Fundamentos de Desenvolvimento Android

MIT em Desenvolvimento Mobile - 2020

ViewModel Factory

ViewModel Factory

Não podemos criar ViewModel por conta própria, precisamos de ViewModelProviders para criar ViewModels.

Mas ViewModelProviders só podem instanciar ViewModels sem nenhum argumento de construtor.

Se houver um ViewModel com um ou vários argumentos, será preciso usar um Factory Method que possa passar para ViewModelProviders juntamente com os argumentos necessários.

Factory Method é um design pattern que usa métodos para criar objetos.

Um Factory Method é um método que retorna uma instância da mesma classe.

MainActivityViewModel.kt

```kotlin
class MainActivityViewModel(application: Application) : AndroidViewModel(application) {


}

class MainActivityViewModelFactory(private val application: Application) : ViewModelProvider.Factory {

    override fun <T : ViewModel?> create(modelClass: Class<T>): T {

        if (modelClass.isAssignableFrom(MainActivityViewModel::class.java)) {

            return MainActivityViewModel(application) as T
        }
        throw IllegalArgumentException("ViewModel não é compatível com essa Factory")
    }
}
```

1

2

ViewModel sensível ao contexto da aplicação

3

4

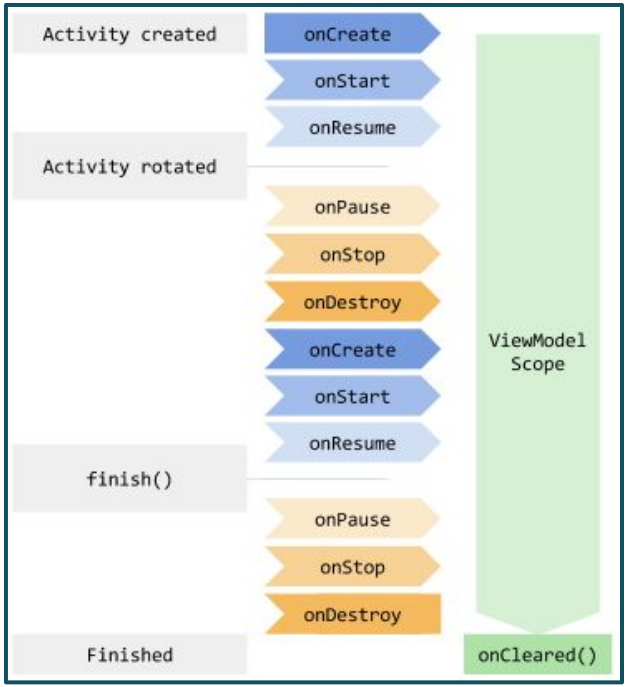
Teste para verificar se é a ViewModel correta

ViewModel X ViewModelFactory

ViewModel é uma classe responsável por preparar e gerenciar os dados de um Activity ou Fragment. Ela lida com a comunicação da Activity ou Fragment com o resto do aplicativo.

Um ViewModel é sempre criado em associação com um escopo (Activity ou Fragment) e será retido enquanto o escopo estiver ativo.

> Isso significa que um ViewModel não será destruído se seu proprietário for destruído por uma alteração de configuração. A nova instância do proprietário apenas será reconectada ao ViewModel existente.

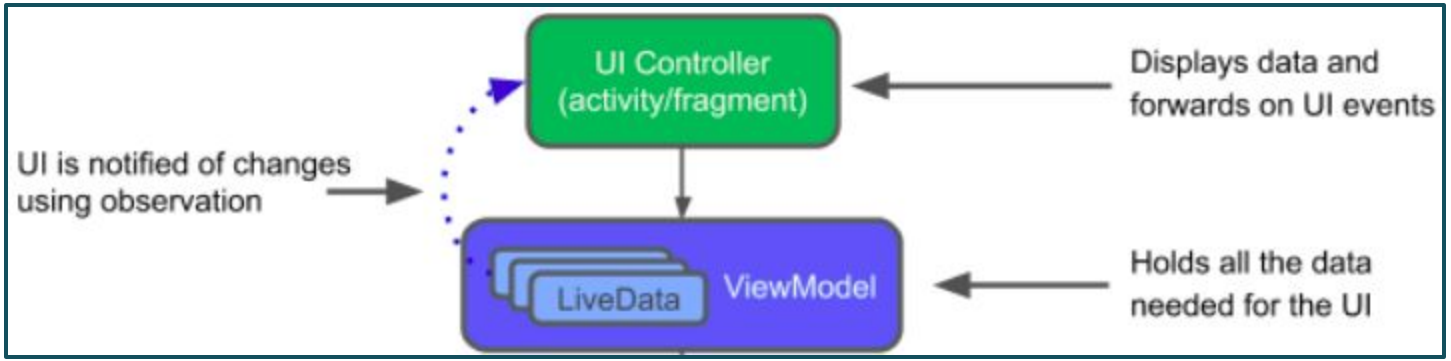


ViewModel X ViewModelFactory

O objetivo do ViewModel é adquirir e manter as informações necessárias para uma Activity ou Fragment.

A Activity ou Fragment deve ser capaz de observar as mudanças no ViewModel.

ViewModels geralmente expõem essas informações por meio do LiveData.



MainActivity.kt

```
18    override fun onCreate(savedInstanceState: Bundle?) {
19
20        super.onCreate(savedInstanceState)
21        //setContentView(R.layout.activity_main)
22        binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_main)
23        viewmodel = ViewModelProvider( owner: this, MainActivityViewModelFactory(this.application))
24            .get(MainActivityViewModel::class.java)
25        binding.viewmodel = viewmodel
26        binding.lifecycleOwner = this
```

1

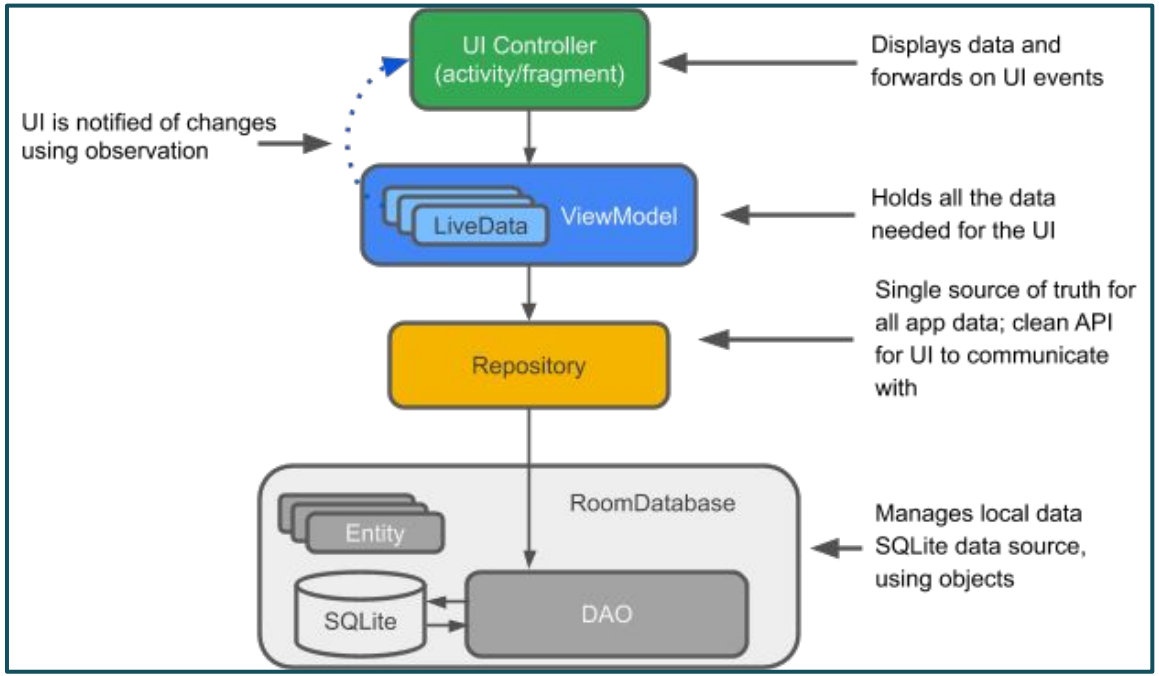
Repository

Repository

Repository é um design patterns que isola fontes de dados do resto do aplicativo.

Um repositório intermedia fontes de dados (como modelos persistentes, serviços da web e caches) e o resto do aplicativo.

O diagrama mostra como os componentes do aplicativo que usam LiveData podem interagir com fontes de dados por meio de um repositório.



ContatoDAO.kt

```kotlin
@Dao
interface ContatoDAO {

    @Query(value = "SELECT * FROM contatos WHERE id= :id")
    suspend fun obter(id : Int): Contato

    @Query(value = "SELECT * FROM contatos")
    suspend fun listar(): List<Contato>

    @Query(value = "SELECT * FROM contatos")
    fun listarLiveData(): Flow<List<Contato>>

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun inserir(contato: Contato)

    @Delete
    suspend fun excluir(contato: Contato)
}
```

ContatoDAO.kt | ContatoRepository.kt

```
10  class ContatoRepository(applicationContext : Application) {
11
12      private lateinit var dao : ContatoDAO
13
14      init {
15
16          val db :AppDatabase = Room.databaseBuilder(applicationContext, AppDatabase::class.java,
17              name: "db_contatos").build()
18          dao = db.getContatoDAO()
19      }
```

1

2

ContatoDAO.kt | ContatoRepository.kt

```
21      fun obterPorId(id : Int) : Contato {
22
23          return runBlocking { this: CoroutineScope
24              return@runBlocking dao.obter(id)
25          }
26      }
27
28      fun listarContatosLiveData() : Flow<List<Contato>> {
29
30          return runBlocking { this: CoroutineScope
31              return@runBlocking dao.listarLiveData()
32          }
33      }
```

1

2

ContatoDAO.kt | ContatoRepository.kt

```
35      fun salvarContato(contato : Contato) {
36
37          runBlocking { this: CoroutineScope
38
39              if(contato.id == 0) {
40
41                  dao.inserir(contato)
42              }
43          }
44      }
45
46      fun excluirContato(contato : Contato) {
47
48          runBlocking { this: CoroutineScope
49
50              dao.excluir(contato)
51          }
52      }
```

3

4

MainActivityViewModel.kt

```
7   class MainActivityViewModel(application: Application) : AndroidViewModel(application) {
8
9       private lateinit var contatoRepository: ContatoRepository
10      lateinit var contatos: LiveData<List<Contato>>
11      lateinit var contato : MutableLiveData<Contato>
12
13      init {
14
15          contatoRepository = ContatoRepository(application)
16          contatos = contatoRepository.listarContatosLiveData().asLiveData()
17          contato = MutableLiveData(Contato( id: 0, nome: "", email: "", fone: ""))
18      }
```

1

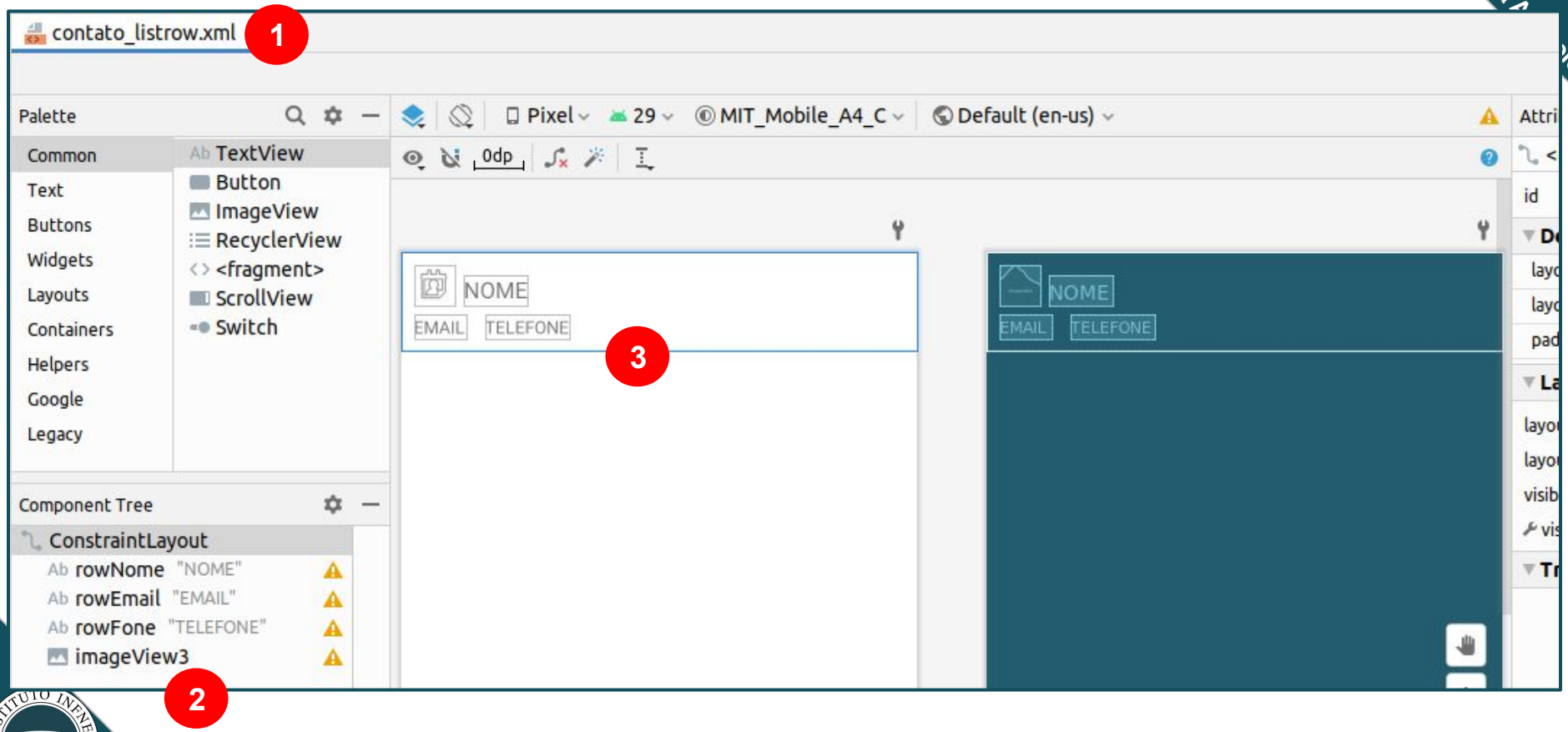
2

MainActivityViewModel.kt

```
20      fun obterPorId(id : Int) : Contato {
21
22          return contatoRepository.obterPorId(id)
23      }
24
25      fun salvarContato(contato:Contato) {
26
27          contatoRepository.salvarContato(contato)
28      }
29
30      fun excluirContato() {
31
32          if(contato.value != null) {
33
34              contatoRepository.excluirContato(contato.value!!)
35          }
36      }
37  }
```

1
2
3

RecyclerView


contato_listrow.xml
1
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
Legacy
TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch
Component Tree
ConstraintLayout
rowNome "NOME"
rowEmail "EMAIL"
rowFone "TELEFONE"
imageView3
2
Pixel
29
MIT_Mobile_A4_C
Default (en-us)
0dp
NOME
EMAIL
TELEFONE
3
NOME
EMAIL
TELEFONE
Attri
id

ListaContatoAdapter.kt

```
10  class ListaContatoAdapter() : RecyclerView.Adapter<ListaContatoAdapter.ViewHolder>() {
11
12      class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
13
14      }
15  }
```

1

2

RecyclerViewItemListener.kt

```
5  interface RecyclerViewItemListener {
6
7      fun recyclerViewItemClicked(view : View, id : Int)
8  }
```

ListaContatoAdapter.kt

```
10 class ListaContatoAdapter() : RecyclerView.Adapter<ListaContatoAdapter.ViewHolder>() {
11
12     var listaContatos = listOf<Contato>()
13         set(value) {
14             field = value
15             notifyDataSetChanged()
16         }
17     lateinit var itemListener : RecyclerViewItemListener
18
19     fun setRecyclerViewItemListener(listener: RecyclerViewItemListener) {
20
21         itemListener = listener
22     }
```

1

2

ListaContatoAdapter.kt

```
24  override fun onCreateViewHolder(
25      parent: ViewGroup,
26      viewType: Int
27  ): ListaContatoAdapter.ViewHolder {
28
29      val view :View! = LayoutInflater
30          .from(parent.context)
31          .inflate(R.layout.contato_listrow, parent, attachToRoot: false)
32      return ListaContatoAdapter.ViewHolder(view)
33  }
34
35  override fun onBindViewHolder(holder: ListaContatoAdapter.ViewHolder, position: Int) {
36
37      holder.bindItem(listaContatos[position], itemListener, position)
38  }
39
40  override fun getItemCount(): Int {
41      return listaContatos.size
42  }
```

1

2

3

ListaContatoAdapter.kt

```
44      class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
45
46  1       fun bindItem(contato: Contato, itemListener: RecyclerViewItemListener, position: Int) {
47
48          val rowNome :TextView! = itemView.findViewById<TextView>(R.id.rowNome)
49          rowNome.setText(contato.nome)
50          val rowEmail :TextView! = itemView.findViewById<TextView>(R.id.rowEmail)
51          rowEmail.setText(contato.email)
52          val rowFone :TextView! = itemView.findViewById<TextView>(R.id.rowFone)
53          rowFone.setText(contato.fone)
54
55          itemView.setOnClickListener { it: View!
56  2
57              itemListener.recyclerViewItemClicked(it, contato.id)
58          }
59      }
60  }
```

MainActivity.kt

```
32      1 binding.lstContatos.layoutManager = LinearLayoutManager( context: this)
33        //binding.lstContatos.layoutManager = GridLayoutManager(this, 2)
34        val adapter = ListaContatoAdapter()
35        adapter.setRecyclerViewItemListener(this)
36        binding.lstContatos.adapter = adapter
37      2 if(viewmodel.contatos.value != null) {
38
39            adapter.listaContatos = viewmodel.contatos.value!!
40        }
41      3 viewmodel.contatos.observe( owner: this, Observer { it: List<Contato>!
42
43            it.let { it: List<Contato>!
44
45                adapter.listaContatos = it
46            }
47        })
```

MainActivity.kt

```
49      binding.btnSalvar.setOnClickListener { it: View!
50
51          var contato = Contato(
52              id: 0,
53              binding.txtNome.text.toString(),
54              binding.txtEmail.text.toString(),
55              binding.txtFone.text.toString()
56          )
57          viewmodel.salvarContato(contato)
58          binding.txtNome.setText("")
59          binding.txtEmail.setText("")
60          binding.txtFone.setText("")
61      }
62
63      binding.btnExcluir.setOnClickListener{ it: View!
64
65          viewmodel.excluirContato()
66          binding.txtNome.setText("")
67          binding.txtEmail.setText("")
68          binding.txtFone.setText("")
69      }
```

1

2

MainActivity.kt

```
72      override fun recyclerViewItemClicked(view: View, id : Int) {
73
74          viewmodel.contato.value = viewmodel.obterPorId(id)
75      }
```

activity_main.xml

```
52
53          <EditText
54              android:id="@+id/txtNome"
55              android:layout_width="match_parent"
56              android:layout_height="wrap_content"
57              android:layout_weight="1"
58              android:ems="10"
59              android:inputType="textPersonName"
60              android:text="@={viewmodel.contato.nome}"
61              />
62      </LinearLayout>
```

1

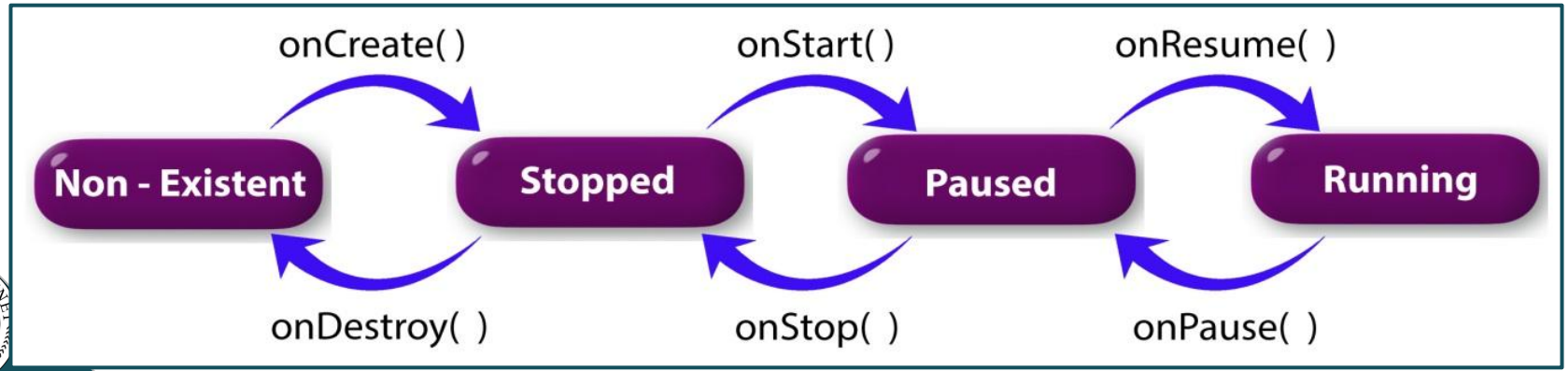
Navegação em Activities

Ciclo de Vida da Activity

Activities no sistema são gerenciadas como pilhas.

Quando uma nova Activity começa, ela é colocada no topo da pilha e se torna a atividade em execução - a atividade anterior sempre permanece abaixo da nova na pilha e não retornará ao primeiro plano até que a nova atividade saia.

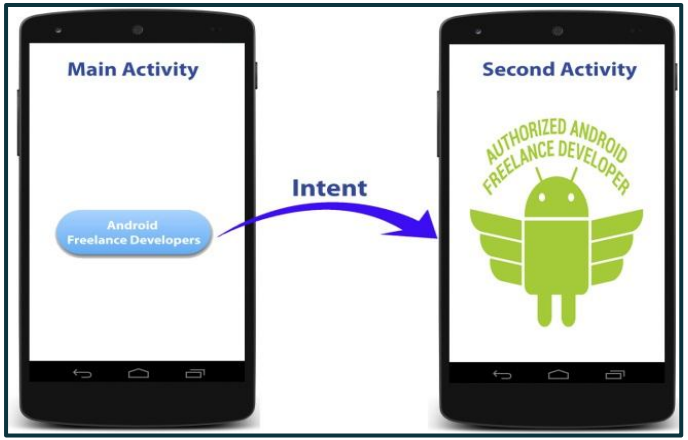
Uma atividade tem essencialmente quatro estados:



Android Intent

Os componentes do aplicativo Android podem se conectar a outros aplicativos Android.

Esta conexão é baseada em uma descrição de tarefa representada por um objeto **Intent**.



Android Intent é uma descrição de classe abstrata de uma operação a ser executada.

Intents são mensagens assíncronas que permitem que componentes do aplicativo solicitem funcionalidade de outros componentes do Android.

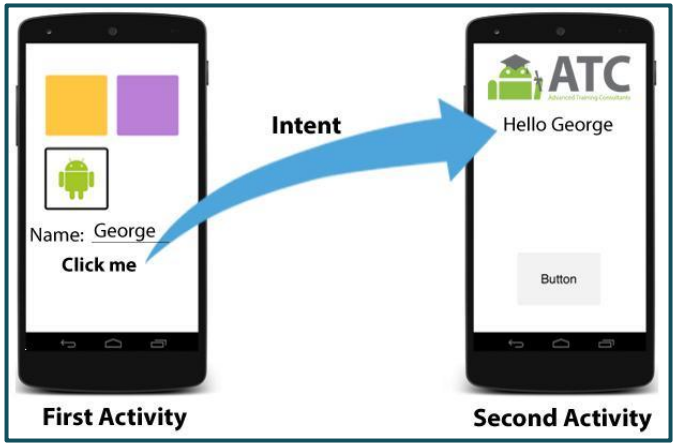
Os intents permitem que você interaja com componentes dos mesmos aplicativos, bem como com componentes fornecidos por outros aplicativos.

Android Intent

É muito provável que você precise transferir alguns dados para a atividade que deseja iniciar por meio do Intent.

O Android SDK fornece essa opção usando métodos extras.

Você pode anexar dados ao seu intent usando o método **putExtra()** do Intent para incluir dados extras na atividade de chamada ou o método **getExtra()** para recuperar dados da atividade chamada; por exemplo, o envio de um dado da primeira atividade para a segunda atividade quando ela começa.



Um objeto Intent é um pacote de informações usado pelo componente que recebe o intent e também as informações usadas pelo sistema Android.

MainActivity.kt

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

        val btnTela2 :Button!  = this.findViewById<Button>(R.id.btnTela2)
        btnTela2.setOnClickListener { it: View!

1       val intent = Intent( packageContext: this, MainActivity2::class.java)
        val txtFrase :EditText!  = this.findViewById<EditText>(R.id.txtFrase)
        intent.putExtra( name: "frase", txtFrase.text.toString())
        startActivity(intent)
        }
    }
}
```

MainActivity.kt | MainActivity2.kt

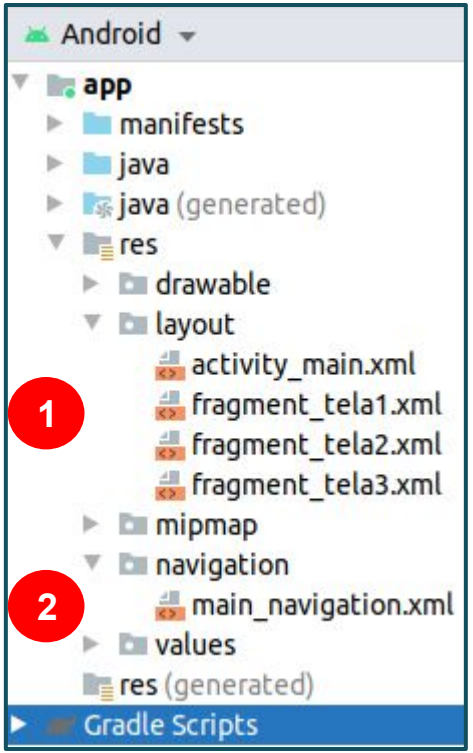
```
7  class MainActivity2 : AppCompatActivity() {
8      override fun onCreate(savedInstanceState: Bundle?) {
9          super.onCreate(savedInstanceState)
10         setContentView(R.layout.activity_main2)
11
12         val frase :String? = intent.getStringExtra( name: "frase")
13         val lblFrase :TextView! = this.findViewById<TextView>(R.id.lblFrase)
14         lblFrase.text = frase
15     }
16 }
```

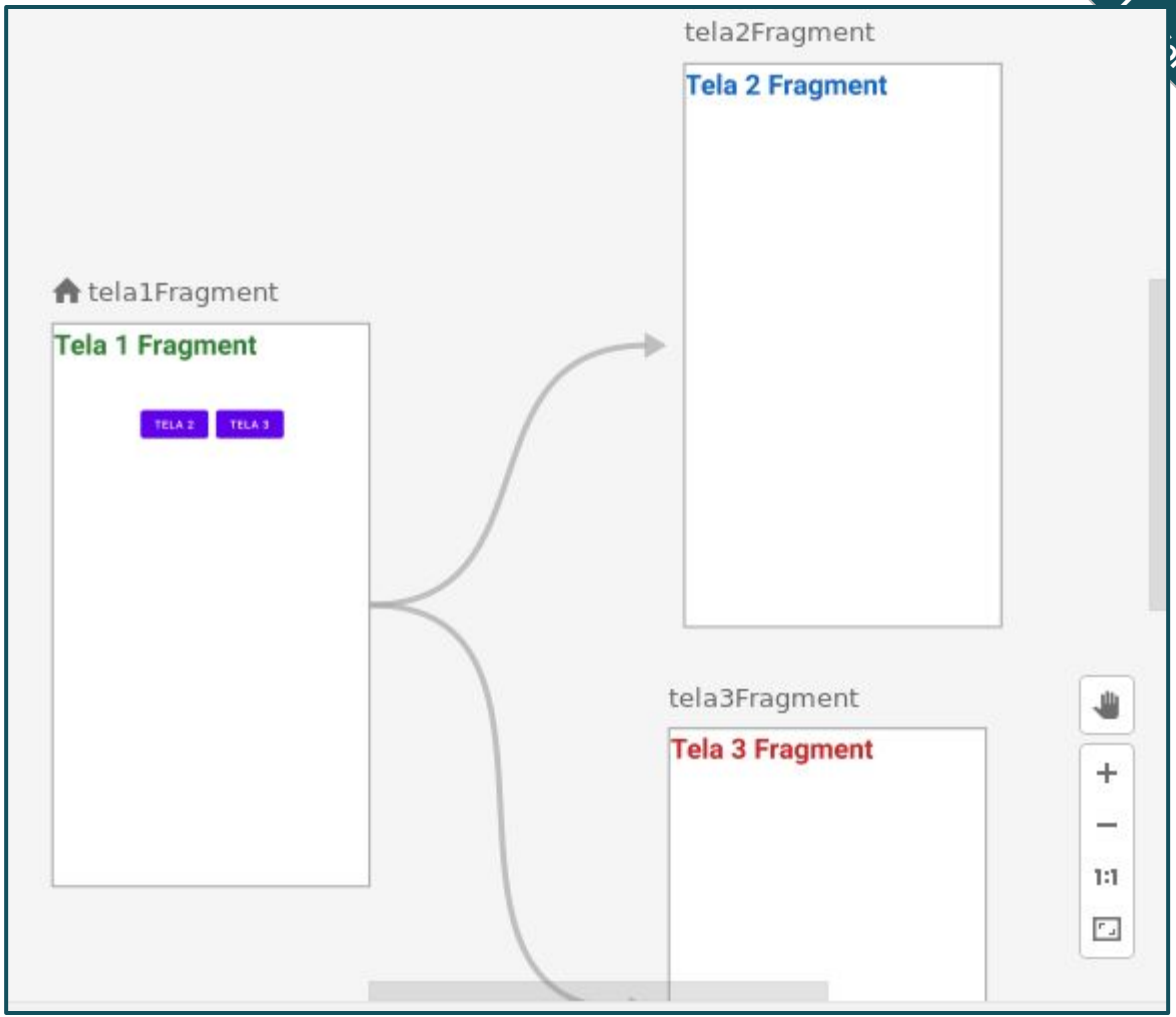
1

Navegar em Fragments

build.gradle (:app)

```
46      //-------------------------------------------------------------------------
47      //Navigation
48      def navigationVersion = "2.3.0"
49      implementation "androidx.navigation:navigation-fragment-ktx:$navigationVersion"
50      implementation "androidx.navigation:navigation-ui-ktx:$navigationVersion"
51  }
```


Android
app
manifests
java
java (generated)
res
drawable
layout
activity_main.xml
fragment_tela1.xml
fragment_tela2.xml
fragment_tela3.xml
mipmap
navigation
main_navigation.xml
values
res (generated)
Gradle Scripts
1
2


tela2Fragment
Tela 2 Fragment
tela1Fragment
Tela 1 Fragment
TELA 2
TELA 3
tela3Fragment
Tela 3 Fragment
1:1

Tela1Fragment.kt

```kotlin
class Tela1Fragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {

        val view : View! = inflater.inflate(R.layout.fragment_tela1, container, attachToRoot: false)

        val btnTela2 : Button! = view.findViewById<Button>(R.id.btnTela2)
        btnTela2.setOnClickListener { it: View!

            val navController : NavController = findNavController( fragment: this)
            val bundle = Bundle()
            bundle.putString("teste", "Instituto Infnet")
            navController.navigate(R.id.action_tela1Fragment_to_tela2Fragment, bundle)
        }
        val btnTela3 : Button! = view.findViewById<Button>(R.id.btnTela3)
        btnTela3.setOnClickListener { it: View!

            val navController : NavController = findNavController( fragment: this)
            navController.navigate(R.id.action_tela1Fragment_to_tela3Fragment)
        }
        return view
    }
}
```

1

2

3

Tela2Fragment.kt

```
10  class Tela2Fragment : Fragment() {
11
12      override fun onCreateView(
13          inflater: LayoutInflater, container: ViewGroup?,
14          savedInstanceState: Bundle?
15      ): View? {
16
17          val frase :String  = arguments?.get("teste").toString()
18          Log.i( tag: "MIT", frase)
19          return inflater.inflate(R.layout.fragment_tela2, container,  attachToRoot: false)
20      }
21  }
```