Explorando CameraX no Android

TDC Recife

Outubro 2019

Quem somos

Wellington Cabral

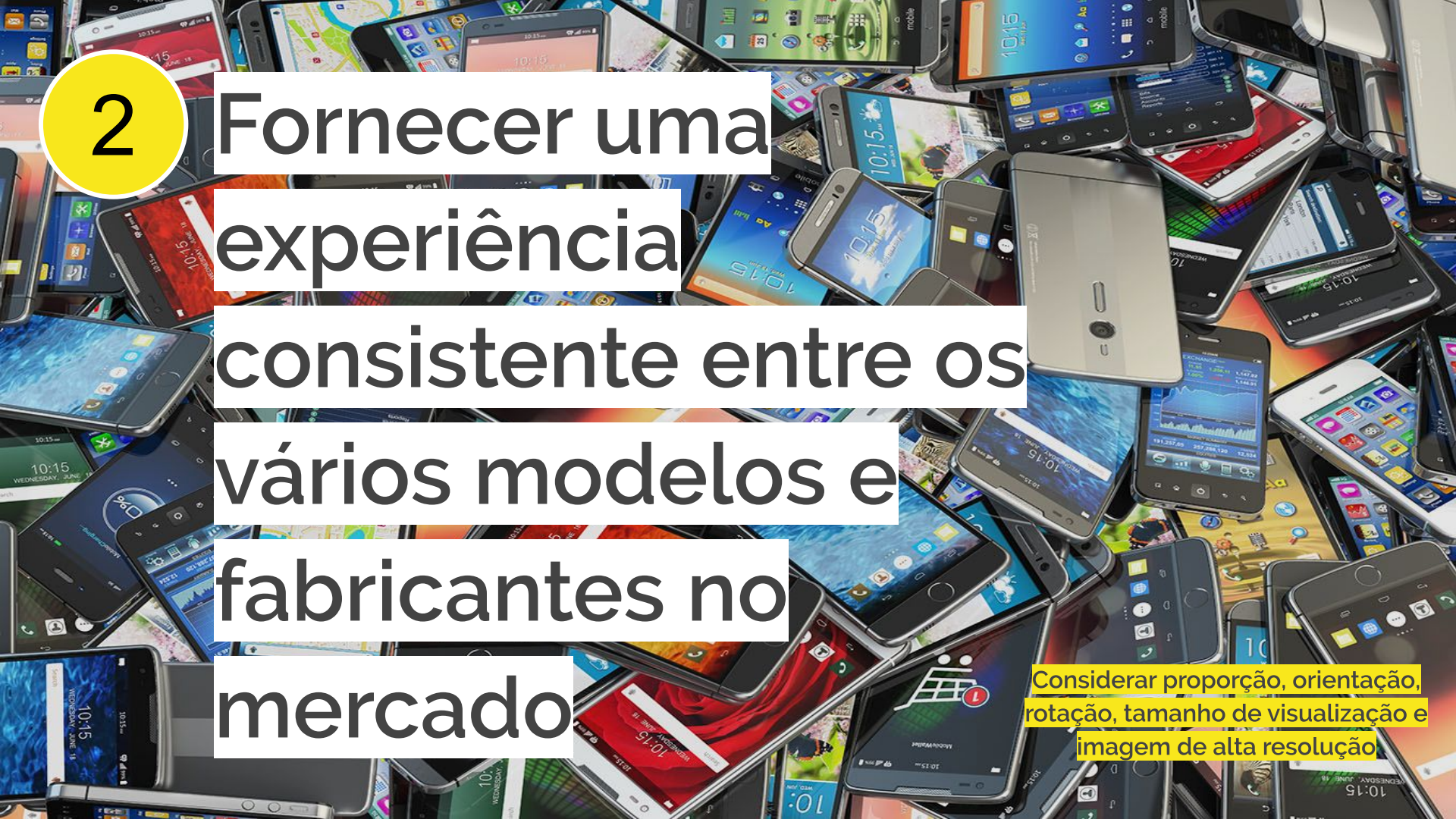
Engenheiro de software no CESAR e professor de Android na CESAR School

José Carlos Moura

Engenheiro de software no CESAR. Associated Android Developer

Por que é **difícil** desenvolver app para **câmera?**


1
Fragmentação do
Sistema Android



2

Fornecer uma experiência consistente entre os vários modelos e fabricantes no mercado

Considerar proporção, orientação, rotação, tamanho de visualização e imagem de alta resolução


3
Complexidade da
API de Câmera

API's Camera

Camera1

Deprecated desde o Lollipop

Simples, porém limitada

Camera2

Substituiu a Camera1

Poderosa, porém muito boilerplate

CameraX

Simplicidade e Abstração

Camera2 API

- Requer uma "tonelada" de código
- É muito complexo!
- Exige que você implemente e gerencie muitos estados
- Apresenta erros na parte da lanterna da câmera. Há muita confusão sobre as diferenças entre o modo “lanterna” e o modo “flash” na Camera2.
- Tratar bugs específicos do fornecedor

CameraX

Jetpack support library

Google I/O 2019

Open-source

Alpha

https://developer.android.com/training/camerax

CameraX

Compatibilidade

- Compatível com Android L
- Podendo ser utilizado em 90% dos celulares Android.
- Usa a API do camera2

Consistência

- Comportamento validado em diferentes dispositivos
- Testes realizados no Laboratório do Google

Usabilidade

- Abstração do gerenciamento da câmera
- Flexibilidade para customização
- Menos boilerplate

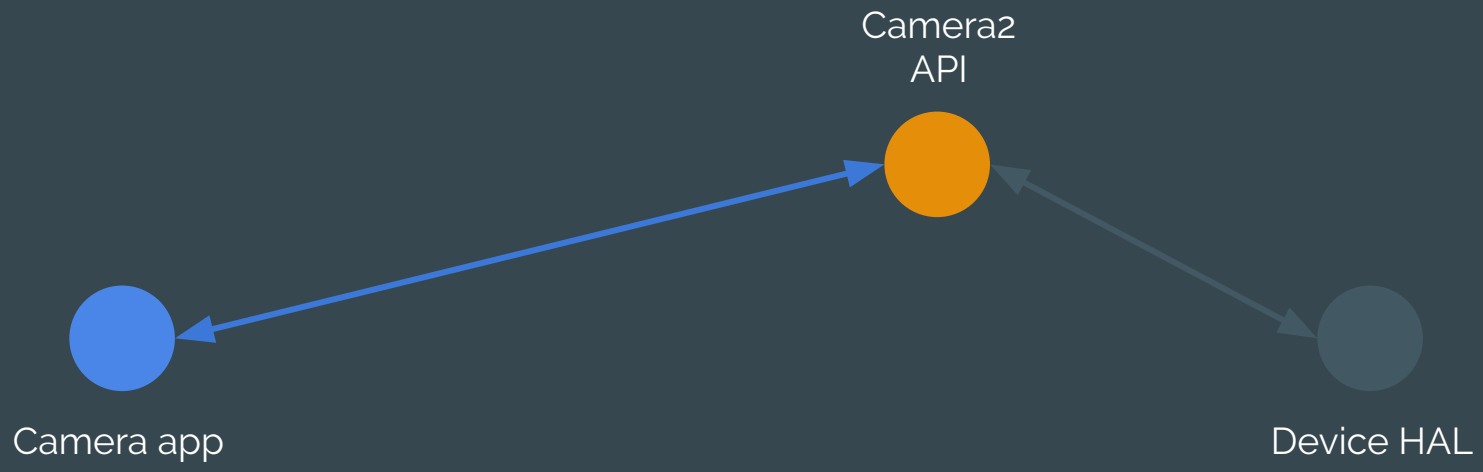
CameraX test lab

- Centenas de dispositivos testados
- Vários fabricantes
- Do Android L ao Q
- 24 por 7
- Vários tipos de testes:
  Orientação, rotação funcional, capturar imagem, integração, casos de uso end-to-end, performance e etc.

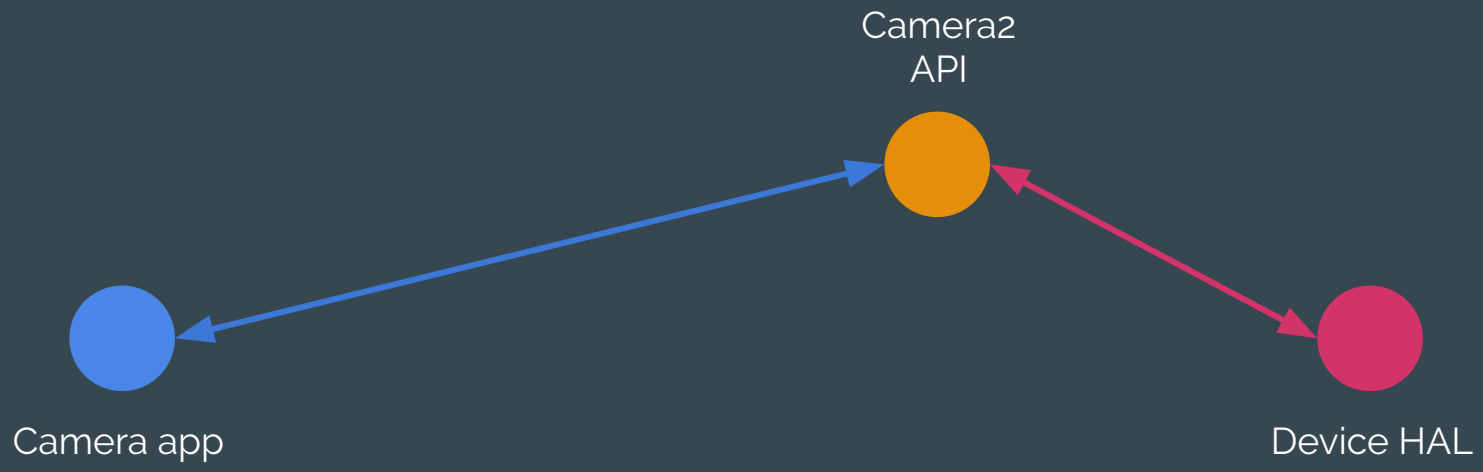
PROBLEMAS RESOLVIDOS

- Erros ao trocar camera front/back
- Otimização de camera closures
- Orientação incorreta
- Flash não é disparado

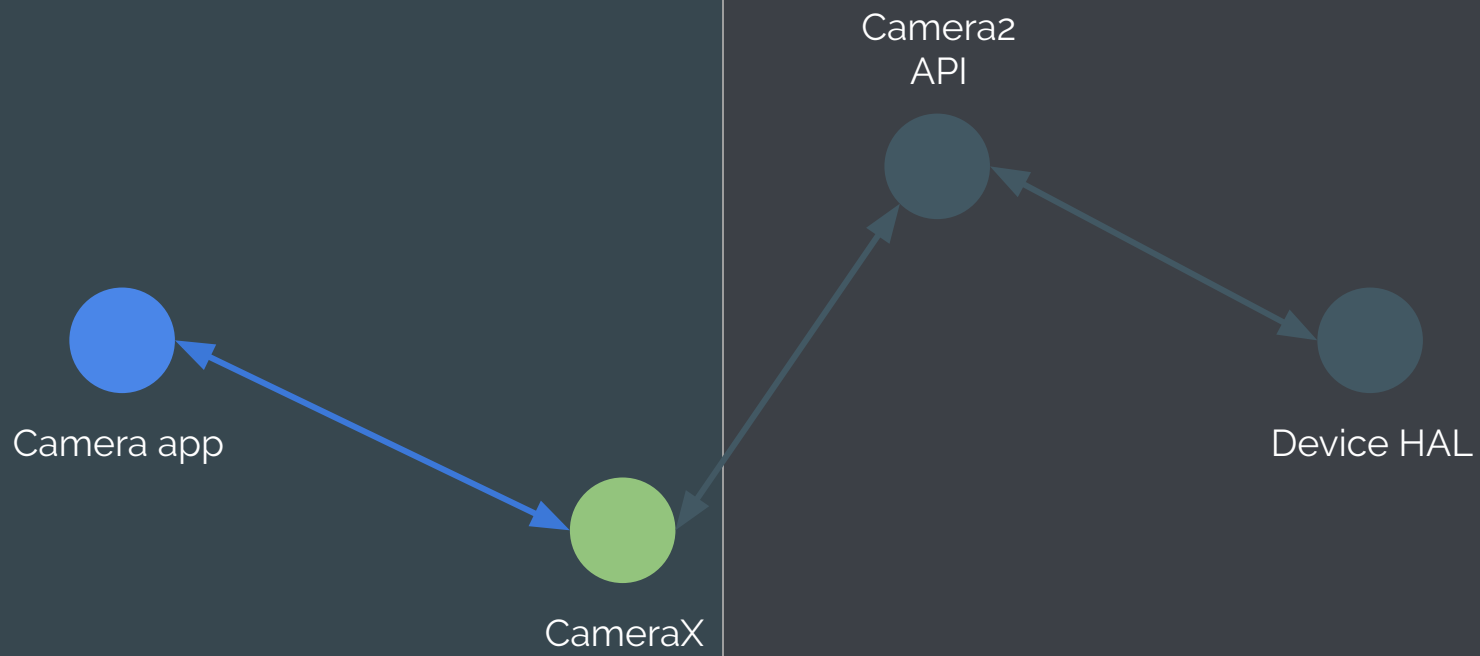
Processo



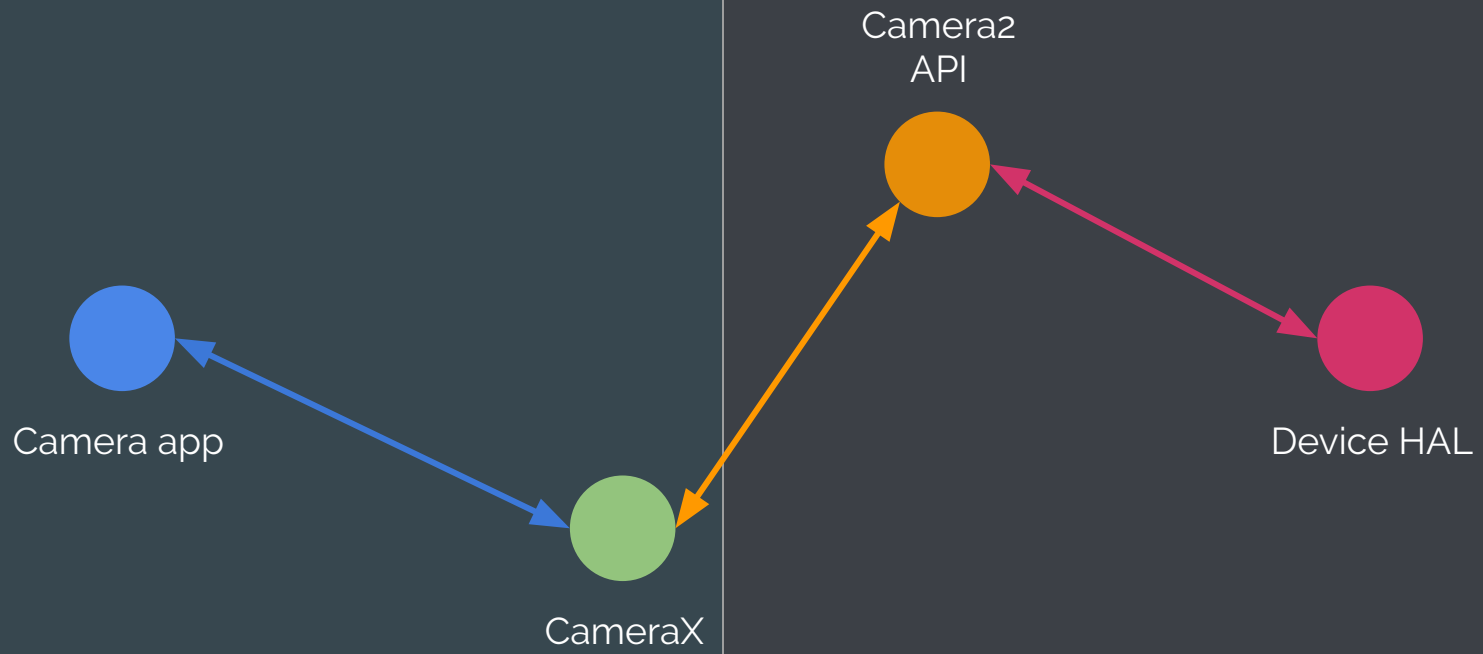
Processo



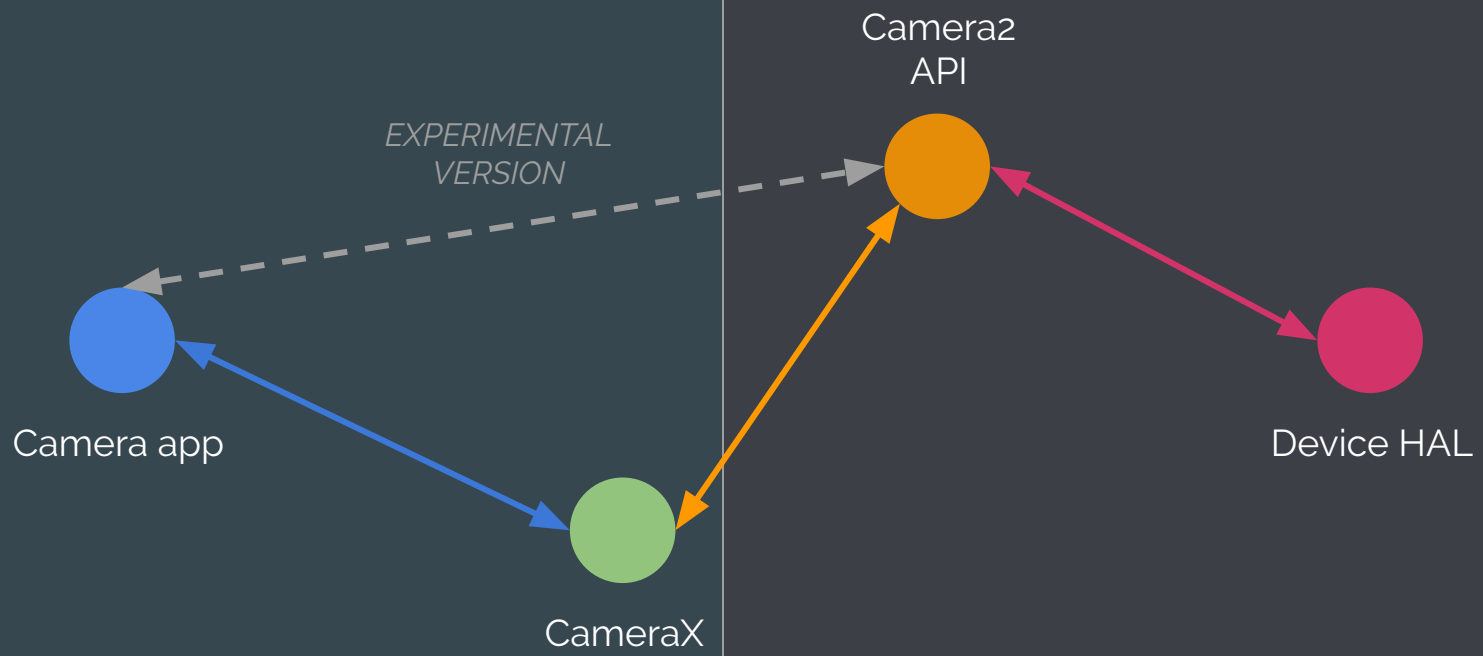
Processo



Processo



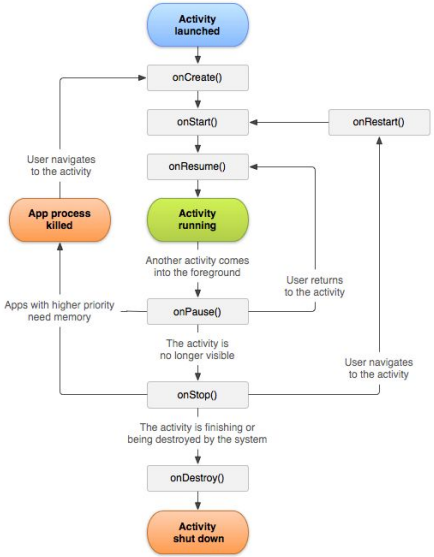
Processo


Camera2
API
EXPERIMENTAL
VERSION
Camera app
CameraX
Device HAL
CameraX API

CameraX é
Lifecycle aware

CameraX é

Lifecycle aware



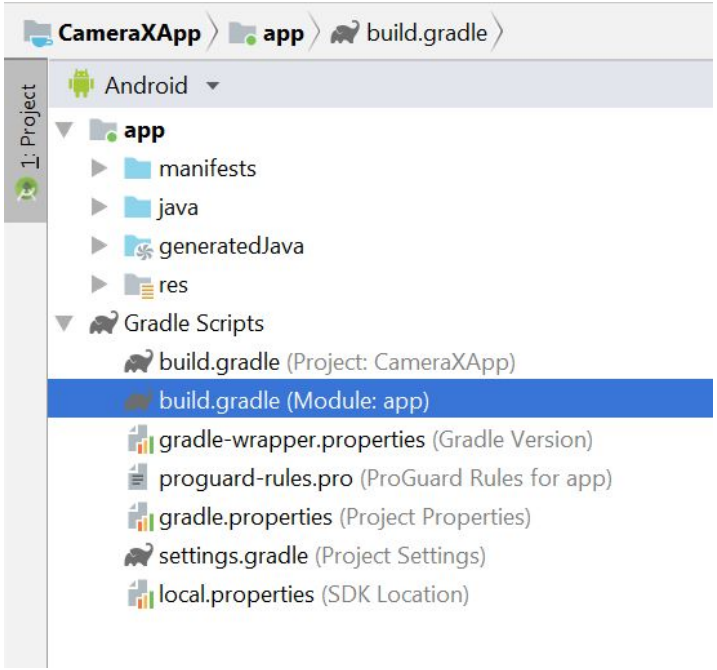

CASOS DE USO


1 Preview
CASOS DE USO


1 Preview
2 Image analysis
CASOS DE USO


1 Preview
2 Image analysis
3 Capture
CASOS DE USO

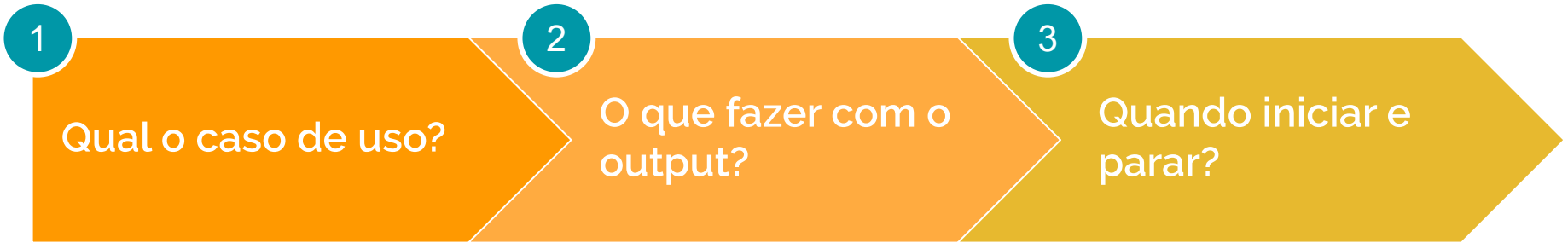
Gradle Dependencies



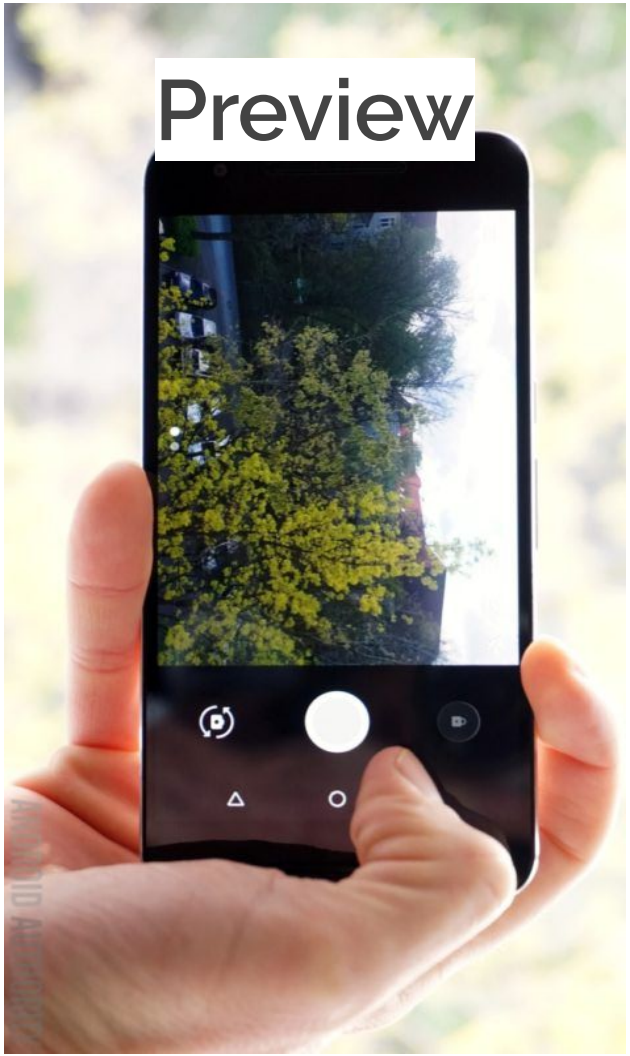

```
def version = "1.0.0-alpha04"
implementation "androidx.camera:camera-core:${version}"
implementation "androidx.camera:camera-camera2:${version}"
```

Use Cases

Como implementar em 3 passos?



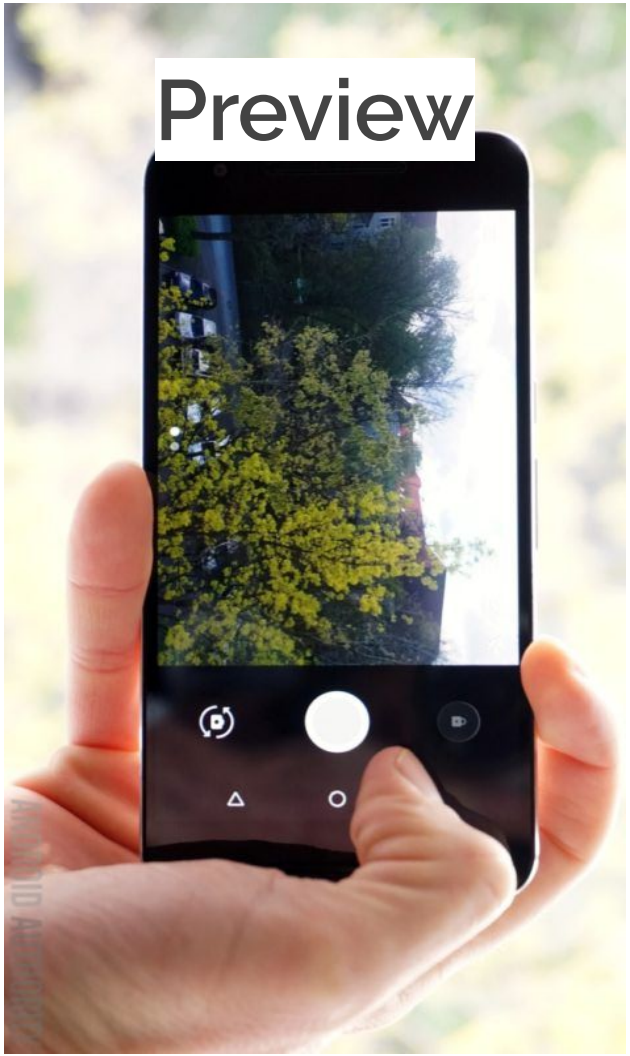

1 Preview
CASOS DE USO



// configure preview

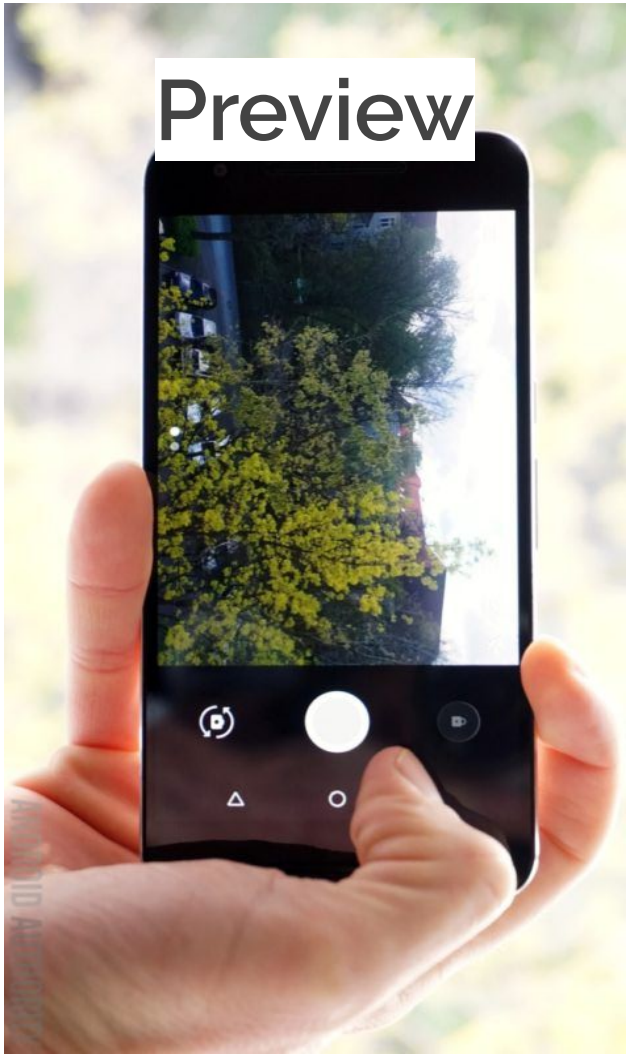
```
val previewConfig = PreviewConfig.Builder().build()
```

1




```
// create preview

val previewConfig = PreviewConfig.Builder().build()

val preview = Preview(previewConfig)
```

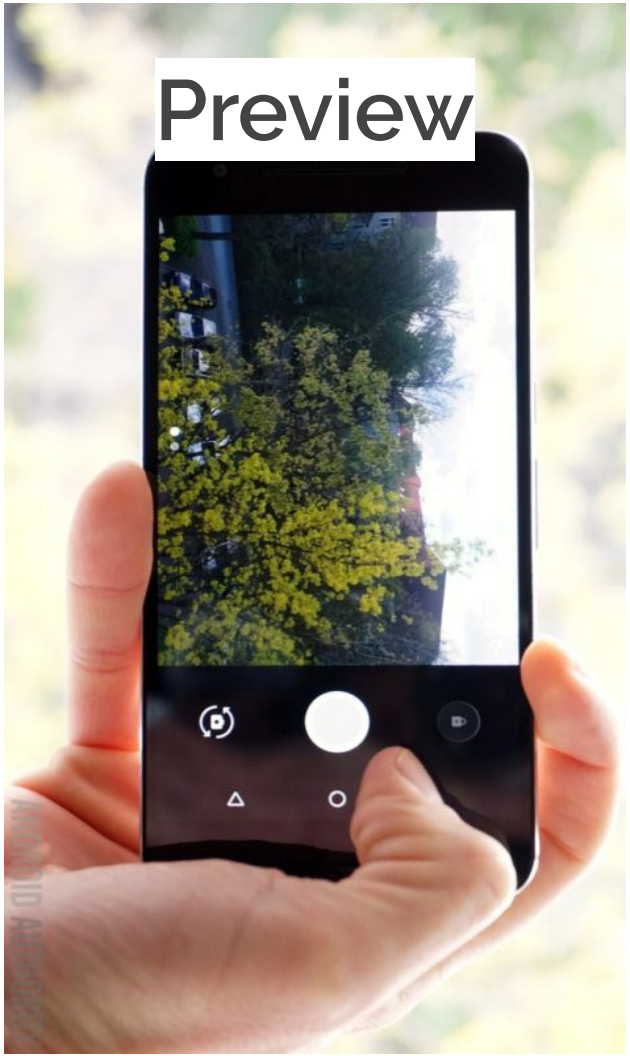
1




```
// attach output to view

val previewConfig = PreviewConfig.Builder().build()

val preview = Preview(previewConfig)

// Every time the viewfinder(TextureView) is updated, recompute layout
preview.setOnPreviewOutputUpdateListener {
    previewOutput: Preview.PreviewOutput? ->

  // To update the SurfaceTexture, we have to remove it and re-add it
  val parent = viewFinder.parent as ViewGroup
  parent.removeView(viewFinder)
  parent.addView(viewFinder, 0)

  viewFinder.surfaceTexture = previewOutput.surfaceTexture
}
```

1

2

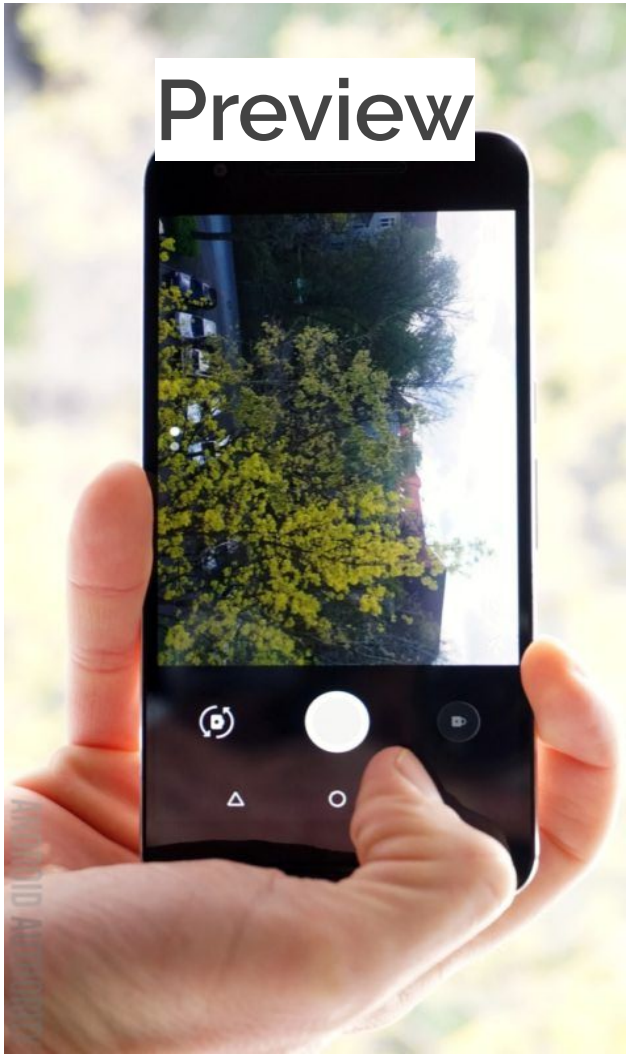



```
// attach preview to lifecycle

val previewConfig = PreviewConfig.Builder().build()

val preview = Preview(previewConfig)

// Every time the viewfinder(TextureView) is updated, recompute layout
preview.setOnPreviewOutputUpdateListener {
    previewOutput: Preview.PreviewOutput? ->

  // To update the SurfaceTexture, we have to remove it and re-add it
  val parent = viewFinder.parent as ViewGroup
  parent.removeView(viewFinder)
  parent.addView(viewFinder, 0)

  viewFinder.surfaceTexture = previewOutput.surfaceTexture
}

// Bind use cases to lifecycle
CameraX.bindToLifecycle(this as LifecycleOwner, preview)
```

1

2

3




```
// display preview on screen

val previewConfig = PreviewConfig.Builder().build()

val preview = Preview(previewConfig)

// Every time the viewfinder(TextureView) is updated, recompute layout
preview.setOnPreviewOutputUpdateListener {
      previewOutput: Preview.PreviewOutput? ->

   // To update the SurfaceTexture, we have to remove it and re-add it
   val parent = viewFinder.parent as ViewGroup
   parent.removeView(viewFinder)
   parent.addView(viewFinder, 0)

   viewFinder.surfaceTexture = previewOutput.surfaceTexture
}

// Bind use cases to lifecycle
CameraX.bindToLifecycle(this as LifecycleOwner, preview)
```


1 Preview
2 Image analysis
CASOS DE USO

// configure image analysis

```
// Create configuration object for the viewfinder use case
val imageAnalysisConfig = ImageAnalysisConfig.Builder()
```

1

// set resolution

```
// Create configuration object for the viewfinder use case
val imageAnalysisConfig = ImageAnalysisConfig.Builder()
        .setTargetResolution(Size(1280, 720))
        .build()
```

1

// create image analysis

```
// Create configuration object for the viewfinder use case
val imageAnalysisConfig = ImageAnalysisConfig.Builder()
        .setTargetResolution(Size(1280, 720))
        .build()

val imageAnalysis = ImageAnalysis(imageAnalysisConfig)
```

1

// attach output

```
// Create configuration object for the viewfinder use case
val imageAnalysisConfig = ImageAnalysisConfig.Builder()
        .setTargetResolution(Size(1280, 720))
        .build()

val imageAnalysis = ImageAnalysis(imageAnalysisConfig)

imageAnalysis.setAnalyzer({image: ImageProxy, rotationDegrees: Int ->
        val cropRect = image.cropRect
        // insert your code here
})
```

1

2

```
// attach image analysis & preview to lifecycle

// Create configuration object for the viewfinder use case
val imageAnalysisConfig = ImageAnalysisConfig.Builder()
        .setTargetResolution(Size(1280, 720))
        .build()

val imageAnalysis = ImageAnalysis(imageAnalysisConfig)

imageAnalysis.setAnalyzer({image: ImageProxy, rotationDegrees: Int ->
      val cropRect = image.cropRect
      // insert your code here
})

CameraX.bindToLifecycle(this as LifecycleOwner, imageAnalysis,
        preview)
```

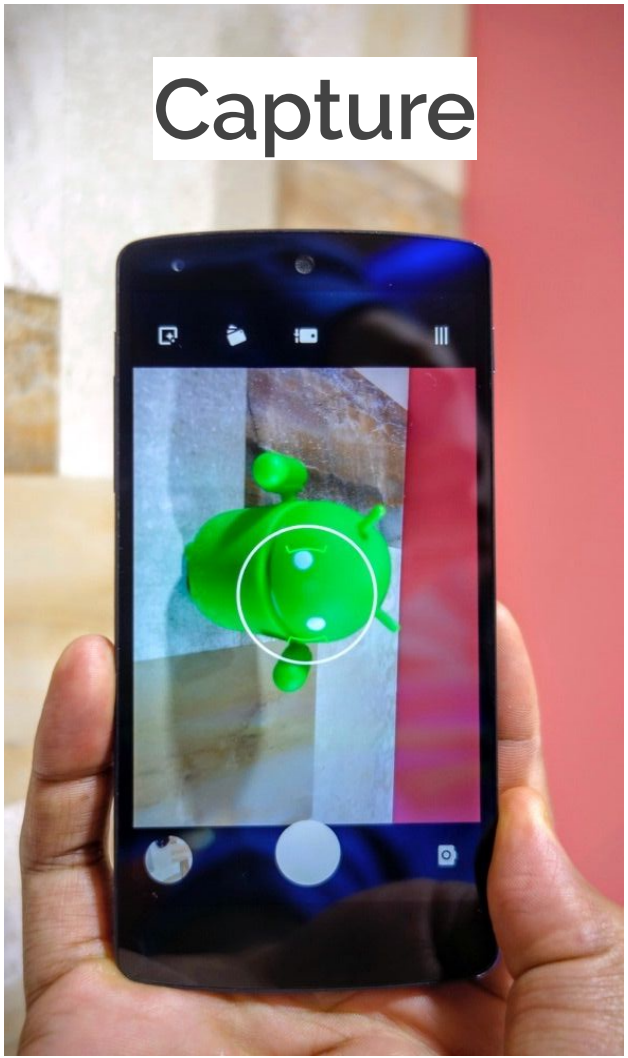
1

2

3

```
// full setup to process images

// Create configuration object for the viewfinder use case
val imageAnalysisConfig = ImageAnalysisConfig.Builder()
        .setTargetResolution(Size(1280, 720))
        .build()

val imageAnalysis = ImageAnalysis(imageAnalysisConfig)

imageAnalysis.setAnalyzer({image: ImageProxy, rotationDegrees: Int ->
        val cropRect = image.cropRect
        // insert your code here
})

CameraX.bindToLifecycle(this as LifecycleOwner, imageAnalysis,
        preview)
```

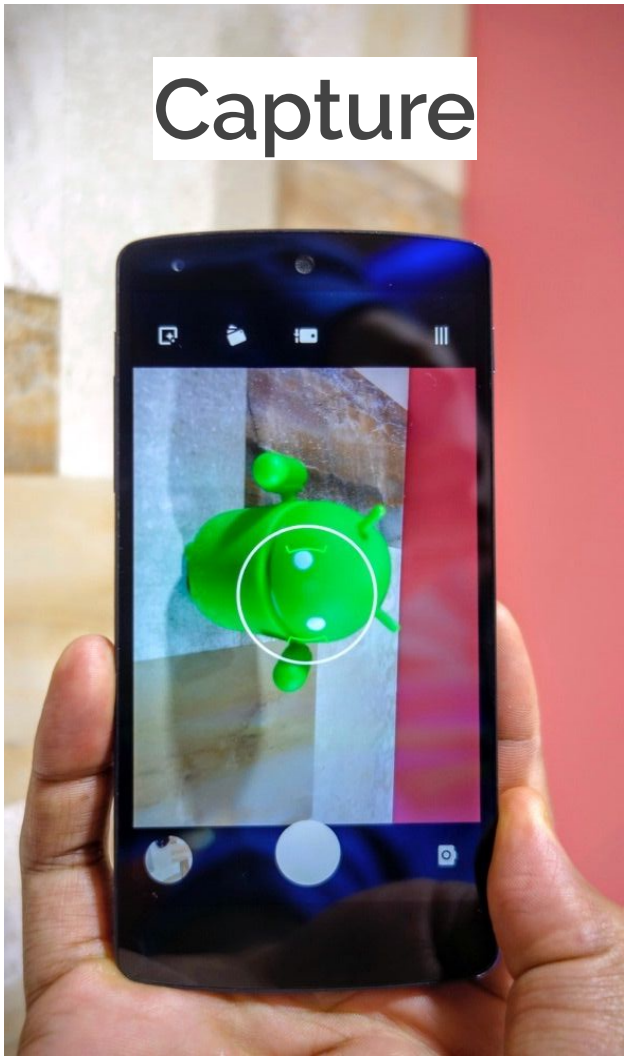

1 Preview
2 Image analysis
3 Capture
CASOS DE USO



// manage rotation

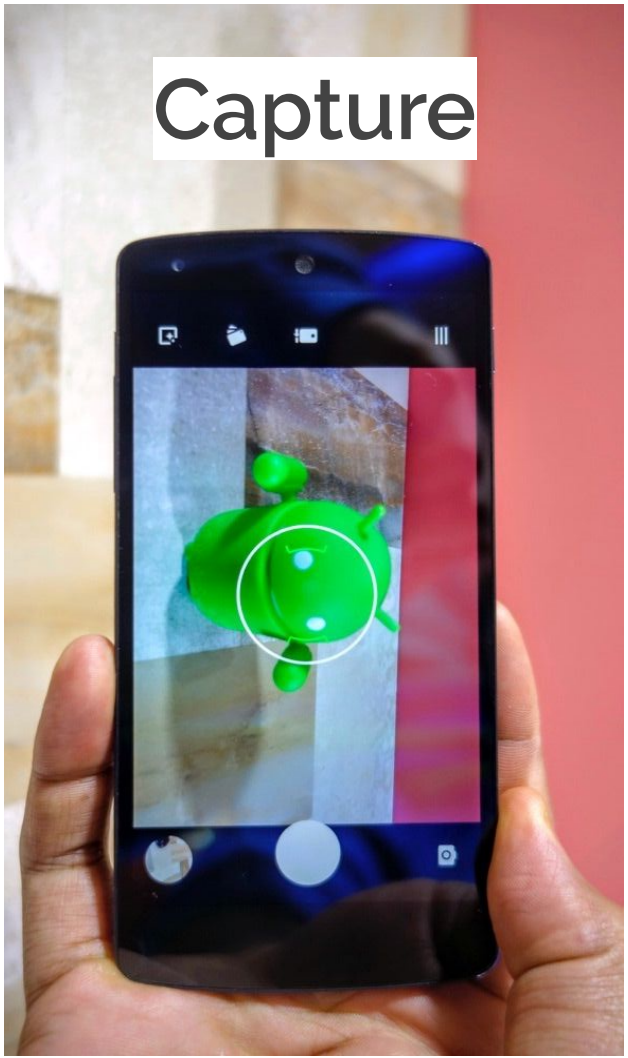
```
// Create configuration object for the viewfinder use case
val imageCaptureConfig = ImageCaptureConfig.Builder()
        .setTargetRotation(windowManager.defaultDisplay.rotation)
        .build()
```

1




```
// create image capture

// Create configuration object for the viewfinder use case
val imageCaptureConfig = ImageCaptureConfig.Builder()
        .setTargetRotation(windowManager.defaultDisplay.rotation)
        .build()

val imageCapture = ImageCapture(imageCaptureConfig)
```

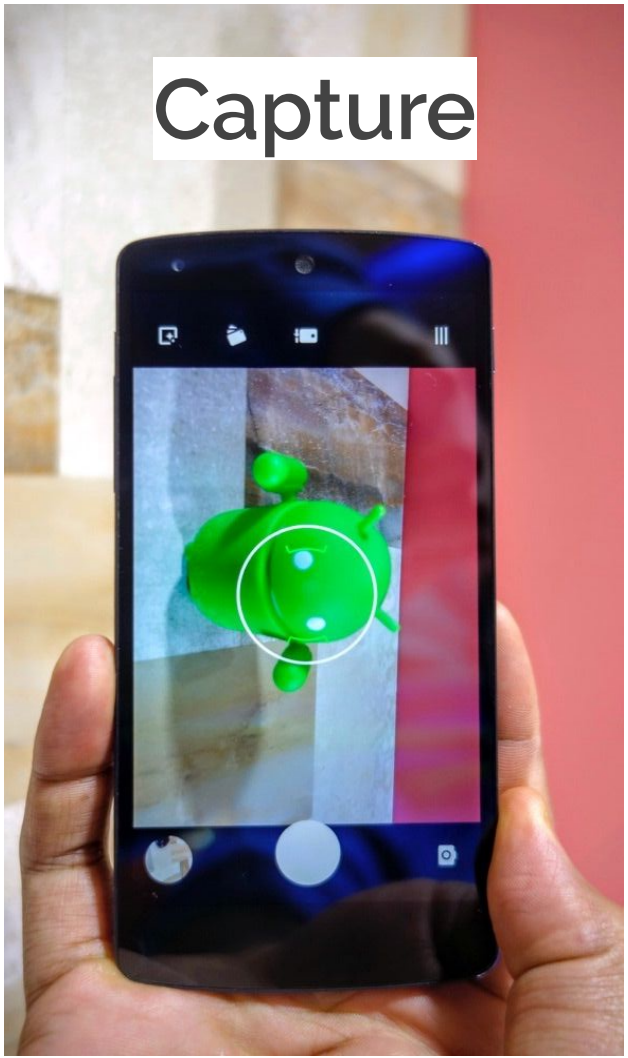
1




```
// bind all use cases

// Create configuration object for the viewfinder use case
val imageCaptureConfig = ImageCaptureConfig.Builder()
        .setTargetRotation(windowManager.defaultDisplay.rotation)
        .build()

val imageCapture = ImageCapture(imageCaptureConfig)

CameraX.bindToLifecycle(this as LifecycleOwner,
        imageCapture, imageAnalysis, preview)
```

1

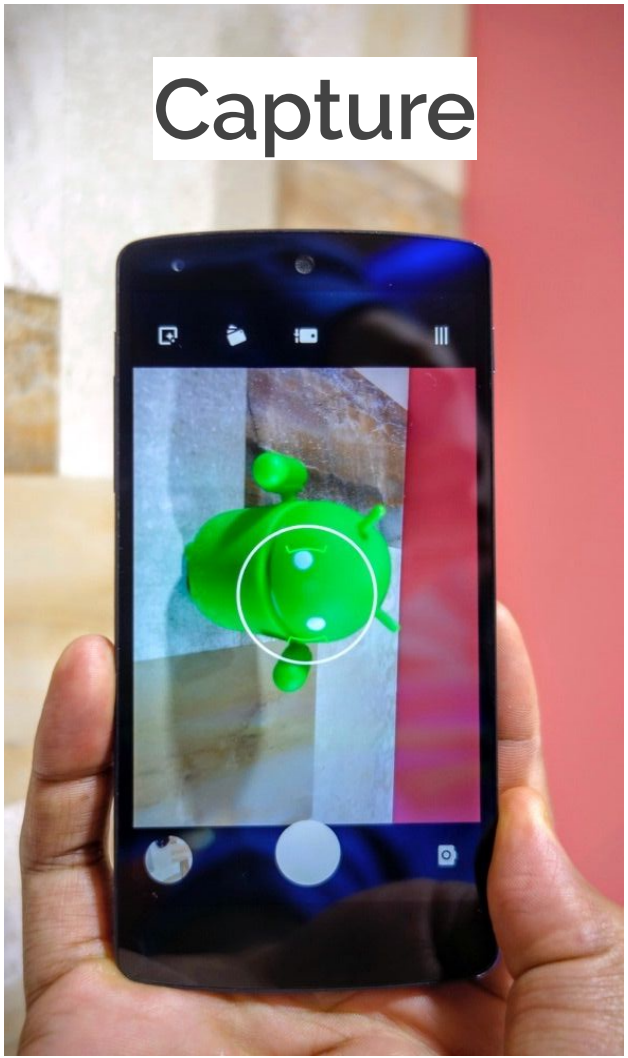
3



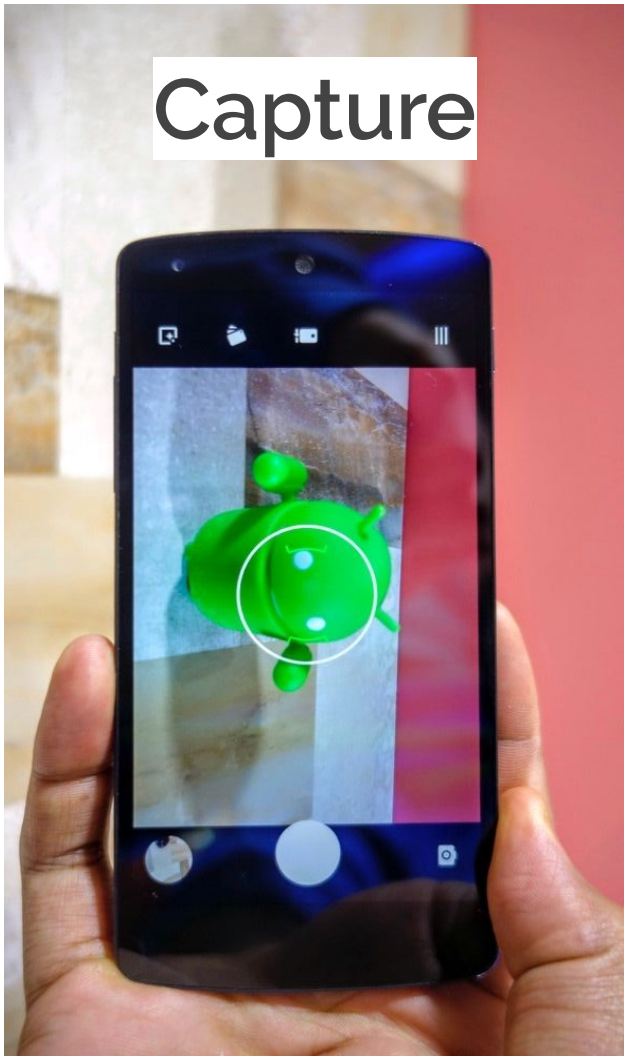

```
// on user action

fun onClick() {

}
```

2

Capture



```
// on user action save a picture

fun onClick() {
        val file = File(..)
        imageCapture.takePicture(



        )

}
```



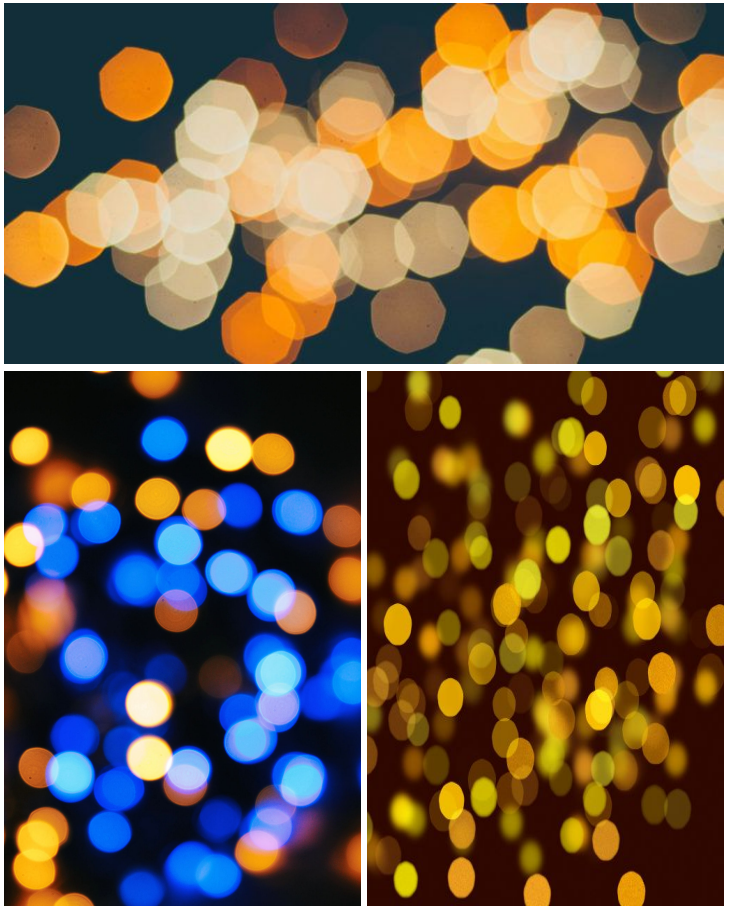
Capture

```
// save a picture on user action

fun onClick() {
        val file = File(..)
        imageCapture.takePicture(file,
                object : ImageCapture.OnImageSavedListener {

                        override fun onError(
                                error: ImageCapture.UseCaseError,
                                message: String,
                                exc: Throwable?) {
                            // insert your code here
                        }
                        override fun onImageSaved(file: File) {
                            // insert your code here
                        }

                })
}
```


EXTENSIONS
Efeitos do fornecedor específicos do dispositivo
HUAWEI
Portrait
SAMSUNG
Galaxy
Night
HUAWEI
SAMSUNG
Galaxy
HDR
SAMSUNG
Galaxy
Beauty

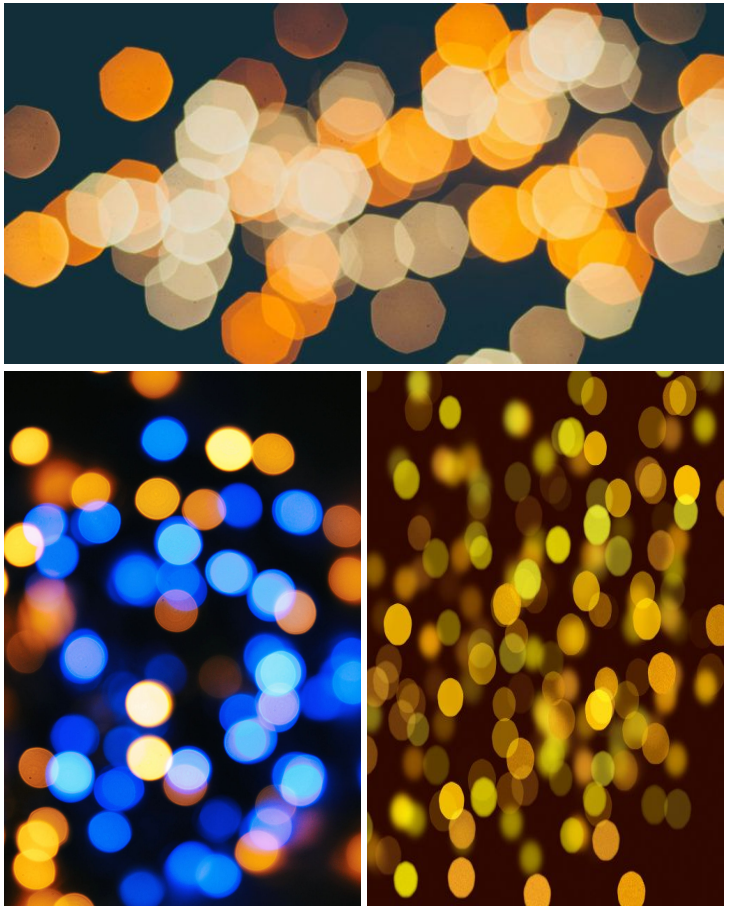
Bokeh Extension

```
val builder = ImageCaptureConfig.Builder()

val config = builder.build()
val useCase = ImageCapture(config)
CameraX.bindToLifecycle(this as LifecycleOwner, useCase)
```



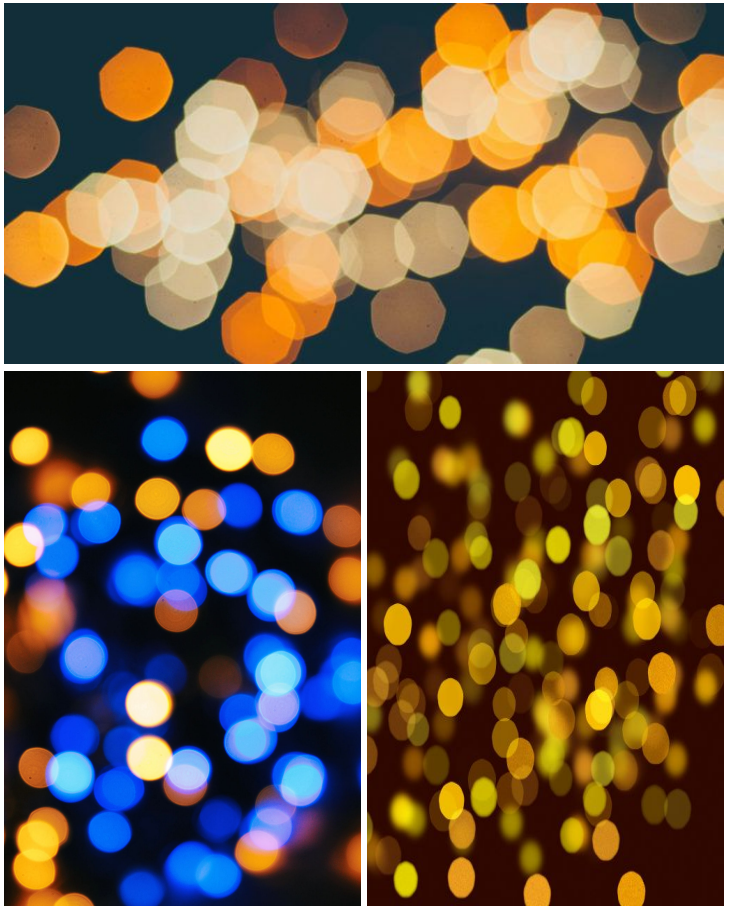
Bokeh Extension

```
import androidx.camera.extensions.BokehExtender

val builder = ImageCaptureConfig.Builder()

val config = builder.build()
val useCase = ImageCapture(config)
CameraX.bindToLifecycle(this as LifecycleOwner, useCase)
```



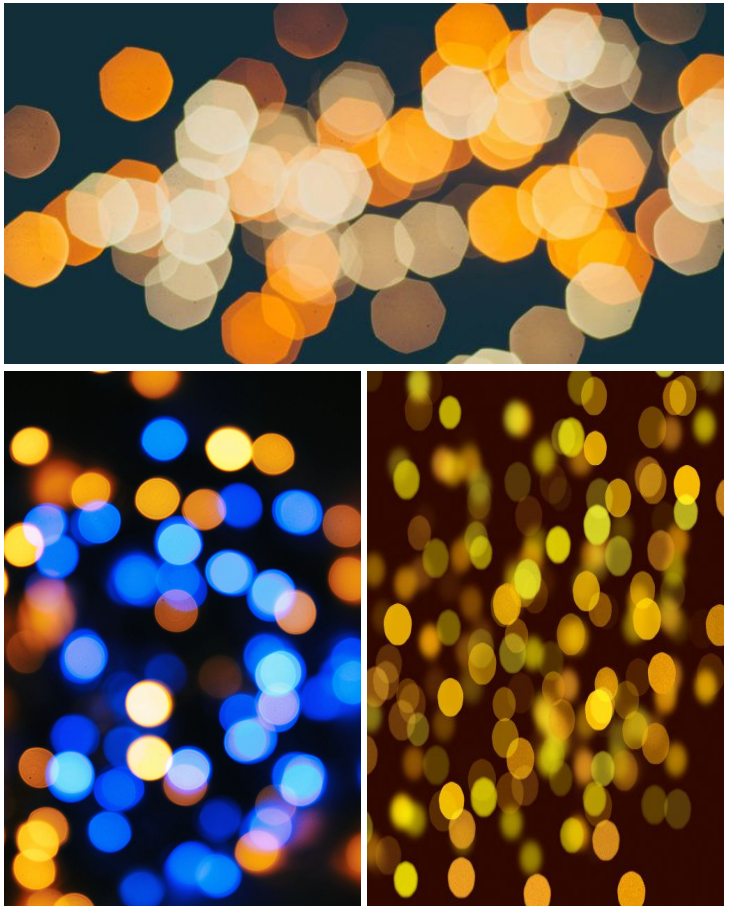
Bokeh Extension

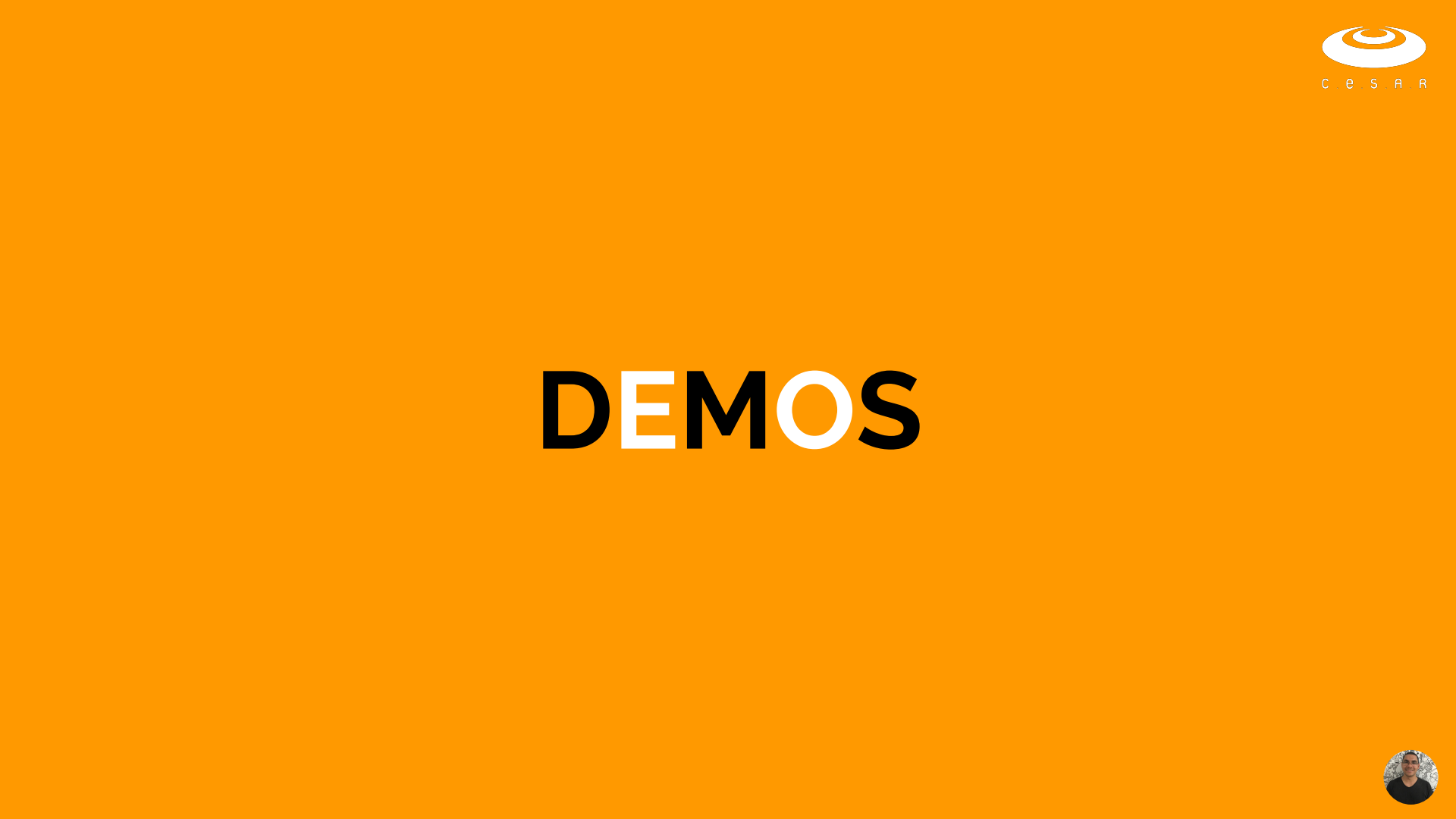
```
import androidx.camera.extensions.BokehExtender

val builder = ImageCaptureConfig.Builder()

val bokehImageCapture = BokehImageCaptureExtender.create(builder)

// Query if extension is available (optional).
if (bokehImageCapture.isExtensionAvailable()) {
   // Enable the extension if available.
   bokehImageCapture.enableExtension()
}

val config = builder.build()
val useCase = ImageCapture(config)
CameraX.bindToLifecycle(this as LifecycleOwner, useCase)
```



Bokeh Extension

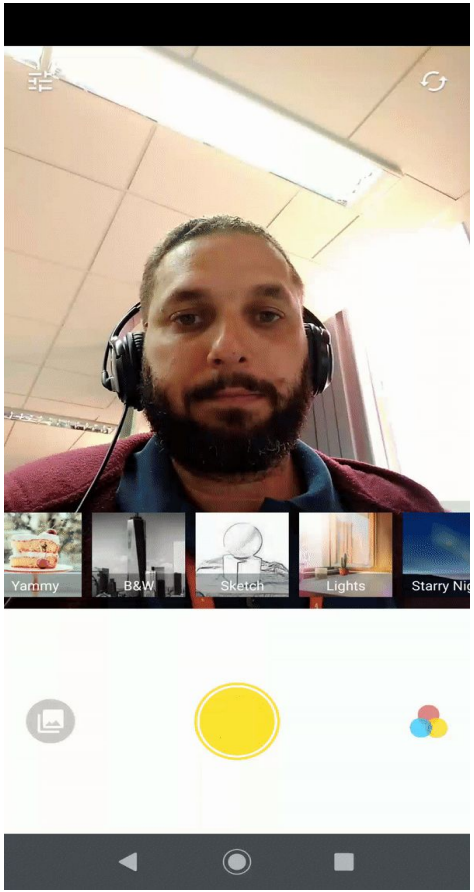
```
import androidx.camera.extensions.BokehExtender

val builder = ImageCaptureConfig.Builder()

val bokehImageCapture = BokehImageCaptureExtender.create(builder)

// Query if extension is available (optional).
if (bokehImageCapture.isExtensionAvailable()) {
   // Enable the extension if available.
   bokehImageCapture.enableExtension()
}

val config = builder.build()
val useCase = ImageCapture(config)
CameraX.bindToLifecycle(this as LifecycleOwner, useCase)
```




DEMOS

Camera 360

- Aplica vários efeitos usando CameraX
- Usa os 3 use cases (Preview, image analysis e captura de imagem)
- Por “debaixo dos panos”, CameraX está conversando com a camada da Camera2



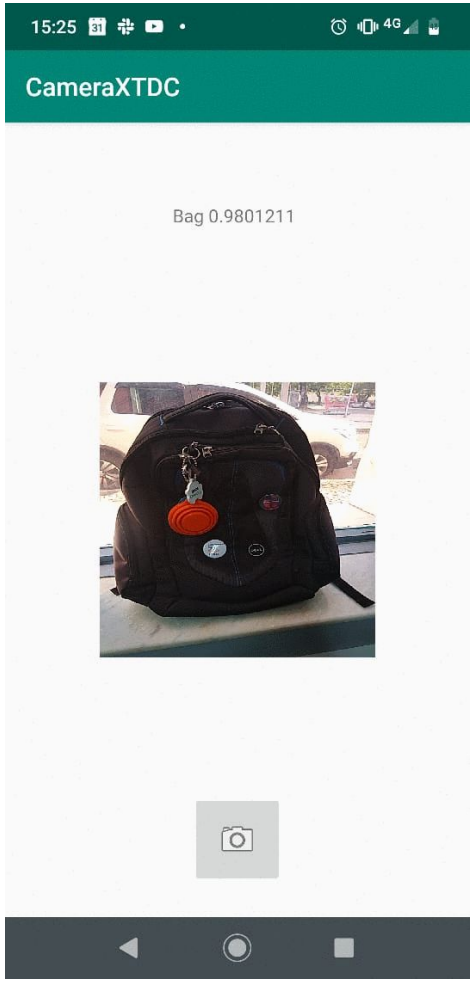
Benefícios da CameraX

- Redução dos testes em dispositivos específicos
- 75% de redução de linhas de código comparado ao Camera2
- Fácil leitura do código
- Diminuiu o tamanho do apk

Camera 360

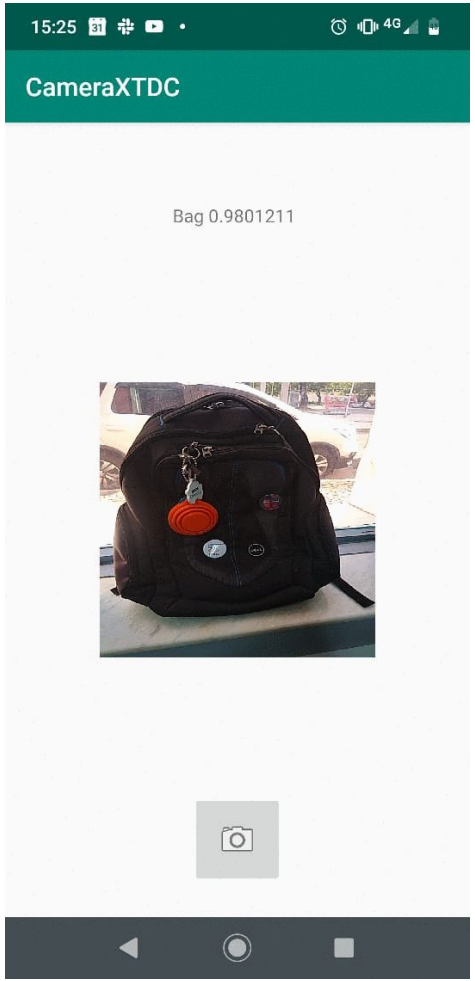
Custom Analysis

```
// Setup image analysis pipeline
val analyzerConfig = ImageAnalysisConfig.Builder().apply {

   // Use a worker thread for image analysis to prevent glitches
   val analyzerThread = HandlerThread("LabelAnalysis").apply { start() }
   setCallbackHandler(Handler(analyzerThread.looper))

   setImageReaderMode(ImageAnalysis.ImageReaderMode.ACQUIRE_LATEST_IMAGE)
}.build()

// Build the image analysis use case and instantiate our analyzer
val analyzerUseCase = ImageAnalysis(analyzerConfig).apply {
   analyzer = LabelAnalyzer(label)
}
```



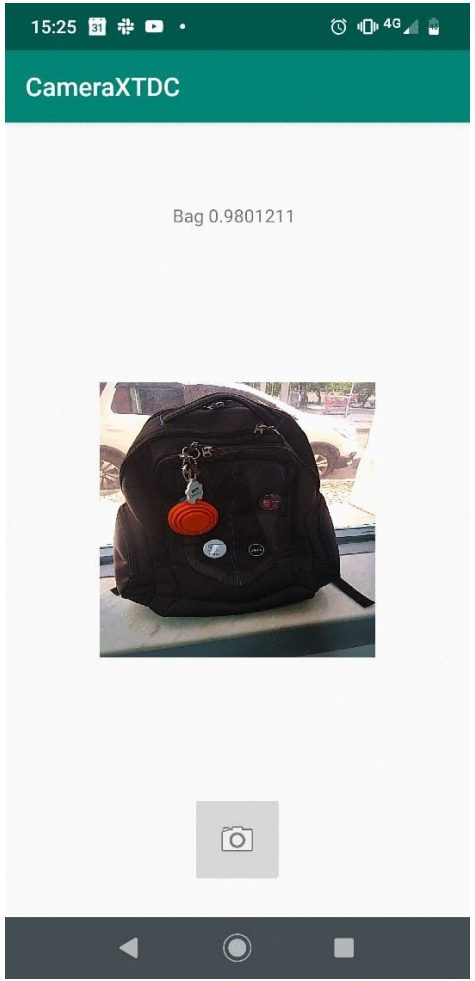
Custom Analysis

```
// Setup image analysis pipeline
val analyzerConfig = ImageAnalysisConfig.Builder().apply {

    // Use a worker thread for image analysis to prevent glitches
    val analyzerThread = HandlerThread("LabelAnalysis").apply { start() }
    setCallbackHandler(Handler(analyzerThread.looper))

    setImageReaderMode(ImageAnalysis.ImageReaderMode.ACQUIRE_LATEST_IMAGE)
}.build()

// Build the image analysis use case and instantiate our analyzer
val analyzerUseCase = ImageAnalysis(analyzerConfig).apply {
    analyzer = LabelAnalyzer(label)
}
```



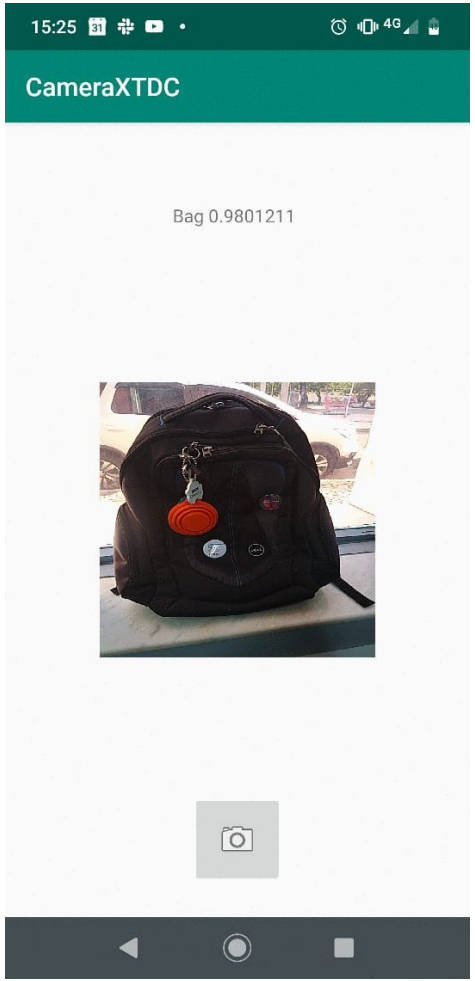
Custom Analysis

```
// Setup image analysis pipeline
val analyzerConfig = ImageAnalysisConfig.Builder().apply {

   // Use a worker thread for image analysis to prevent glitches
   val analyzerThread = HandlerThread("LabelAnalysis").apply { start() }
   setCallbackHandler(Handler(analyzerThread.looper))

   setImageReaderMode(ImageAnalysis.ImageReaderMode.ACQUIRE_LATEST_IMAGE)
}.build()

// Build the image analysis use case and instantiate our analyzer
val analyzerUseCase = ImageAnalysis(analyzerConfig).apply {
   analyzer = LabelAnalyzer(label)
}
```



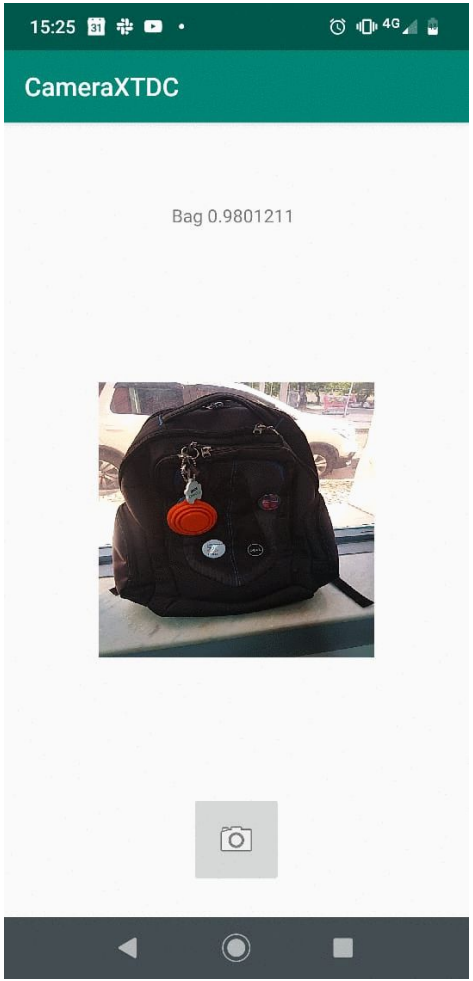
Custom Analysis

```
// Setup image analysis pipeline
val analyzerConfig = ImageAnalysisConfig.Builder().apply {

    // Use a worker thread for image analysis to prevent glitches
    val analyzerThread = HandlerThread("LabelAnalysis").apply { start() }
    setCallbackHandler(Handler(analyzerThread.looper))

    setImageReaderMode(ImageAnalysis.ImageReaderMode.ACQUIRE_LATEST_IMAGE)
}.build()

// Build the image analysis use case and instantiate our analyzer
val analyzerUseCase = ImageAnalysis(analyzerConfig).apply {
    analyzer = LabelAnalyzer(label)
}
```



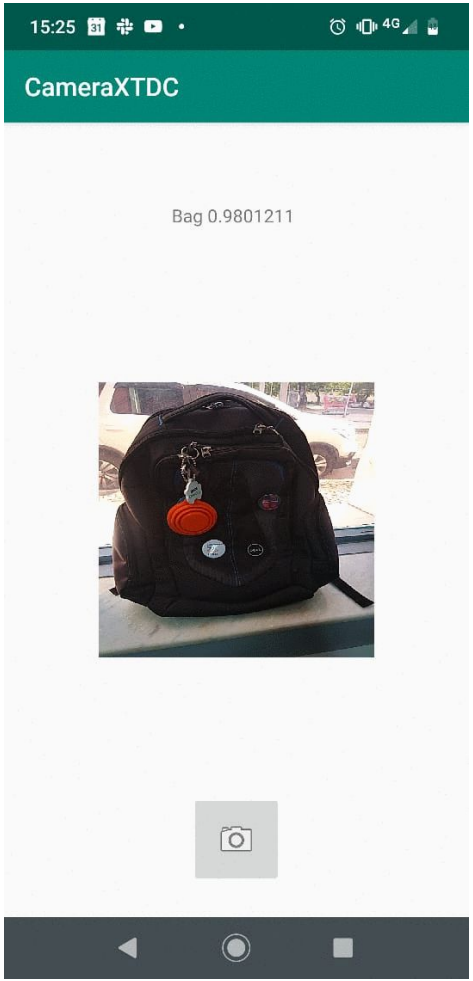
Custom Analysis

```
class LabelAnalyzer(val textView: TextView) : ImageAnalysis.Analyzer {
  override fun analyze(image: ImageProxy, rotationDegrees: Int) {
    val data = imageByteArray(image)

    val metadata = FirebaseVisionImageMetadata.Builder()
        .setFormat(FirebaseVisionImageMetadata.IMAGE_FORMAT_YV12)
        .setHeight(image.height)
        .setWidth(image.width)
        .setRotation(getRotation(rotationDegrees))
        .build()

    val labelImage = FirebaseVisionImage.fromByteArray(data, metadata)

    val labeler = FirebaseVision.getInstance().getOnDeviceImageLabeler()
    labeler.processImage(labelImage).addOnSuccessListener { labels ->
          textView.run {
            if (labels.size >= 1) {
              text = labels[0].text + " " + labels[0].confidence
            }
          }
      }
  }
```



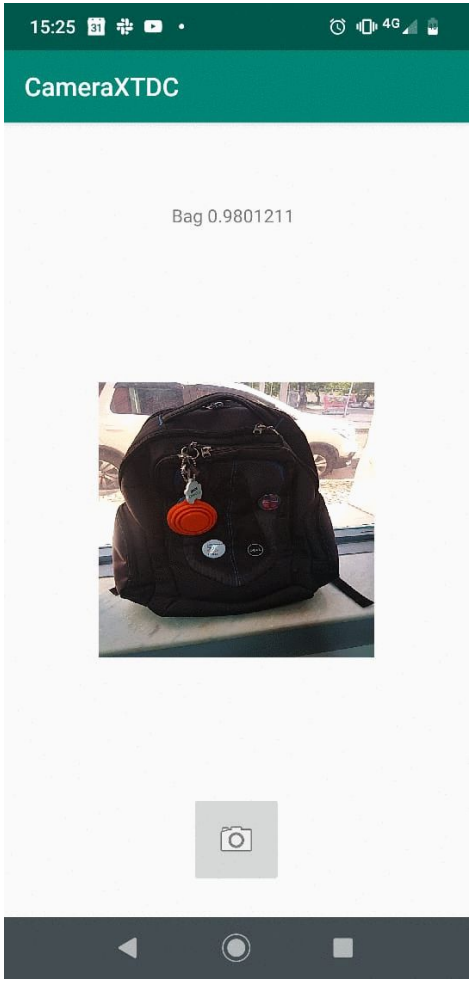
Custom Analysis

```
class LabelAnalyzer(val textView: TextView) : ImageAnalysis.Analyzer {
  override fun analyze(image: ImageProxy, rotationDegrees: Int) {
    val data = imageByteArray(image)

    val metadata = FirebaseVisionImageMetadata.Builder()
        .setFormat(FirebaseVisionImageMetadata.IMAGE_FORMAT_YV12)
        .setHeight(image.height)
        .setWidth(image.width)
        .setRotation(getRotation(rotationDegrees))
        .build()

    val labelImage = FirebaseVisionImage.fromByteArray(data, metadata)

    val labeler = FirebaseVision.getInstance().getOnDeviceImageLabeler()
    labeler.processImage(labelImage).addOnSuccessListener { labels ->
          textView.run {
              if (labels.size >= 1) {
                  text = labels[0].text + " " + labels[0].confidence
              }
          }
      }
}
```



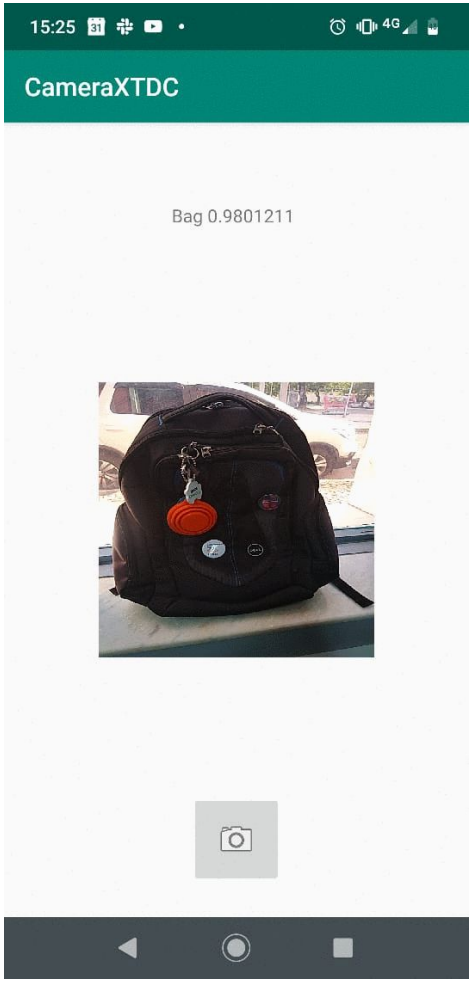
Custom Analysis

```
class LabelAnalyzer(val textView: TextView) : ImageAnalysis.Analyzer {
  override fun analyze(image: ImageProxy, rotationDegrees: Int) {
    val data = imageByteArray(image)

    val metadata = FirebaseVisionImageMetadata.Builder()
        .setFormat(FirebaseVisionImageMetadata.IMAGE_FORMAT_YV12)
        .setHeight(image.height)
        .setWidth(image.width)
        .setRotation(getRotation(rotationDegrees))
        .build()

    val labelImage = FirebaseVisionImage.fromByteArray(data, metadata)

    val labeler = FirebaseVision.getInstance().getOnDeviceImageLabeler()
    labeler.processImage(labelImage).addOnSuccessListener { labels ->
          textView.run {
              if (labels.size >= 1) {
                  text = labels[0].text + " " + labels[0].confidence
              }
          }
      }
}
```



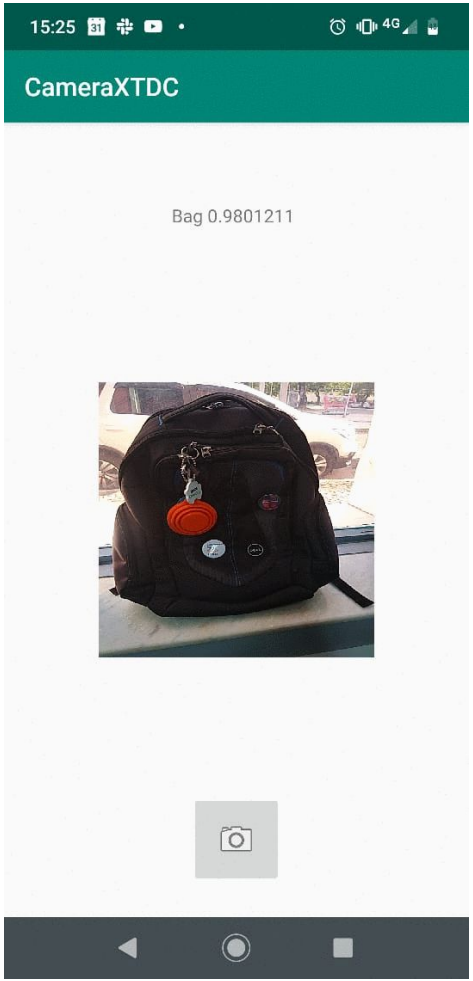
Custom Analysis

```
class LabelAnalyzer(val textView: TextView) : ImageAnalysis.Analyzer {
  override fun analyze(image: ImageProxy, rotationDegrees: Int) {
    val data = imageByteArray(image)

    val metadata = FirebaseVisionImageMetadata.Builder()
        .setFormat(FirebaseVisionImageMetadata.IMAGE_FORMAT_YV12)
        .setHeight(image.height)
        .setWidth(image.width)
        .setRotation(getRotation(rotationDegrees))
        .build()

    val labelImage = FirebaseVisionImage.fromByteArray(data, metadata)

    val labeler = FirebaseVision.getInstance().getOnDeviceImageLabeler()
    labeler.processImage(labelImage).addOnSuccessListener { labels ->
          textView.run {
              if (labels.size >= 1) {
                  text = labels[0].text + " " + labels[0].confidence
              }
          }
      }
}
```



Custom Analysis

```
class LabelAnalyzer(val textView: TextView) : ImageAnalysis.Analyzer {
  override fun analyze(image: ImageProxy, rotationDegrees: Int) {
    val data = imageByteArray(image)

    val metadata = FirebaseVisionImageMetadata.Builder()
        .setFormat(FirebaseVisionImageMetadata.IMAGE_FORMAT_YV12)
        .setHeight(image.height)
        .setWidth(image.width)
        .setRotation(getRotation(rotationDegrees))
        .build()

    val labelImage = FirebaseVisionImage.fromByteArray(data, metadata)

    val labeler = FirebaseVision.getInstance().getOnDeviceImageLabeler()
    labeler.processImage(labelImage).addOnSuccessListener { labels ->
          textView.run {
            if (labels.size >= 1) {
              text = labels[0].text + " " + labels[0].confidence
            }
          }
      }
}
```



Obrigado!

wellingtoncab@gmail.com

josemourajr@gmail.com