças de estado.

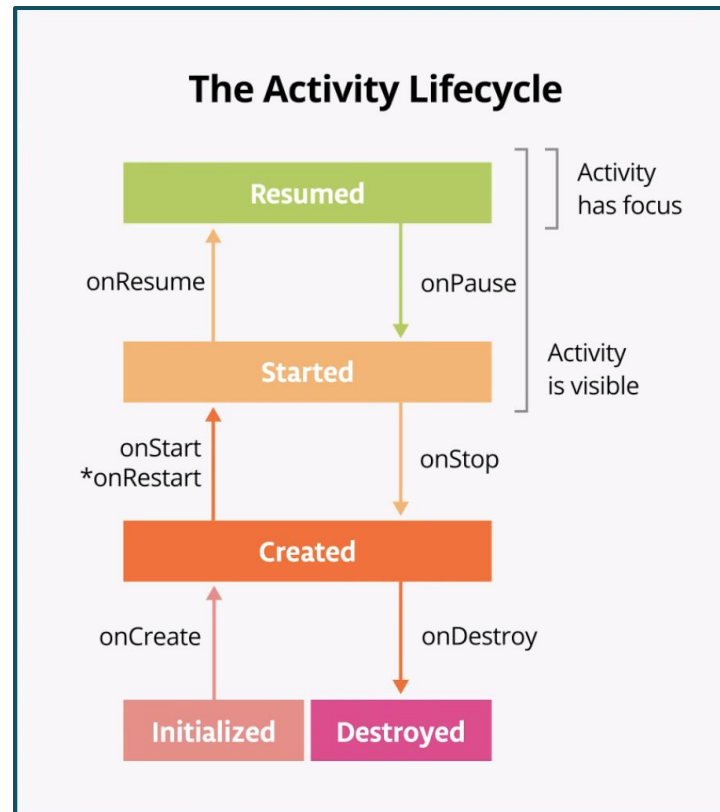
The Activity Lifecycle



Ciclo de Vida de Activity

A Activity e quaisquer subclasses implementam um conjunto de métodos de callback do ciclo de vida.

O Android invoca esses callbacks quando a atividade muda de um estado para outro, e você pode substituir esses métodos em suas próprias atividades para realizar tarefas em resposta a essas mudanças de estado do ciclo de vida.



Ciclo de Vida de Activity

Log grava mensagens no Logcat. Existem três partes para este comando:

- Tipo de mensagem de log:
 - Log.i() → information.
 - Log.e() → error.
 - Log.w() → warning.
- Tag de log, uma string que permite encontrar mais facilmente suas mensagens de log no Logcat.
- A mensagem de log real.



Ciclo de Vida de Activity

1. Criar uma aplicação com uma Activity.
2. Inserir logs para os métodos de ciclo de vida:
 - onCreate() / onDestroy().
 - onStart() / onStop() / onRestart().
 - onResume() / onPause().
3. Testar INICIAR.
4. Testar VOLTAR.
5. Testar HOME.



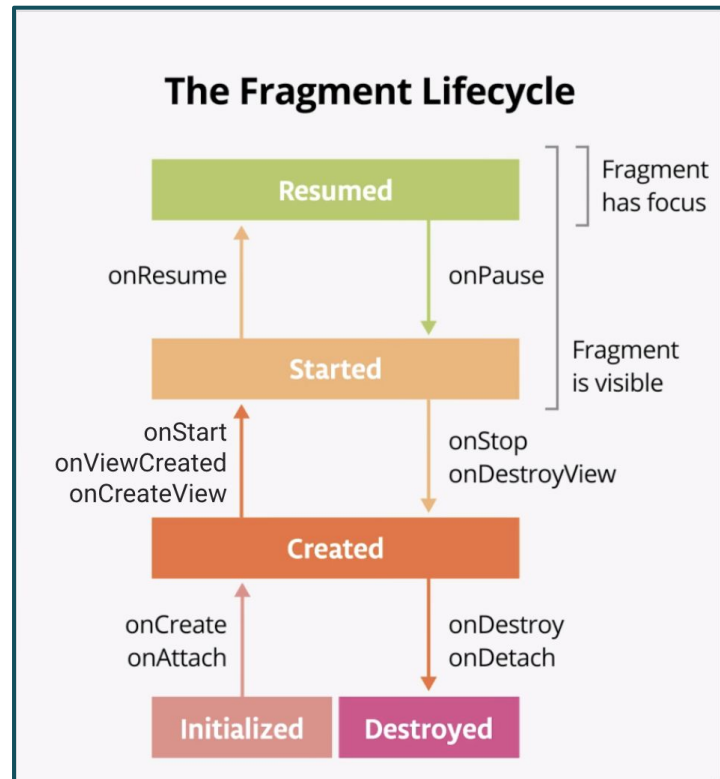
Ciclo de Vida de Fragment

Ciclo de Vida de Fragment

Um fragmento é um "trecho" reaproveitável e auto contido de interface.

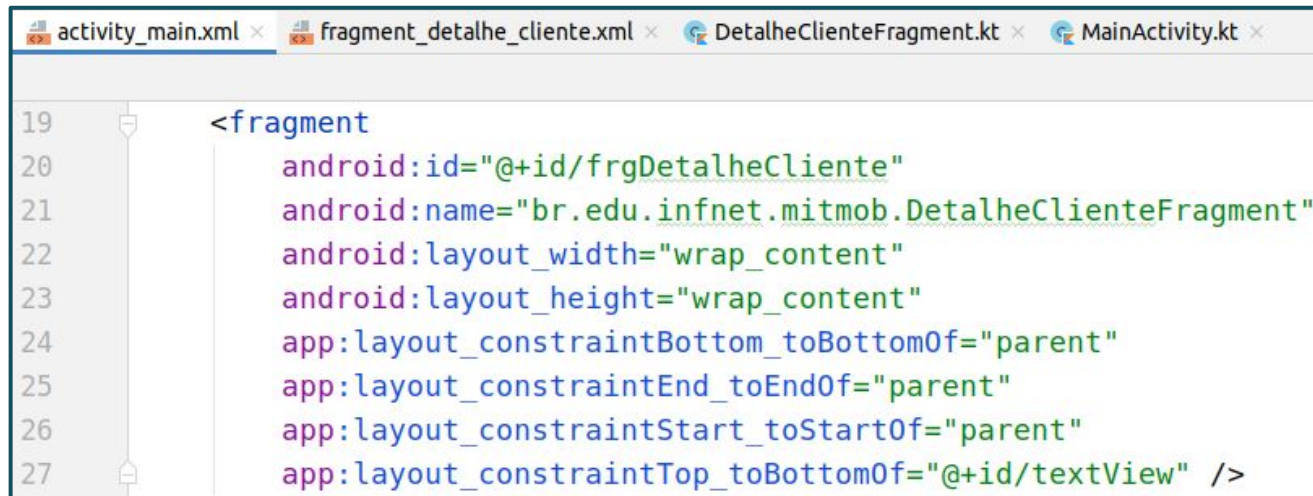
Um fragmento também tem um ciclo de vida.

O ciclo de vida de um fragmento é semelhante ao ciclo de vida de uma atividade, portanto, muito do que você aprende se aplica a ambos.



Ciclo de Vida de Activity e Fragment

1. Inserir um Fragment na aplicação.
2. Inserir logs para os métodos de ciclo de vida:
 - onAttach() / onDettach().
3. Testar INICIAR.
4. Testar VOLTAR.
5. Testar HOME.



The screenshot shows the Android Studio interface with four tabs open: activity_main.xml, fragment_detalhe_cliente.xml, DetalheClienteFragment.kt, and MainActivity.kt. The active tab is fragment_detalhe_cliente.xml, which displays the following XML code:

```
19 <fragment
20     android:id="@+id/frgDetalheCliente"
21     android:name="br.edu.infnet.mitmob.DetalheClienteFragment"
22     android:layout_width="wrap_content"
23     android:layout_height="wrap_content"
24     app:layout_constraintBottom_toBottomOf="parent"
25     app:layout_constraintEnd_toEndOf="parent"
26     app:layout_constraintStart_toStartOf="parent"
27     app:layout_constraintTop_toBottomOf="@+id/textView" />
```

LifeCycleObserver

LifecycleObserver

Em um aplicativo Android mais complexo você pode configurar muitas coisas em `onStart()` / `onCreate()` e em seguida desmontá-las em `onStop()` / `onDestroy()`.

Por exemplo, você pode ter animações, música, sensores ou cronômetros que você precisa para configurar e derrubar, e iniciar e parar.

`LifecycleObserver` é especialmente útil nos casos em que você precisa rastrear muitos componentes - o próprio componente observa as mudanças do ciclo de vida e faz o que é necessário quando essas mudanças acontecem.

```
MainActivity.kt x MusicaController.kt x
8 class MusicaController(lifecycle: Lifecycle) : LifecycleObserver {
9
10     init {
11         lifecycle.addObserver( observer: this)
12     }
13
14     @OnLifecycleEvent(Lifecycle.Event.ON_START) 2
15     fun iniciarMusica() {
16
17         Log.i( tag: "MIT", msg: "MusicaController.iniciarMusica")
18     }
19
20     @OnLifecycleEvent(Lifecycle.Event.ON_STOP)
21     fun pararMusica() {
22
23         Log.i( tag: "MIT", msg: "MusicaController.pararMusica")
24     }
25 }
```

1

2

3

```
MainActivity.kt x MusicaController.kt x
6 class MainActivity : AppCompatActivity() {
7     override fun onCreate(savedInstanceState: Bundle?) {
8         super.onCreate(savedInstanceState)
9         setContentView(R.layout.activity_main)
10
11     3 val musicaController = MusicaController(this.lifecycle)
12     }
13 }
```

ViewModel

ViewModel

Um controlador de UI é uma classe como Activity ou Fragment.

Um controlador de UI deve conter apenas a lógica que lida com as interações da interface com o usuário e do sistema operacional, como exibir visualizações e capturar a entrada do usuário.

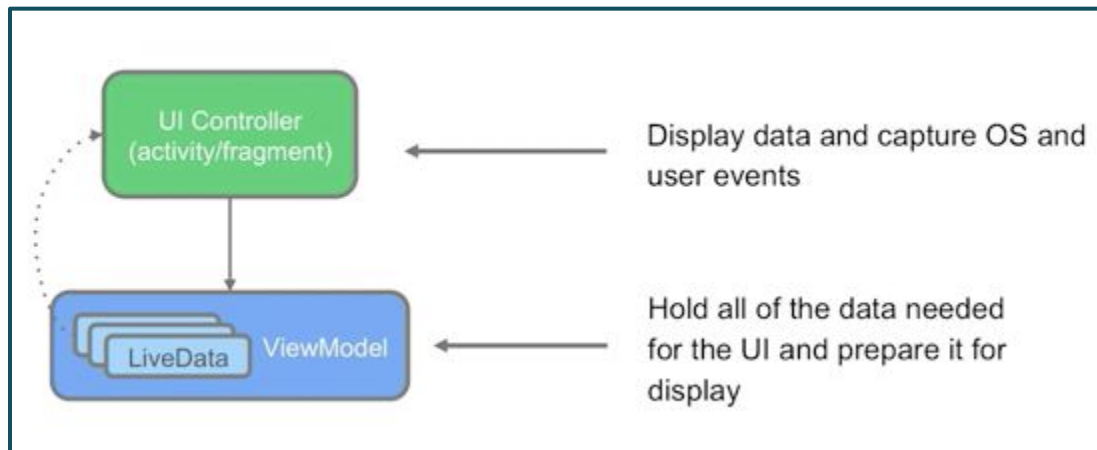
Não coloque lógica de tomada de decisão, como a lógica que determina o texto a ser exibido, no controlador de UI.



ViewModel

ViewModel contém dados a serem exibidos em um fragmento ou atividade.

Um ViewModel pode fazer cálculos e transformações simples nos dados para prepará-los para serem exibidos pelo controlador de UI.



```
MainActivity.kt x build.gradle (:app) x MainActivityViewModel.kt x
6  android {
7      compileSdkVersion 29
8      buildToolsVersion "30.0.1"
9
10     defaultConfig {...}
19
20     buildTypes {...}
26     compileOptions {...}
30     kotlinOptions {jvmTarget = '1.8'}
33
34     buildFeatures {
35         dataBinding true
36     }
37 }
```

```
MainActivity.kt x build.gradle (:app) x MainActivityViewModel.kt x
39 dependencies {
40
41     implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
42     implementation 'androidx.core:core-ktx:1.3.2'
43     implementation 'androidx.appcompat:appcompat:1.2.0'
44     implementation 'com.google.android.material:material:1.2.1'
45     implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
46     implementation 'androidx.legacy:legacy-support-v4:1.0.0'
47     implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'
48
49     testImplementation 'junit:junit:4.+
50     androidTestImplementation 'androidx.test.ext:junit:1.1.2'
51     androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
52
53 }
```



```
MainActivityViewModel.kt x
7  class MainActivityViewModel : ViewModel() {
8
9  1  var frase = ""
10
11  init {
12
13      Log.i( tag: "MIT", msg: "MainActivityViewModel Criada")
14      val calendar : Calendar = Calendar.getInstance()
15      when(calendar.get(Calendar.HOUR_OF_DAY)) {
16
17          in 0..11 -> frase = "Bom Dia!"
18          in 12..17 -> frase = "Boa Tarde!"
19          in 18..23 -> frase = "Boa Noite"
20      }
21  }
22
23  override fun onCleared() {
24
25      super.onCleared()
26      Log.i( tag: "MIT", msg: "MainActivityViewModel Destruída")
27  }
```

Testar o ciclo de vida da aplicação

activity_main.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2
3  1 <layout xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      tools:context=".MainActivity">
7
8  2 <androidx.constraintlayout.widget.ConstraintLayout
9      android:layout_width="match_parent"
10     android:layout_height="match_parent">
11
12     <TextView
13         android:id="@+id/textView"
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:layout_marginTop="16dp"

```

MainActivity.kt x

```
10 class MainActivity : AppCompatActivity() {  
11  
12     1 private lateinit var binding: ActivityMainBinding  
13     private lateinit var viewModel: MainActivityViewModel  
14  
15     override fun onCreate(savedInstanceState: Bundle?) {  
16  
17         super.onCreate(savedInstanceState)  
18         //setContentView(R.layout.activity_main)  
19     2 binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_main)  
20  
21     3 viewModel = ViewModelProvider( owner: this).get(MainActivityViewModel::class.java)  
22     binding.textView.text = viewModel.frase  
23 }
```

LiveData

LiveData

LiveData é uma classe portadora de dados observável que reconhece o ciclo de vida.

Observável significa que um observador é notificado quando os dados mantidos pelo objeto LiveData mudam.

É um wrapper que pode ser usado com qualquer dado.

Está ciente do ciclo de vida - o observador é associado a um LifecycleOwner (geralmente uma Activity ou fragmento). O LiveData atualiza apenas observadores que estão em um estado de ciclo de vida ativo, como STARTED ou RESUMED.



```
MainActivityViewModel.kt x
8  class MainActivityViewModel : ViewModel() {
9
10     var frase = ""
11     var nome = MutableLiveData<String>()
```

```
MainActivity.kt x
16  override fun onCreate(savedInstanceState: Bundle?) {
17
18      super.onCreate(savedInstanceState)
19      //setContentView(R.layout.activity_main)
20      binding = DataBindingUtil.setContentView<activity: this, R.layout.activity_main>
21      viewModel = ViewModelProvider<owner: this>.get(MainActivityViewModel::class.java)
22      binding.textView.setText(viewModel.frase)
23      //-----
24      viewModel.nome.observe<owner: this, Observer { nome ->
25
26          binding.lblNome.setText(nome)
27      })
28      //-----
29      binding.txtNome.setOnKeyListener { view, i, keyEvent ->
30
31          viewModel.nome.value = binding.txtNome.text.toString()
32          true ^setOnKeyListener
33      }
34  }
```

MutableLiveData X LiveData

Os dados em um MutableLiveData podem ser alterados, como o nome indica. Dentro do ViewModel, os dados devem ser editáveis, para que ele use MutableLiveData.

Os dados em um LiveData podem ser lidos, mas não alterados. De fora do ViewModel, os dados devem ser legíveis, mas não editáveis, portanto, os dados devem ser expostos como LiveData.

Esse assunto será
aprofundado na
próxima aula

Data Binding

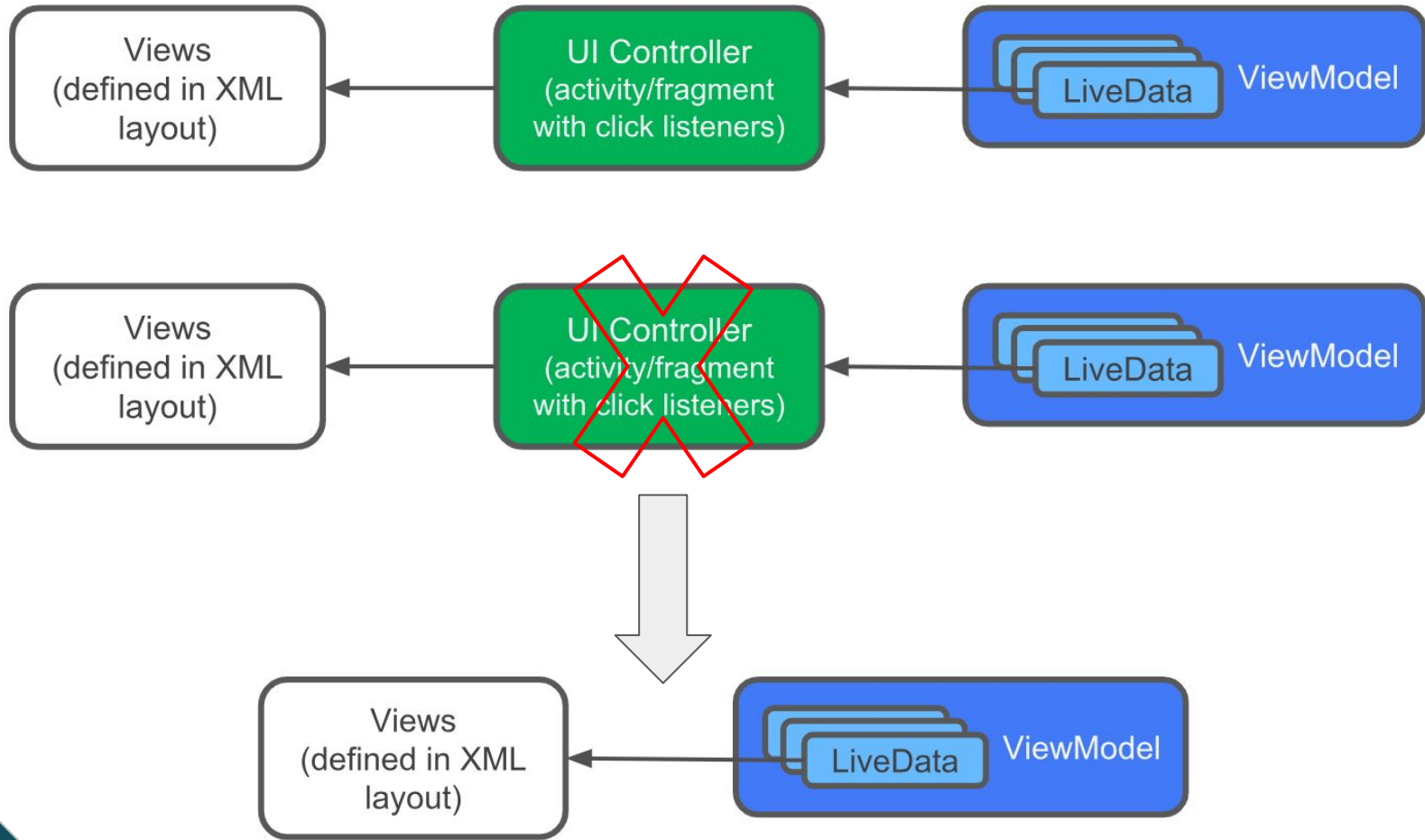
Data Binding

O verdadeiro poder da vinculação de dados está em fazer o que o nome sugere: vincular dados diretamente aos objetos de exibição em seu aplicativo.

Em seu aplicativo, as visualizações são definidas no layout XML e os dados para essas visualizações são mantidos em ViewModel. Entre cada visualização e seu correspondente ViewModel está um controlador de UI, que atua como um retransmissor entre eles.

Seria mais simples se as visualizações no layout se comunicassem diretamente com os dados nos ViewModel, sem depender de controladores de UI como intermediários.





activity_main.xml x

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <layout xmlns:android="http://schemas.android.com/apk/res/android"
4         xmlns:app="http://schemas.android.com/apk/res-auto"
5         xmlns:tools="http://schemas.android.com/tools"
6         tools:context=".MainActivity">
7
8      1 <data>
9         <variable
10            name="viewmodel"
11            type="br.edu.infnet.mitmob.MainActivityViewModel" />
12    </data>
13
14    <androidx.constraintlayout.widget.ConstraintLayout
15        android:layout_width="match_parent"
16        android:layout_height="match_parent">
```

activity_main.xml x

```
18 <TextView
19     android:id="@+id/textView"
20     android:layout_width="wrap_content"
21     android:layout_height="wrap_content"
22     android:layout_marginTop="16dp"
23     1 android:text="@{viewModel.frase}"
24     app:layout_constraintEnd_toEndOf="parent"
25     app:layout_constraintStart_toStartOf="parent"
26     app:layout_constraintT
```

activity_main.xml x

```
39 <TextView
40     android:id="@+id/lblNome"
41     2 android:text="@{viewModel.nome}"
42     android:layout_width="wrap_content"
43     android:layout_height="wrap_content"
44     android:layout_marginTop="16dp"
45     app:layout_constraintEnd_toEndOf="parent"
46     app:layout_constraintStart_toStartOf="parent"
47     app:layout_constraintTop_toBottomOf="@+id/txtNome" />
48
49 </androidx.constraintlayout.widget.ConstraintLayout>
50 </layout>
```

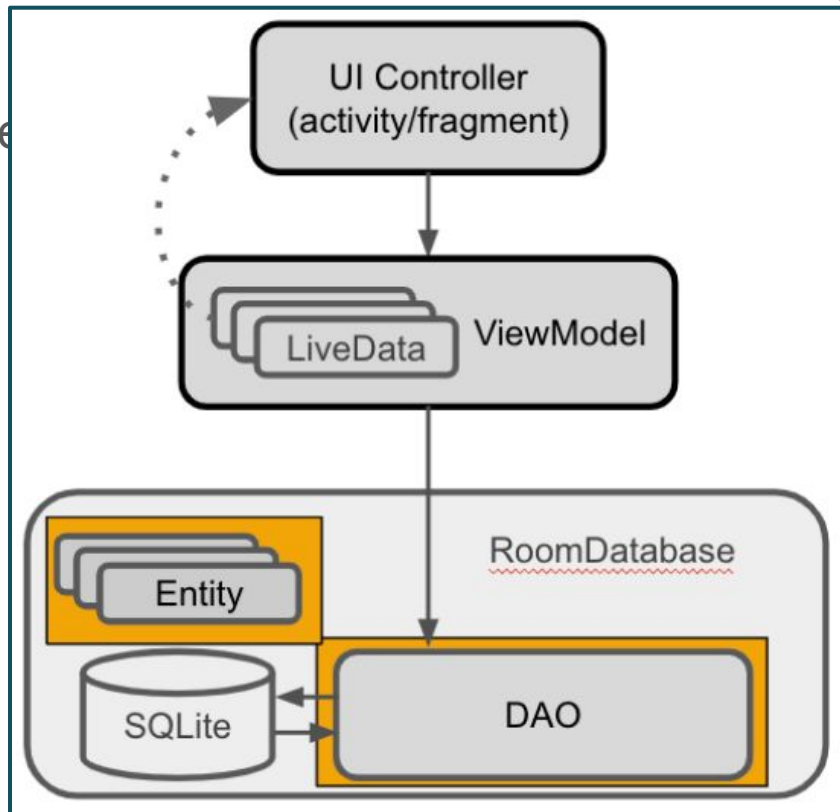
```
18 super.onCreate(savedInstanceState)
19 //setContentView(R.layout.activity_main)
20 binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_main)
21 viewModel = ViewModelProvider( owner: this).get(MainActivityViewModel::class.java)
22 binding.viewModel = viewModel
23 binding.lifecycleOwner = this
24 //binding.textView.setText(viewModel.frase)
25 //-----
26 /*
27 viewModel.nome.observe(this, Observer { nome ->
28
29     binding.lblNome.setText(nome)
30 })
31 */
32 //-----
33 binding.txtNome.setOnKeyListener { view, i, keyEvent ->
34
35     viewModel.nome.value = binding.txtNome.text.toString()
36     true ^setOnKeyListener
37 }
38 }
```

Room

Room

No Android, os dados são representados em classes de dados, e os dados são acessados e modificados usando chamadas de função.

Room faz todo o trabalho duro para você ir das classes de dados Kotlin às entidades que podem ser armazenadas em tabelas SQLite e de declarações de funções a consultas SQL.



Room

Item 0
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9

Nome: _____
Email: _____
Telephone: _____

SALVAR **EXCLUIR**

listContatos

Nome: _____
Email: _____
Telephone: _____

SALVAR **EXCLUIR**

1- Desenhar a
Tela


```
build.gradle (:app) x
1  plugins {
2      id 'com.android.application'
3      id 'kotlin-android'
4      id 'kotlin-android-extensions'
5      1 id 'kotlin-kapt'
6  }
```

```
build.gradle (:app) x
8  android {
9      compileSdkVersion 29
10     buildToolsVersion "30.0.1"
11
12     defaultConfig {...}
21
22     buildTypes {...}
28     compileOptions {...}
32     kotlinOptions {jvmTarget = '1.8'}
35     2 buildFeatures {
36
37         dataBinding true
38     }
39 }
```

```
build.gradle (:app) x
51  //-----
52  //ViewModel
53  3 implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'
54  //-----
55  //Room
56  def room_version = "2.2.5"
57  implementation "androidx.room:room-runtime:$room_version"
58  kapt "androidx.room:room-compiler:$room_version"
59  implementation "androidx.room:room-ktx:$room_version"
60  testImplementation "androidx.room:room-testing:$room_version"
61  }
```

2- Configurar o projeto

activity_main.xml x

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  1 <layout xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      tools:context=".MainActivity">
7
8      <androidx.constraintlayout.widget.ConstraintLayout
9          android:layout_width="match_parent"
10         android:layout_height="match_parent">
11
12         <androidx.recyclerview.widget.RecyclerView
13             android:layout_width="0dp"
14             android:layout_height="0dp"
15             app:layout_constraintBottom_toTopOf="@+id/scrollView2"
16             app:layout_constraintEnd_toEndOf="parent"
17             app:layout_constraintStart_toStartOf="parent"
18             app:layout_constraintTop_toTopOf="parent" />
19
```

3- Configurar o
Data Binding

```
MainActivityViewModel.kt x
1 package br.edu.infnet.roomtest
2
3 import androidx.lifecycle.ViewModel
4
5 1 class MainActivityViewModel : ViewModel() {
6
7
8 }
```

4- Criar o
ViewModel

```
activity_main.xml x
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <layout xmlns:android="http://schemas.android.com/apk/res/android"
4         xmlns:app="http://schemas.android.com/apk/res-auto"
5         xmlns:tools="http://schemas.android.com/tools"
6         tools:context=".MainActivity">
7
8 2 <data>
9     <variable
10         name="viewmodel"
11         type="br.edu.infnet.roomtest.MainActivityViewModel" />
12 </data>
13
14 <androidx.constraintlayout.widget.ConstraintLayout
```

```
MainActivity.kt x
9  class MainActivity : AppCompatActivity() {
10
11  1 private lateinit var binding : ActivityMainBinding
12  private lateinit var viewmodel: MainActivityViewModel
13
14  2 override fun onCreate(savedInstanceState: Bundle?) {
15
16      super.onCreate(savedInstanceState)
17      //setContentView(R.layout.activity_main)
18      binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_main)
19      viewmodel = ViewModelProvider( owner: this).get(MainActivityViewModel::class.java)
20      binding.viewmodel = viewmodel
21      binding.lifecycleOwner = this
22  }
23 }
```

5- Ligar a UI
Controller ao
ViewModel

```
Contato.kt x
1 package br.edu.infnet.roomtest
2
3 import androidx.room.ColumnInfo
4 import androidx.room.Entity
5 import androidx.room.PrimaryKey
6
7 1 @Entity(tableName = "contatos")
8   data class Contato(
9       2 @PrimaryKey(autoGenerate = true)
10         var id: Int,
11         @ColumnInfo(name = "nome")
12         var nome: String,
13       3 @ColumnInfo(name = "email")
14         var email: String,
15         @ColumnInfo(name = "fone")
16         var fone: String
17     )
```

6- Criar a Entidade do Room

<https://developer.android.com/reference/androidx/room/package-summary>




```
5 1 @Dao
6  interface ContatoDAO {
7
8      @Query( value: "SELECT * FROM contatos")
9      suspend fun listar(): List<Contato>
10
11      @Query( value: "SELECT * FROM contatos WHERE nome LIKE :nome LIMIT 1")
12      suspend fun obterPorNome(nome: String): Contato
13
14      @Insert(onConflict = OnConflictStrategy.IGNORE)
15      suspend fun inserir(vararg contato: Contato)
16
17      @Update
18      suspend fun atualizar(contato: Contato)
19
20      @Delete
21      suspend fun excluir(contato: Contato)
22 }
```

7- Criar o DAO
para a Entidade

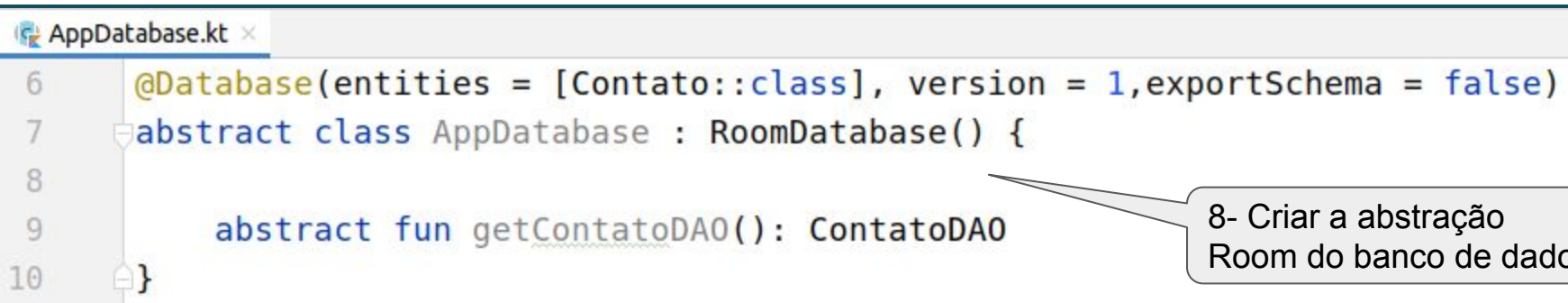
Uma função "suspend" é simplesmente uma função que pode ser pausada e retomada posteriormente.

Elas podem executar uma operação de longa duração e esperar que ela seja concluída sem bloquear.

Room

@Database requer vários argumentos, para que o Room possa construir o banco de dados.

- Forneça **Contato** como o único item com a lista de entities.
- Defina o **version** como 1. Sempre que você alterar o esquema, terá que aumentar o número da versão.
- Defina **exportSchema** como false, para não manter backups do histórico de versões do esquema.



```
AppDatabase.kt x
6  @Database(entities = [Contato::class], version = 1,exportSchema = false)
7  abstract class AppDatabase : RoomDatabase() {
8
9      abstract fun getContatoDAO(): ContatoDAO
10 }
```

8- Criar a abstração
Room do banco de dados

```
MainActivity.kt x
27     val db :AppDatabase = Room.databaseBuilder(
28         applicationContext,
29         AppDatabase::class.java, name: "db_contatos").build()
30     dao = db.getContatoDAO()
31     binding.btnSalvar.setOnClickListener { it: View!
32
33         var contato = Contato(
34             id: 0,
35             binding.txtNome.text.toString(),
36             binding.txtEmail.text.toString(),
37             binding.txtFone.text.toString()
38         )
39         binding.txtNome.setText("")
40         binding.txtEmail.setText("")
41         binding.txtFone.setText("")
42         runBlocking { this: CoroutineScope
43             dao.inserir(contato)
44
45             val lista :List<Contato> = dao.listar()
46             for(contato :Contato in lista) {
47
48                 Log.i( tag: "MIT", msg: "${contato.id} - ${contato.nome}")
49             }
50         }
51     }
```