Fundamentos de Desenvolvimento Android

MIT em Desenvolvimento Mobile - 2020

INSTITUTO INFNET 1994

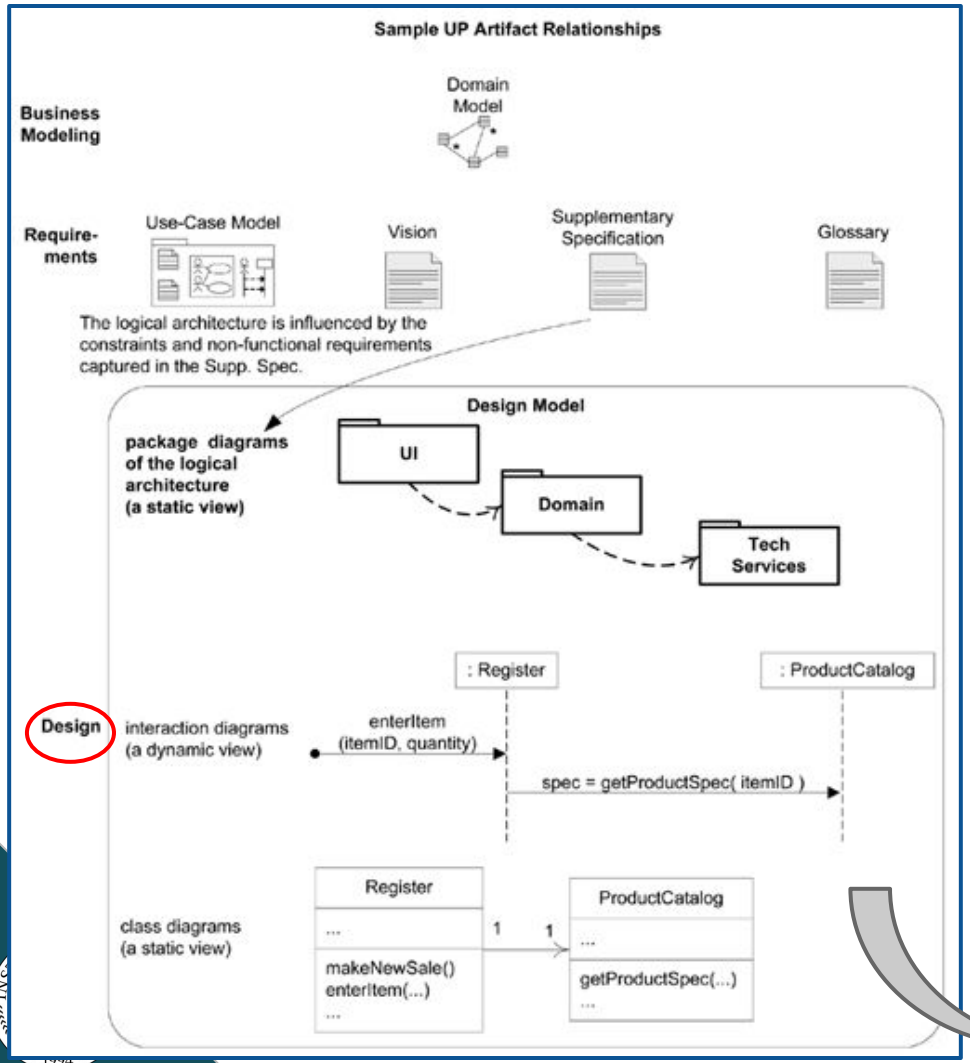
Arquitetura Lógica

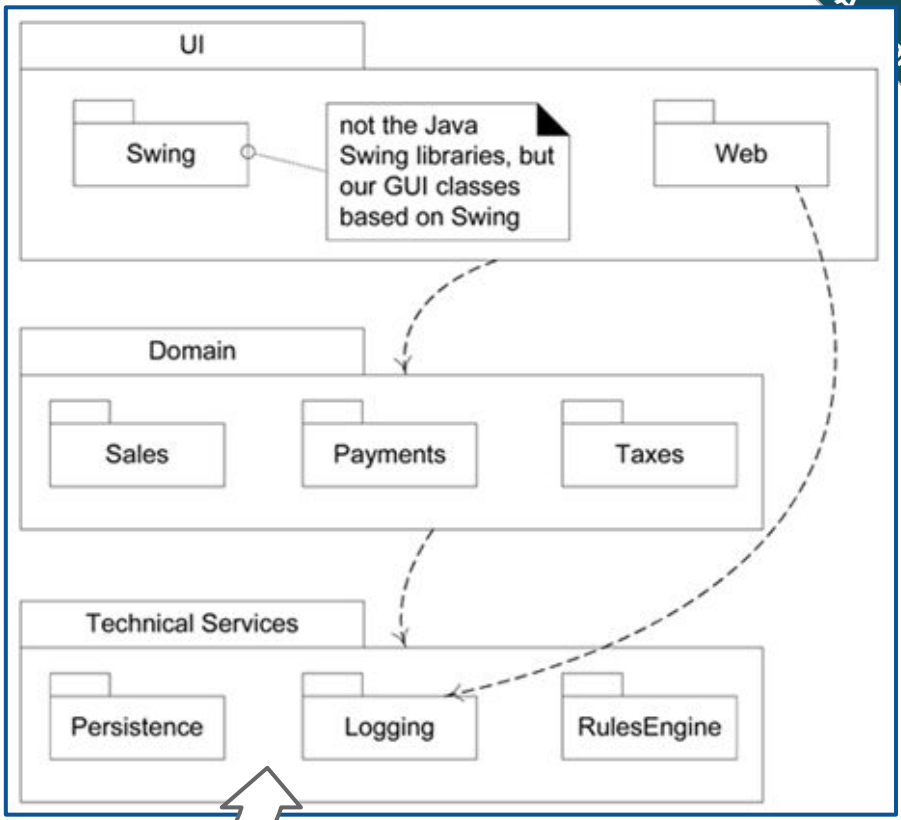
Arquitetura Lógica

O projeto de um sistema OO típico é baseado em várias camadas de arquitetura, como uma camada de IU, uma camada de lógica de aplicativo (ou "domínio") e assim por diante.

Os diagramas de pacotes da UML podem ilustrar a Arquitetura Lógica como parte do **modelo de projeto** e também serem resumidos como uma visualização no Documento de Arquitetura de Software.

A Arquitetura Lógica define os pacotes dentro dos quais as classes de software são definidas.


Sample UP Artifact Relationships
Business Modeling
Domain Model
Require-ments
Use-Case Model
Vision
Supplementary Specification
Glossary
The logical architecture is influenced by the constraints and non-functional requirements captured in the Supp. Spec.
Design Model
package diagrams of the logical architecture (a static view)
UI
Domain
Tech Services
Design
interaction diagrams (a dynamic view)
: Register
: ProductCatalog
enterItem (itemID, quantity)
spec = getProductSpec(itemID)
class diagrams (a static view)
Register
...
makeNewSale()
enterItem(...)
...
1
1
ProductCatalog
...
getProductSpec(...)
...


UI
Swing
not the Java Swing libraries, but our GUI classes based on Swing
Web
Domain
Sales
Payments
Taxes
Technical Services
Persistence
Logging
RulesEngine

O Que é a Arquitetura Lógica? E Camadas?

A arquitetura lógica é a organização em grande escala das classes de software em pacotes (ou namespaces), subsistemas e camadas.

É chamada de arquitetura lógica porque não há decisão sobre como esses elementos são implantados em diferentes processos do sistema operacional ou em computadores físicos em uma rede (essas últimas decisões fazem parte da arquitetura de implantação / arquitetura física).

O Que é a Arquitetura Lógica? E Camadas?

Uma camada é um agrupamento com alta **granularidade** (grain) de classes, pacotes ou subsistemas que tem responsabilidade **coesa** por um aspecto principal do sistema.

Além disso, as camadas são organizadas de forma que as camadas "superiores" (como a camada de IU) solicitem serviços de camadas "inferiores", mas normalmente não o contrário.

O Que é a Arquitetura Lógica? E Camadas?

Normalmente, as camadas em um sistema OO incluem:

- **Interface do usuário**.
- **Lógica de aplicativo (lógica de negócios)** e objetos de domínio — objetos de software que representam conceitos de domínio (por exemplo, uma venda de classe de software) que atendem aos requisitos do aplicativo, como o cálculo de um total de vendas.
- **Serviços técnicos** — objetos de uso geral e subsistemas que fornecem serviços técnicos de suporte, como interface com um banco de dados ou registro de erros. Esses serviços geralmente são independentes do aplicativo e reutilizáveis em vários sistemas.

O Que é a Arquitetura Lógica? E Camadas?

Em uma arquitetura estritamente em camadas, uma camada apenas chama os serviços da camada diretamente abaixo dela.

Esse design é comum em pilhas de protocolo de rede, mas não em sistemas de informação, que geralmente têm **uma arquitetura mais liberal**, na qual uma camada superior chama várias camadas inferiores.

Por exemplo, a camada de IU pode chamar sua camada lógica de aplicativo diretamente subordinada e também elementos de uma camada de serviço técnico inferior para registro e assim por diante.

O que é Arquitetura de Software?

Uma arquitetura é o conjunto de **decisões significativas** sobre:

- **organização de um sistema de software (modularização)**.
- a seleção dos **elementos estruturais** e suas interfaces pelas quais o sistema é composto (pacotes e classes).
- seu **comportamento** conforme especificado nas colaborações entre esses elementos (caso de uso).
- a composição desses elementos estruturais e elementos comportamentais em subsistemas **progressivamente maiores**.
- **estilo arquitetônico** que orienta esta organização - esses elementos e suas interfaces, suas colaborações e sua composição.

Aplicando UML: Diagramas de Pacote

OS diagramas de pacote UML são freqüentemente usados para ilustrar a arquitetura lógica de um sistema - as camadas, subsistemas, pacotes (no sentido de Java), etc.

Um DIAGRAMA DE pacote UML fornece uma maneira de agrupar elementos.

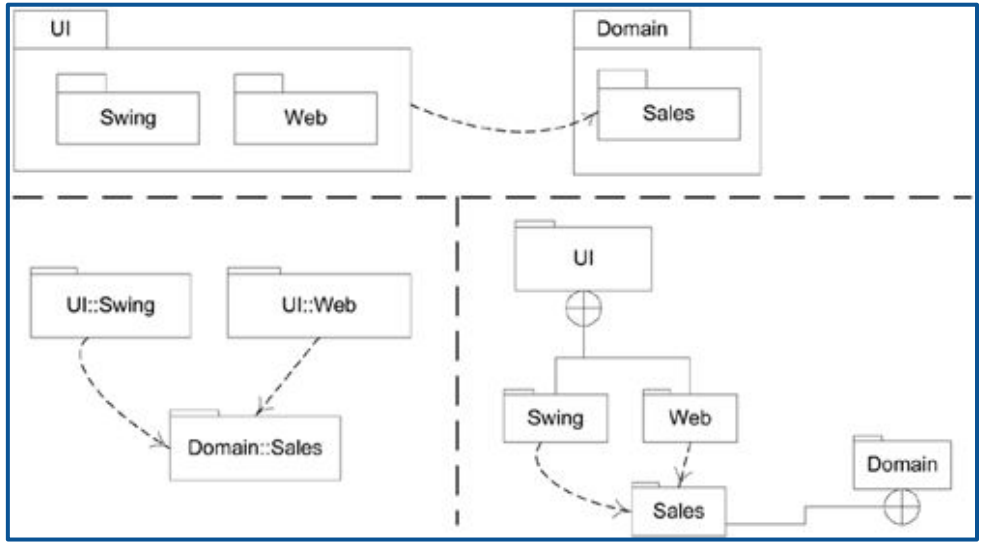
Um pacote UML pode agrupar qualquer coisa: classes, outros pacotes, casos de uso e assim por diante.

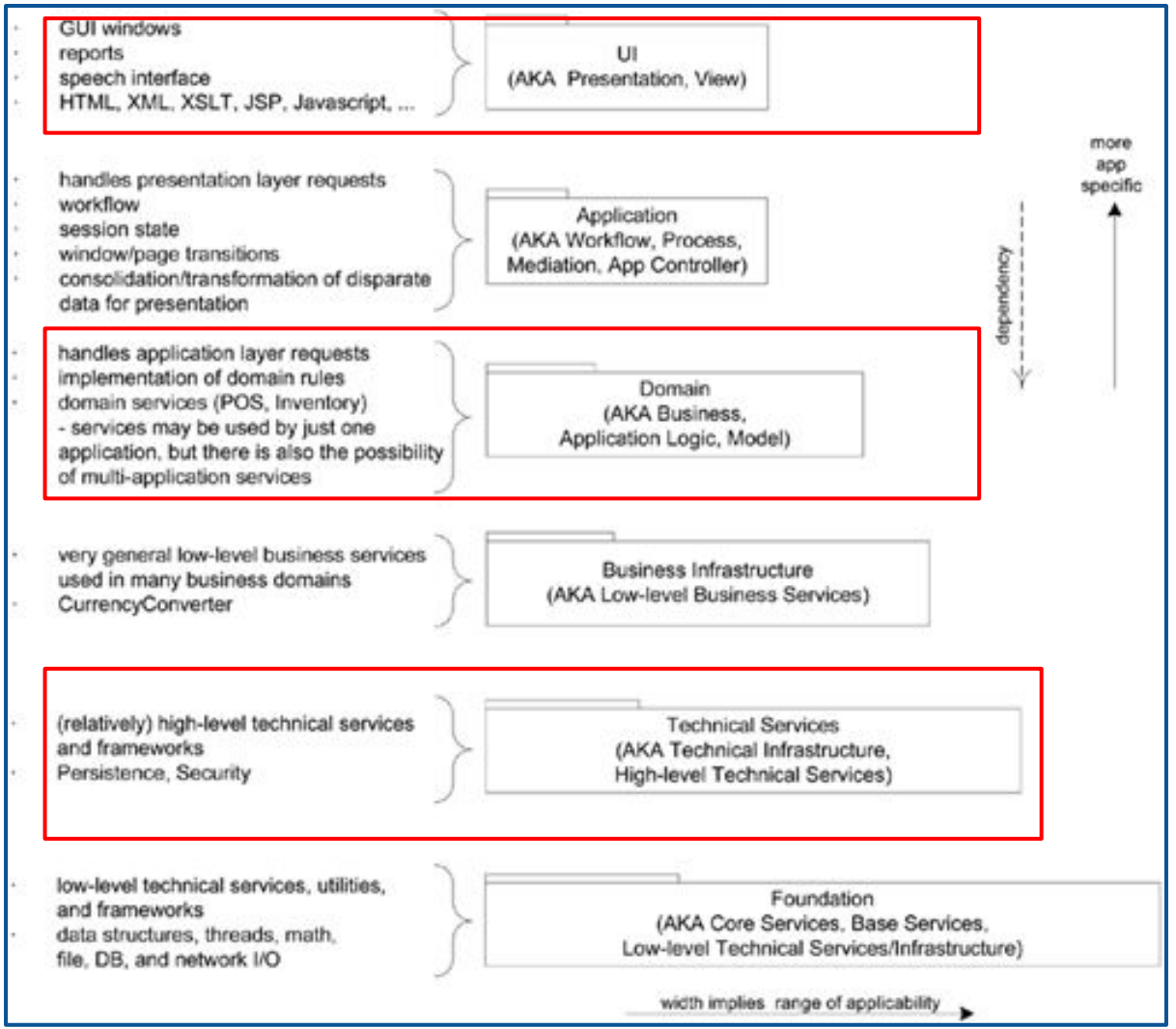
Pacotes de aninhamento são muito comuns. Um pacote UML é um conceito mais geral do que simplesmente um pacote Kotlin ou namespace .NET, embora um pacote UML possa representá-los - e ir além.

Aplicando UML : Diagramas de Pacote

É comum querer mostrar dependência (um acoplamento) entre pacotes para que os desenvolvedores possam ver o acoplamento em grande escala no sistema.

A linha de dependência UML é usada para isso, uma linha tracejada com a seta apontando para o pacote do qual se depende.





MIT Plataforma Android
INSTITUTO INFNET 1994

Separação de Responsabilidades / Interesses

As responsabilidades dos objetos em uma camada devem **estar fortemente relacionadas entre si** e não devem ser misturadas com responsabilidades de outras camadas (coesão).

Objetos de IU não devem fazer lógica de aplicativo.

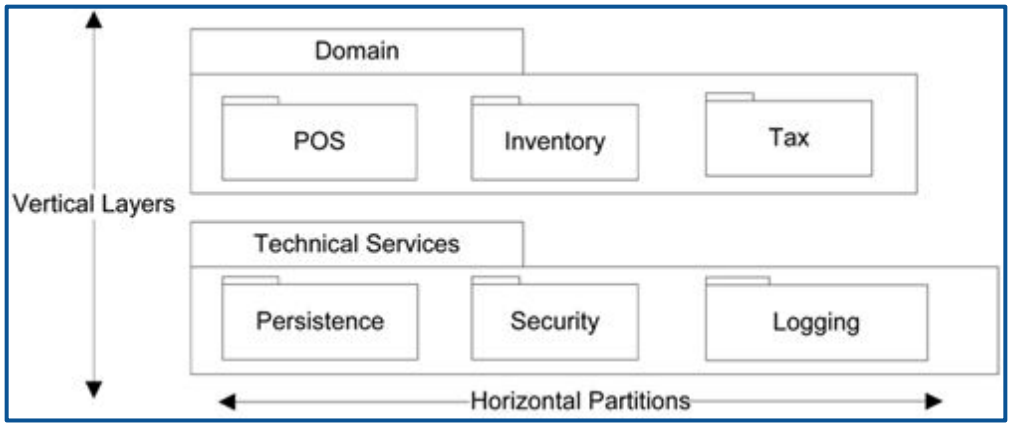
Por outro lado, as classes de lógica do aplicativo não devem capturar eventos de mouse ou teclado da IU.

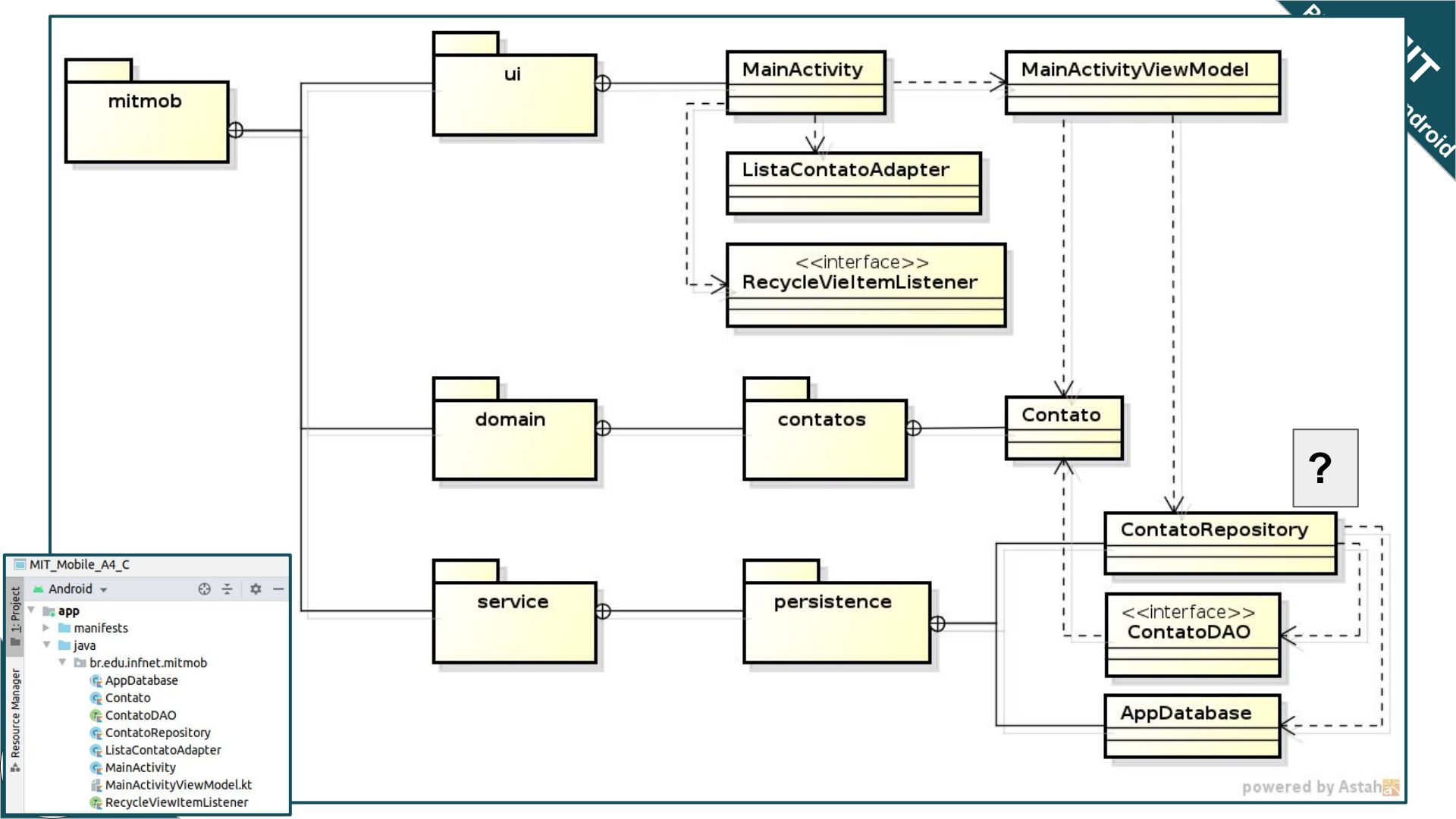
Objetivo Final: uma separação clara de interesses, mantendo uma alta coesão - princípios arquitetônicos básicos.

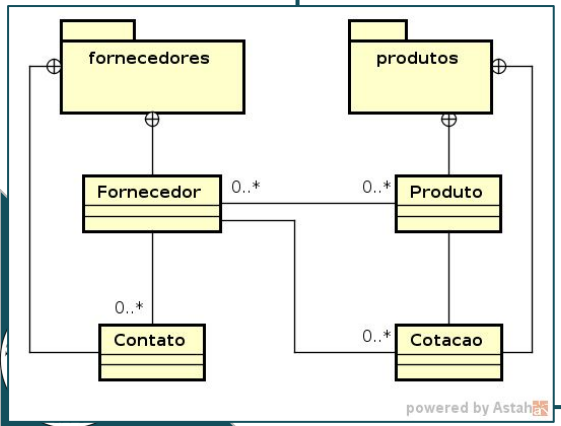
Camadas e Partições

As camadas de uma arquitetura representam as fatias verticais, enquanto as partições representam uma divisão horizontal de subsistemas relativamente paralelos de uma camada.

Por exemplo, a camada de Serviços Técnicos pode ser dividida em partições, como Segurança e Relatórios.




mitmob
ui
MainActivity
MainActivityViewModel
ListaContatoAdapter
<<interface>>
RecycleVieItemListener
domain
contatos
Contato
?
ContatoRepository
service
persistence
<<interface>>
ContatoDAO
AppDatabase
MIT_Mobile_A4_C
Android
app
manifests
java
br.edu.infnet.mitmob
AppDatabase
Contato
ContatoDAO
ContatoRepository
ListaContatoAdapter
MainActivity
MainActivityViewModel.kt
RecycleViewItemListener
1: Project
Resource Manager
powered by Astah



MainActivity

mitmob

ui

fornecedores

FornecedoresFragment

ContatosFragment

produtos

ProdutosFragment

ContacoesFragment

domain

fornecedores

Fornecedor

Contato

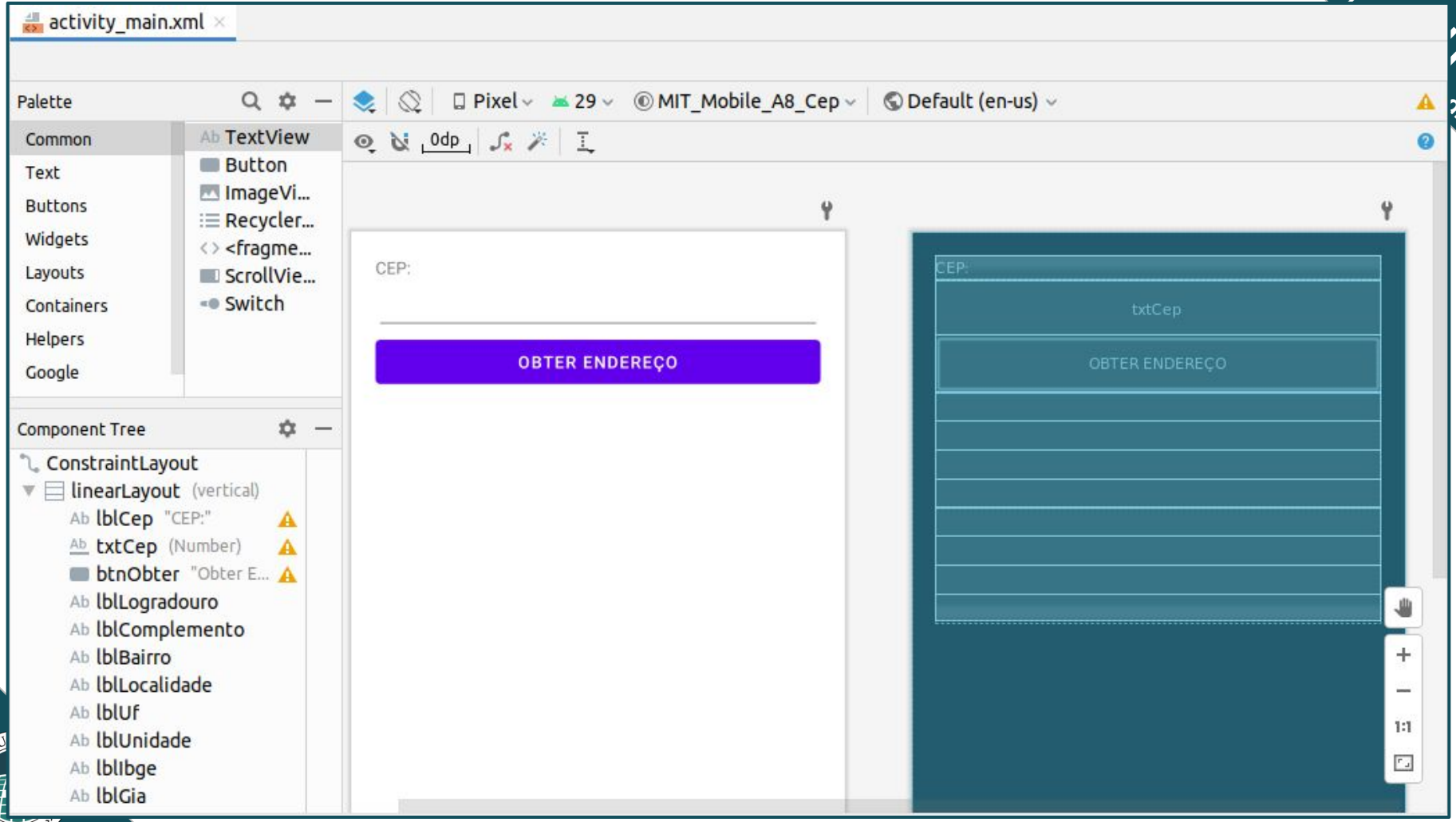
produtos

Produto

Cotacao

powered by Astah

Introdução REST


activity_main.xml
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
TextView
Button
ImageVi...
Recycler...
<fragme...
ScrollVie...
Switch
Component Tree
ConstraintLayout
linearLayout (vertical)
lblCep "CEP:"
txtCep (Number)
btnObter "Obter E...
lblLogradouro
lblComplemento
lblBairro
lblLocalidade
lblUf
lblUnidade
lblIbge
lblGia
Pixel
29
MIT_Mobile_A8_Cep
Default (en-us)
0dp
CEP:
OBTER ENDEREÇO
CEP:
txtCep
OBTER ENDEREÇO
1:1

build.gradle (:app)

You can use the Project Structure dialog to view and edit your project configuration — Open (Ctrl+Alt+Shift+S) — Hide

```
35  dependencies {
36
37      implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
38      implementation 'androidx.core:core-ktx:1.3.2'
39      implementation 'androidx.appcompat:appcompat:1.2.0'
40      implementation 'com.google.android.material:material:1.2.1'
41      implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
42      testImplementation 'junit:junit:4.+'
43      androidTestImplementation 'androidx.test.ext:junit:1.1.2'
44      androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
45      //-----------------------------------------------------------------------
46      //Bibliotecas do Retrofit
47      implementation 'com.squareup.retrofit2:retrofit:2.9.0'
48      implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
49  }
```

1

AndroidManifest.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="br.edu.infnet.mitmob">
4
5       <uses-permission android:name="android.permission.INTERNET" />
6
7       <application
8           android:allowBackup="true"
9           android:icon="@mipmap/ic_launcher"
10          android:label="MIT_Mobile_A8_Cep"
```

2

Endereco.kt

```kotlin
package br.edu.infnet.mitmob

class Endereco {

    var cep: String? = null
    var logradouro: String? = null
    var complemento: String? = null
    var bairro: String? = null
    var localidade: String? = null
    var uf: String? = null
    var unidade: String? = null
    var ibge: String? = null
    var gia: String? = null
}
```

1

CepAPI.kt

```kotlin
package br.edu.infnet.mitmob

import retrofit2.Call
import retrofit2.http.GET
import retrofit2.http.Path


interface CepAPI {

    @GET( value: "{CEP}/json/")
    fun obterEndereco(@Path( value: "CEP") CEP: String?): Call<Endereco?>?
}
```

2

CepServiceListener.kt

```kotlin
package br.edu.infnet.mitmob

interface CepServiceListener {

    fun obterEnderecoTerminou(endereco: Endereco?)

    fun falhaReportada(falha : String?)
}
```

3

CepService.kt

```kotlin
class CepService {

    private lateinit var api : CepAPI
    private lateinit var listener : CepServiceListener

    init {

        val retrofit = Retrofit.Builder()
            .baseUrl( baseUrl: "https://viacep.com.br/ws/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        api = retrofit.create(CepAPI::class.java)
    }

    public fun setCepServiceListener(listener: CepServiceListener) {

        this.listener = listener
    }
```

1

2

CepService.kt

```
29      public fun obterEndereco(cep : String) {
30
31          val call = api.obterEndereco(cep)
32          call!!.enqueue(object : Callback<Endereco?> {
33
34              override fun onResponse(call: Call<Endereco?>, response: Response<Endereco?>) {
35
36                  if(response.isSuccessful) {
37
38                      listener.obterEnderecoTerminou(response.body())
39                  }
40              }
41
42              override fun onFailure(call: Call<Endereco?>, t: Throwable) {
43
44                  listener.falhaReportada(t.message)
45              }
46          })
47      }
48  }
```

1
2
3

Parte 2

MainActivity.kt

```
1  class MainActivity : AppCompatActivity(), CepServiceListener {
12
13     private val servico = CepService()
14
15     override fun onCreate(savedInstanceState: Bundle?) {
16
17         super.onCreate(savedInstanceState)
18         setContentView(R.layout.activity_main)
19
20         servico.setCepServiceListener(this)
21
22         val btnObter = this.findViewById<Button>(R.id.btnObter)
23         btnObter.setOnClickListener {  it: View!
24
25             val txtCep = this.findViewById<EditText>(R.id.txtCep)
26             servico.obterEndereco(txtCep.text.toString())
27         }
28     }
```

1
2
3
4

Parte 1

MainActivity.kt

```kotlin
override fun obterEnderecoTerminou(endereco: Endereco?) {

    if(endereco != null) {

        this.findViewById<TextView>(R.id.lblBairro).setText(endereco.bairro)
        this.findViewById<TextView>(R.id.lblComplemento).setText(endereco.complemento)
        this.findViewById<TextView>(R.id.lblGia).setText(endereco.gia)
        this.findViewById<TextView>(R.id.lblIbge).setText(endereco.ibge)
        this.findViewById<TextView>(R.id.lblLocalidade).setText(endereco.localidade)
        this.findViewById<TextView>(R.id.lblLogradouro).setText(endereco.logradouro)
        this.findViewById<TextView>(R.id.lblUf).setText(endereco.uf)
        this.findViewById<TextView>(R.id.lblUnidade).setText(endereco.unidade)
    }
}

override fun falhaReportada(falha: String?) {

    Toast.makeText( context: this, falha, Toast.LENGTH_SHORT).show()
}
}
```

Parte 2