Arquitetura de Aplicações para Android

Thiago O. Maciel

2018

Arquitetura de Aplicações para Android

Thiago O. Maciel

Sumário

Capítulo 1. Introdução ao Android

Neste capítulo faremos uma introdução geral ao sistema operacional Android, sua evolução com o tempo e alguns detalhes do novo paradigma de ter um sistema operacional em tudo.

Sistema operacional Android

O Android é um SO baseado em Linux e, atualmente, desenvolvido pela Google. Ele está presente em nossas vidas há quase dez anos. Durante esse tempo, vimos uma evolução extraordinária no ciclo de desenvolvimento de sistemas operacionais (SOs). Quando a Google entrou na briga pelos *smartphones*, decidiu trazer o ciclo de atualizações Web para um SO. O resultado foi o melhor possível e, cada vez mais, tem constantes melhorias.

Depois de um período de maturação – do lançamento do *Ice Cream Sandwich* até o *KitKat* – essas atualizações do sistema operacional já não serão tão frequentes (comparadas ao início, quando tínhamos uma atualização a cada dois meses e meio). Com algumas estatísticas feitas nos últimos anos, a Google percebeu que os fabricantes não atualizam seus dispositivos numa taxa adequada e, com isso, a empresa decidiu tentar dividir o sistema operacional em uma maior quantidade de partes atualizáveis via Google Play.

Hoje, temos o sistema operacional mais utilizado no mundo [1-3], com quase dois bilhões de dispositivos vendidos e quase 1,5 milhões de dispositivos ativados por dia!

Evolução do sistema operacional

A lista abaixo mostra de forma simples a evolução do sistema operacional:

- *Android 0.5 Milestone 3*: a primeira *build* pública;
- *Android 0.5 Milestone 5*: a adição de *scrapped interfaces;*
- *Android 0.9 Beta:* já começa a ter “cara de Android”;
- *Android 1.0:* introduzindo Google Apps
- *Android 1.1:* a primeira atualização de fato;
- Android 1.5 Cupcake: teclado virtual. Google Maps é o primeiro aplicativo embutido no mercado Android;
- Android 1.6 suporte Donut: suporte a CDMA traz Android a qualquer dispositivo;
- Android 2.0 Éclair: aquecendo a indústria de GPS;
- O Nexus One: o primeiro *Google Phone;*
- Android 2.1: descoberta (e abuso) de animações;
    - Android 2.1 atualização 1: o início de uma guerra sem fim;
- Android 2.2 Froyo: rápido e *Flash-ier*;
    - Ações à voz: um supercomputador no seu bolso;
- Android 2.3 Gingerbread: a primeira grande reformulação da interface com o usuário;
- Android 3.0 Honeycomb: tablets e a renascença do design;
    - Google Music Beta: de armazenamento na nuvem ao invés de um armazenamento de conteúdo;
- Android 4.0 Ice Cream Sandwich: a era moderna;
    - Google Play e do retorno de vendas de dispositivos direcionados ao consumidor;
- Android 4.1 Jelly Bean: Google Agora olha para o futuro;
    - Google Play Services: fragmentação e fazendo a atualização do SO (quase) obsoleta;
- Android 4.2 Jelly Bean: novos dispositivos Nexus, nova interface dos tablets;
    - Atualizações *out-of-cycle:* em que precisa de um novo sistema operacional?
- Android 4.3 Jelly Bean: iniciando o suporte aos wearables;

- Android 4.4 KitKat: mais polido; menor uso de memória;
- Android 5.0 Lollipop: Material Design, ART, Documentos;
- Android 5.1 M: Voz em HD, proteção do dispositivo;
- Android 6.0 Marshmallow: nova arquitetura de permissões, novo gerenciamento de energia;
- Android 7.0 Nougat: dividir telas, open-JDK;
- *Android 8.0 Oreo: modo Picture-in-Picture, novas notificações, framework autofill, WebView APIs, suporte multi-display.*
- *Android P: Wi-Fi RTT, suporte multi-câmera, Neural Networks API 1.1.*
- Hoje - Android em todos os lugares!

Você encontra uma discussão mais detalhada dessas versões em [4].

Dispositivos Android

Como dissemos na sessão anterior, nosso paradigma atual é "*Android em todos os lugares!",* e o Android está e estará cada vez mais presente em tudo que possa ter um SO (e até onde nem pensávamos que teríamos um SO), seguindo a internet das coisas (do inglês, *internet of things*).

O sistema operacional, que começou como um pequeno projeto paralelo dentro do Google, está agora ao longo da empresa toda. Quase tudo que o Google vende será compatível com Android.

Ainda não sabemos onde essa aderência a SOs móveis irá chegar, mas temos certeza que já existem implicações sérias em nosso cotidiano: principalmente no que toca à privacidade, como o último escândalo com o Facebook (cf. [9]).

O caminho até a versão atual

Muitas pessoas já pensavam que as atualizações do Android não importavam mais. Das atualizações constantes do Jelly Bean até o KitKat parecia apenas polir o que já existia. Isso mudou quando o Google apresentou o “Android Lollipop” no Google I/O 2014, então ficou claro que aquela versão era diferente.

Seguindo a discussão em [4] e a lista da seção 1.2, siga as iterações infinitas do Android 0.5 para Android 4.4. O Android 5.0 Lollipop foi pelo menos a maior atualização desde que o Android 4.0 e é, provavelmente, o maior lançamento Android já feito até a versão 8.0. A atualização trouxe uma reformulação visual completa de cada aplicativo, com uma nova e bela linguagem de design chamada Material Design. As animações começam a aparecer em toda parte!

A versão 5.0 também trouxe uma tonelada de novos recursos: as notificações estão, finalmente, na tela quando bloqueada; a funcionalidade dos aplicativos recentes foi renovada para tornar multitarefa algo que era muito mais fácil; o reconhecimento de voz funciona em todos os lugares, mesmo quando a tela está desligada e muito mais.

As reformas sob o capô são muito extensas, incluindo uma API de câmera completamente nova, com suporte para imagens RAW, um foco de todo o sistema sobre a vida da bateria e um novo *runtime*: o *ART*, que substitui a velha máquina virtual Dalvik.

Da versão 5.0 para a 5.1 houve muitas correções de erros: o que é normal logo após o lançamento de uma versão maior, principalmente um erro grave de memory leak, constatado em dezembro de 2014 e que, finalmente, foi corrigido.

Além disso, a performance de dispositivos criptografados foi corrigida e melhorada. A funcionalidade de proteger seu dispositivo em caso de roubo foi adicionada: uma

vez que a tela de bloqueio foi definida, a proteção do dispositivo seria acionada e persistiria durante uma limpeza do dispositivo. Se você limpasse o telefone da maneira que um proprietário faria normalmente – desbloqueando o telefone e escolhendo “reset” das configurações – nada aconteceria. No entanto, se você limpar o telefone usando as ferramentas do desenvolvedor, o dispositivo exigirá que você “verifique uma Conta do Google sincronizada anteriormente” durante a próxima configuração.

A ideia era que o usuário conheceria as credenciais do Google no dispositivo, mas um ladrão não o faria e ficaria preso na tela de configuração. Na prática, isso desencadeou um jogo de gato e rato de pessoas encontrando vulnerabilidades que contornam a proteção do dispositivo, e o Google, recebendo notícias sobre o bug, os corrigia.

Uma ressalva para a nova funcionalidade de conversar em alta qualidade. Isso já está presente nas redes europeias da T-Mobile. No Brasil, as operadoras não têm previsão para implementar isso em suas redes.

A outra grande história é que, enquanto a Google está construindo um império de *smartphones* em todo o mundo, o Android não está apenas em *smartphones* e tablets, o Android 5.0 foi lançada no Android Wear, Android TV e Android Auto. O sistema operacional se tornou o sistema operacional do Google de fato, servindo como base para o Google Glass e o Chromecast. Até mesmo o Chrome OS suporta aplicativos Android.

Também em março de 2015, o Google lançou o Android Auto, uma nova interface inspirada no Android para sistemas de infoentretenimento veicular. O Android Auto foi a resposta do Google ao CarPlay da Apple e funcionou da mesma maneira. Não era um sistema operacional completo – é uma interface que roda em seu telefone e usa a tela integrada do carro como um monitor externo. Executar o Android Auto significa ter um carro compatível, instalar o aplicativo Android Auto em seu telefone (Android 5.0 e superior) e conectar o telefone ao carro com um cabo USB.

O Android Auto não viu muito em termos de atualizações após o lançamento inicial, mas já viu uma tonelada de suporte aos fabricantes de automóveis. Em 2017, houve mais de 100 modelos de veículos compatíveis (cf. [8]).

Em outubro de 2015, o Google trouxe o Android 6.0 Marshmallow para o mundo. Para o lançamento do sistema operacional, o Google encomendou dois novos dispositivos Nexus: o Huawei Nexus 6P e o LG Nexus 5X. Em vez de apenas aumentar a velocidade normal, os novos telefones também incluíam uma peça-chave de hardware: um leitor de impressões digitais para a nova API de impressão digital do Marshmallow. Ele também estava organizando um novo recurso de busca chamado “Google Now on Tap”, permissões de aplicativos controladas pelo usuário, um novo sistema de backup de dados e muitos outros refinamentos.

Em vez de tornar isso um recurso exclusivo, o Google criou uma nova “API do Assistente” no Android. O usuário poderia escolher um “aplicativo de assistência” que receberia informações suficientes ao pressionar o botão de início. O aplicativo assistente receberia todo o texto carregado atualmente pelo aplicativo, não apenas o que estava na tela, junto com todas as imagens e os metadados especiais que o desenvolvedor queria incluir. Essa API funcionava com o Google Now on Tap e também permitia que terceiros criassem rivais do Now on Tap, se quisessem.

O Google empolgou com o Now on Tap durante a apresentação inicial do Marshmallow, mas, na prática, o recurso não foi muito útil. A Pesquisa do Google vale a pena porque você está fazendo uma pergunta exata – você digita o que quiser e vasculha toda a Internet procurando a resposta ou a página da web.

O Now on Tap tornou as coisas infinitamente mais difíceis, porque nem sabia que pergunta você estava fazendo. Você abria o Now on Tap com uma intenção muito específica, mas enviava ao Google uma consulta muito específica de “tudo na tela”. O Google teve que adivinhar qual era a sua consulta e depois tentou fornecer resultados de pesquisa ou ações úteis com base nisso.

Com o lançamento dos Google Pixels em 2016, a empresa aparentemente admitiu a derrota. Ele renomeou o Now on Tap para “Pesquisa da tela” e o rebaixou em favor do Google Assistente. O Assistente – novo sistema de comando de voz do Google – assumiu o gesto do botão home do On Tap e o relacionou a um segundo gesto depois que o sistema de voz foi ativado. O Google também parece ter aprendido com a pouca visibilidade do Now on Tap. Com o assistente, o Google adicionou um conjunto de pontos coloridos animados ao botão inicial que ajudava os usuários a descobrirem e serem lembrados sobre o recurso.

O Android 6.0 finalmente introduziu um sistema de permissões de aplicativos que dava aos usuários um controle granular sobre o que os aplicativos tinham acesso.

Os aplicativos não oferecem mais uma lista enorme de permissões durante a instalação. Com Marshmallow, aplicativos instalados sem solicitar nenhuma permissão. Quando os aplicativos precisavam de uma permissão, como acesso à sua localização, câmera, microfone ou lista de contatos, eles perguntavam no momento exato em que precisavam. Durante o uso de um aplicativo, uma caixa de diálogo "Permitir ou negar" apareceu a qualquer momento em que o aplicativo queria uma nova permissão. Alguns fluxos de configuração de aplicativos resolveram isso solicitando algumas permissões importantes na inicialização e tudo o mais apareceu conforme o aplicativo precisava. Isso melhorou a comunicação com o usuário a fim de entender para que servem as permissões - esse aplicativo precisa de acesso à câmera porque você acabou de tocar no botão da câmera.

Antes do Marshmallow, poucos OEMs (do inglês *Original Equipment Manufacturers*) criaram sua própria solução de impressão digital em resposta ao Touch ID da Apple. Mas com Marshmallow, o Google finalmente criou uma API para todo o ecossistema para reconhecimento de impressões digitais. O novo sistema incluía a interface do usuário para registrar impressões digitais, uma tela de bloqueio com impressão digital e APIs que permitiam que os aplicativos protegessem o conteúdo por trás de uma digitalização de impressão digital ou de uma dificuldade da tela de bloqueio.

O Marshmallow expandiu as APIs JobScheduler de economia de energia que foram originalmente introduzidas no Lollipop. O JobScheduler se tornou basicamente um guarda de trânsito de processamentos em segundo plano.

O armazenamento adotável foi um dos melhores recursos do Marshmallow. Ele transformou os cartões SD de um pool de armazenamento secundário em uma solução perfeita de armazenamento mesclado. Coloque um cartão SD, formate-o e você instantaneamente tem mais espaço de armazenamento em seu dispositivo.

A versão 7.0 nos trouxe, finalmente, a capacidade de dividir a tela com mais de um aplicativo simultaneamente.

Nougat fez grandes alterações nos gráficos e no pipeline de sensores para o Daydream VR, a próxima experiência de VR com tecnologia de smartphone do Google. Um novo recurso “Atualização contínua” emprestou um mecanismo de atualização do Chrome OS, que usa partições de sistema duplo para atualizar silenciosamente uma partição em segundo plano enquanto você ainda está usando a outra.

O Android e o Chrome OS também continuam crescendo juntos. Os aplicativos para Android podem ser executados em alguns Chromebooks agora, dando ao SO uma Play Store e um ecossistema de aplicativos. Há rumores de que o futuro do Chrome OS e do Android se aproximará ainda mais, com o nome “Andromeda” – um portmanteau de “Android” e “Chrome” – sendo usado como codinome de um sistema operacional Chrome / Android mesclado.

Android 8.0 Oreo é a 26ª versão do sistema operacional mais popular do mundo. Este ano, o sistema operacional do Google para dispositivos móveis e todo o resto atingiu dois bilhões de usuários ativos mensais – e isso está apenas contando telefones e tablets.

O que todos esses usuários podem esperar da nova versão? O vice-presidente de engenharia da Android, Dave Burke, disse que a versão 8.0 seria sobre “fundações e fundamentos”. Sua equipe foi guiada por uma única pergunta: o que estamos fazendo com o Android para garantir que o Android esteja em um ótimo lugar nos próximos cinco a dez anos?

Dê uma olhada no Oreo e você realmente verá o foco nos fundamentos. O Google está reformulando o sistema de notificações com um novo layout, novos controles e um novo esquema de cores. Ele está assumindo a responsabilidade pela segurança do Android com uma solução de segurança da marca Google.

O processamento em segundo plano de aplicativos foi refinado, o que, esperamos, proporcionará maior duração da bateria e desempenho mais consistente. Houve também algum trabalho no problema de atualização contínua do Android, com o Project Treble permitindo atualizações mais fáceis e dinâmicas, permitindo uma instalação mais fácil por parte dos usuários. E, como em todos os lançamentos, mais partes do Android se tornam mais modulares, com emojis e atualizações de drivers de GPU agora disponíveis sem uma atualização do sistema operacional.

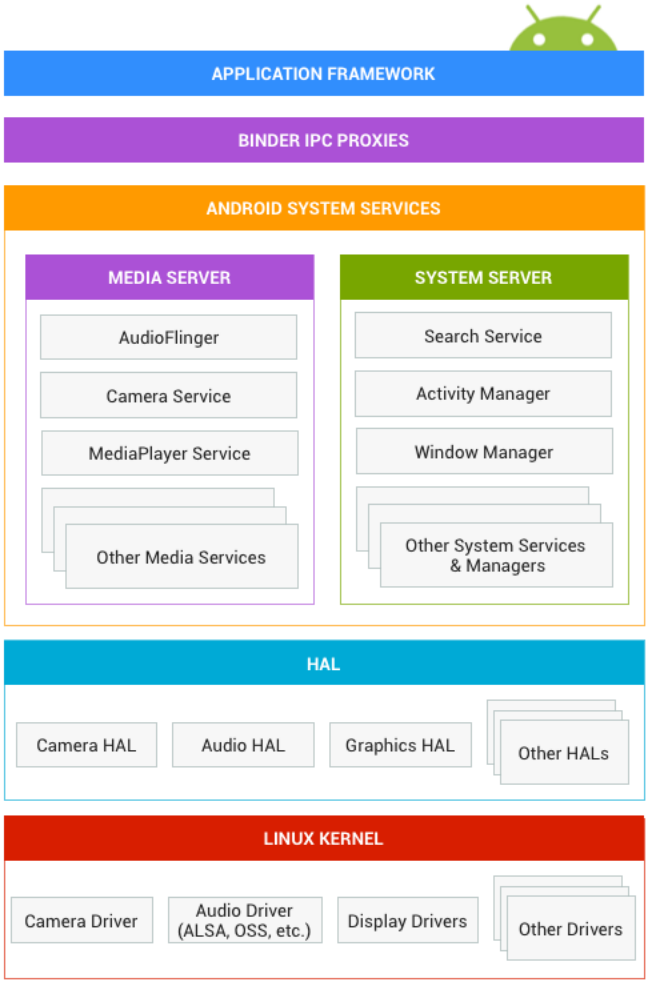
Oreo também servirá como base para três novos fatores de forma do Android. Ele será integrado a carros como “Android Automotive”, no qual o Google trabalha com OEMs de carros para integrar o Android. O Android 8.0 também será o sistema operacional básico do “Android Things”, uma versão “internet das coisas” (IoT) do sistema operacional projetada para gerenciar facilmente em dispositivos embutidos. Finalmente, o grupo de realidade virtual do Google “Daydream” também lançará um novo fator de forma com o Oreo: headsets de realidade virtual autônomos.

Capítulo 2. Arquitetura do Android

Neste capítulo discutiremos sobre a arquitetura do Android: tanto do sistema operacional como a arquitetura de aplicações.

Arquitetura do Sistema Operacional

Fig. 2.1.1 – Arquitetura Android



Fonte: Google (2017), cf. [10]

Iremos discutir com mais detalhes a nova arquitetura do Android, segundo o próprio Google, veja a Fig. 2.1.1.

Application Framework

O *Application Framework* é usado com mais frequência pelos desenvolvedores de aplicativos. Como desenvolvedor de hardware, você deve estar ciente das APIs de desenvolvedor, pois muitos mapeiam diretamente para as interfaces HAL (do inglês *Hardware Abstraction Layer*) subjacentes e podem fornecer informações úteis sobre a implementação de drivers.

Binder IPC

O mecanismo de comunicação entre processos (IPC, do inglês *Binder Inter-Process Communication*) permite que o framework de aplicativos atravesse os limites dos processos e chame os serviços do sistema do Android. Isso permite que os frameworks de APIs de alto nível interajam com os serviços do sistema Android. Isso é transparente ao desenvolvedor e as coisas parecem “apenas funcionar”.

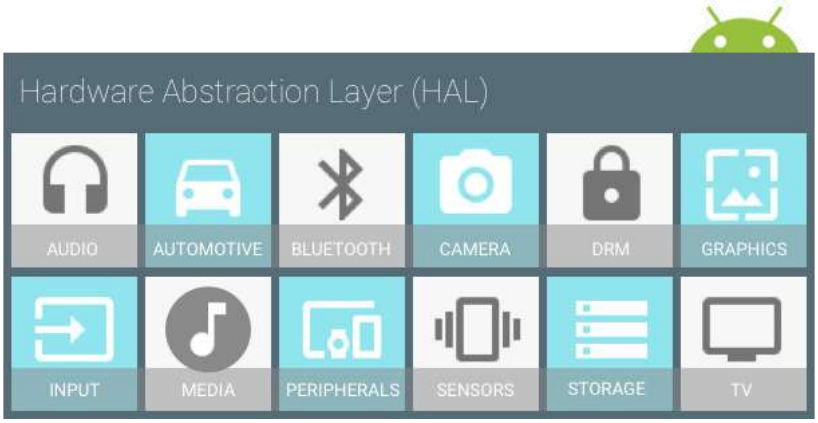
System services

Os serviços do sistema são componentes modulares e focados, como o Gerenciador de Janelas, o Serviço de Pesquisa ou o Gerenciador de Notificações. A funcionalidade exposta pelas APIs da estrutura de aplicativo se comunica com os serviços do sistema para acessar o hardware subjacente. O Android inclui dois grupos de serviços: sistema (como Gerenciador de Janelas e Gerenciador de Notificações) e mídia (serviços envolvidos em reprodução e gravação de mídia).

Hardware abstraction layer (HAL)

Um HAL define uma interface padrão para os fornecedores de hardware implementarem, o que permite que o Android seja agnóstico sobre implementações de driver de nível inferior. O uso de uma HAL permite implementar a funcionalidade sem afetar ou modificar o sistema de nível superior. As implementações do HAL são empacotadas em módulos e carregadas pelo sistema Android no momento apropriado, veja a Fig. 2.1.2.

Fig. 2.1.2 – Componentes do HAL



Fonte: Google (2017), cf. [11].

Linux kernel

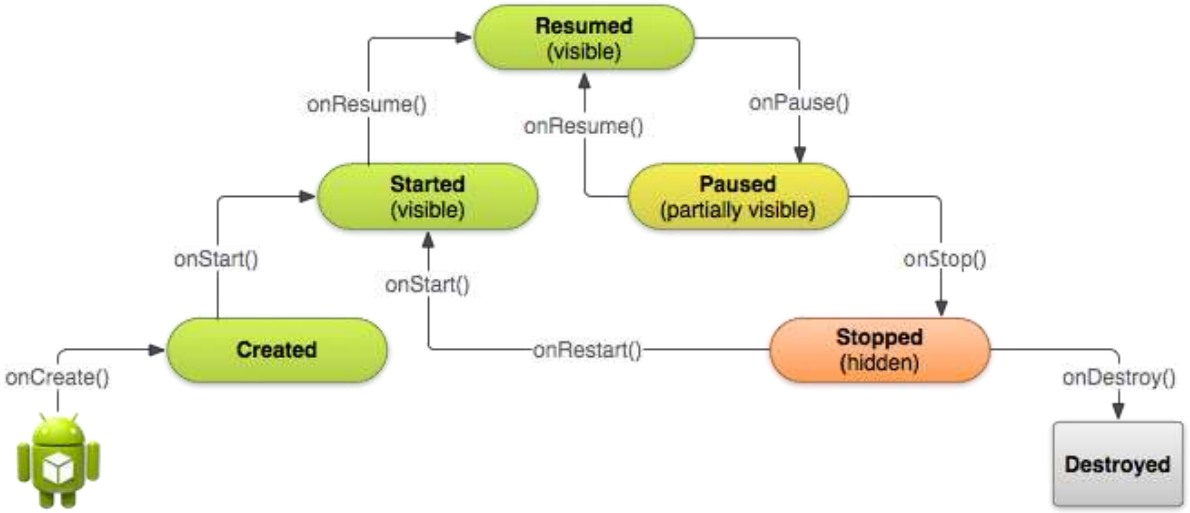
Desenvolver seus drivers de dispositivo é semelhante ao desenvolvimento de um driver de dispositivo típico do Linux. O Android usa uma versão do kernel do Linux com algumas adições especiais, como o Low Memory Killer (um sistema de gerenciamento de memória mais agressivo para preservar memória), wake locks (um serviço do sistema PowerManager), o driver Binder IPC e outros recursos importantes para uma plataforma incorporada móvel. Essas adições são principalmente para a funcionalidade do sistema e não afetam o desenvolvimento do driver.

Ciclo de vida de uma aplicação

Ao contrário de outros paradigmas de programação em que os aplicativos são iniciado com um método main(), o Android inicia o código em uma instância da classe Activity invocando métodos de eventos específicos, que correspondem a fases do seu ciclo de vida. Há uma sequência de métodos de eventos que iniciam uma atividade e uma sequência de métodos de eventos que destroem uma atividade.

Esse ciclo de vida é semelhante a uma cadeia tipo pirâmide, como é mostrada abaixo:

Fig. 2.2.1 - Ciclo de vida de uma aplicação



Os estados da aplicação estão nos balões e os métodos invocados estão escritos nos caminhos das setas.

Dentre os estados da aplicação, três deles podem ser estáticos, são eles:

- **Resumed**: a atividade está em primeiro plano e o usuário pode interagir com ela. Ela está *rodando;*
- **Paused**: a atividade está parcialmente obscurecida por outra atividade. Nesse estado, ela não recebe *input* do usuário, nem executa nenhum código;
- **Stopped**: a atividade está completamente escondida; considerada em *background*. Enquanto parada, a instância da atividade e seus membros são retidos, mas não executam nenhum código.

Os estados chamados transientes são os seguintes:

- **Created**: quando o SO lê o manifest e permite que a aplicação seja criada;
- **Started**: quando a aplicação é criada, ela inicia.

Quando a atividade entra no estado **Paused**, o sistema chama o método onPause(), permitindo a você parar o que não pode acontecer enquanto pausado (*e.g.*, um

vídeo), ou salvar algo permanentemente no celular caso o usuário feche sua aplicação.

Se o usuário voltar para a atividade, o sistema volta para o estado **Resumed** e chama o método onResume().

Outra situação possível é a aplicação para as situações em que a mesma para, conforme listado abaixo:

- O usuário troca a sua aplicação por outra (*e.g.*, via notificação, ou aplicações recentes);
- O usuário abre outra atividade dentro da sua aplicação (e.g., chama o preview da câmera numa outra atividade);
- O usuário recebe uma ligação enquanto usava seu aplicativo.

O Google tem um exemplo que fará você se acostumar com o ciclo de vida. Você pode fazer um check out do projeto via GitHub:

https://github.com/qumaciel/igti_android_activityLifeCycle.git

Importe-o e execute até você conseguir prever o que irá acontecer em cada ação tomada. Para uma explicação mais detalhada sobre o ciclo de vida, veja o vídeo da aula 2.3.

FutebolQuiz

Criaremos uma aplicação chamada **FutebolQuiz**. Essa aplicação faz uma afirmação sobre o futebol brasileiro e a aplicação responde se o usuário acertou ou errou.

Nossa aplicação servirá tanto para introduzir alguns conceitos do material design, como para introduzir a programação de aplicações Android. Ao fim do capítulo, ela terá a aparência semelhante à figura abaixo:

Fig. 2.3.1 - FutebolQuiz

Vamos criar um novo projeto. Eis os detalhes:

- Application name: FutebolQuiz
- Company Domain: android.igti.com.br
- Minimum SDK: API 21 Android 5.0 (Lollipop)
- Template: Blank Activity
- Activity Name: FutebolQuizActivity
- Layout name: activity_futebol_quiz
- Title: FutebolQuiz
- Menu resource name: menu_futebol_quiz

Como nossas aplicações seguirão o conceito do material design *(MD)* precisaremos criar alguns arquivos que servirão de template para todos os outros. Eles seguirão as diretrizes descritas em [12].

Pode-se fazer o check-out do projeto completo via GitHub (endereço: https://github.com/qumaciel/igti_android_futebolquiz.git).

Clique com o botão direito na pasta res/values e crie um novo arquivo de recurso chamado **color.xml**. Este arquivo definirá as cores padrões que você viu na figura acima.

values/color.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="primary">#4DB6AC</color>
    <color name="primary_dark">#009688</color>
    <color name="primary_light">#B2DFDB</color>
    <color name="accent">#FFEB3B</color>
</resources>
```

Agora vamos aplicar essas cores no tema material em res/values/styles.xml. O styles.xml é o arquivo que define o tema da sua aplicação.

values/styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="AppTheme" parent="android:Theme.Material.Light">
        <!--  cor da barra da app -->
        <item name="android:colorPrimary">@color/primary</item>
        <!--   a variação escura para a barra de status  -->
        <item name="android:colorPrimaryDark">@color/primary_dark</item>
        <!--   tema dos controles da app, como botões, checkboxes -->
        <item name="android:colorAccent">@color/accent</item>
    </style>
</resources>
```

Logo em seguida, clique com o botão direito na pasta res/values e crie um novo arquivo de recurso chamado **template-dimens.xml**. Este arquivo definirá o tamanho padrão das margens.

values/template-dimens.xml

values/template-dimens.xml

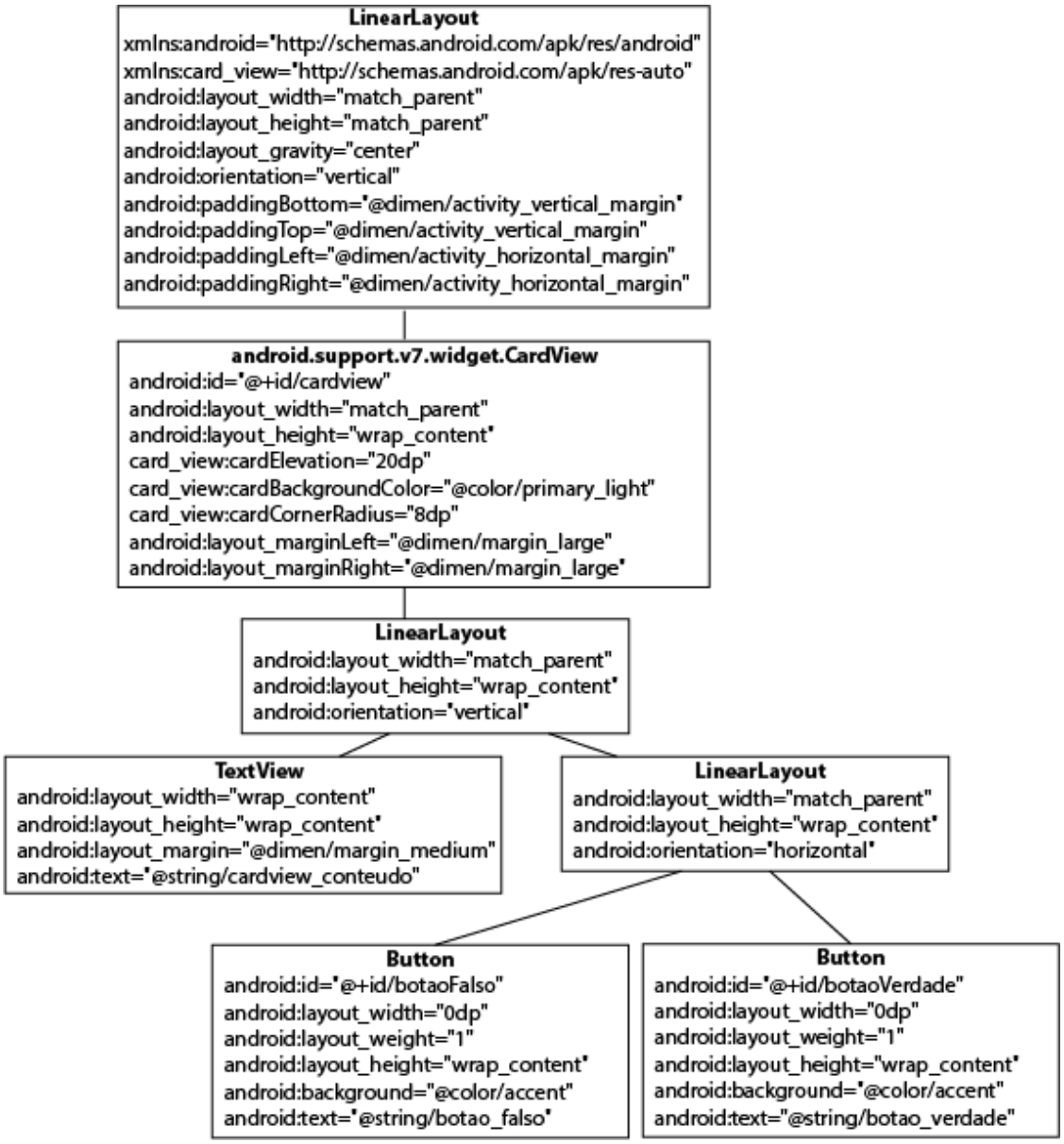
```
<resources>
   <dimen name="activity_horizontal_margin">16dp</dimen>
   <dimen name="activity_vertical_margin">16dp</dimen>
   <dimen name="margin_tiny">4dp</dimen>
   <dimen name="margin_small">8dp</dimen>
   <dimen name="margin_medium">16dp</dimen>
   <dimen name="margin_large">32dp</dimen>
   <dimen name="margin_huge">64dp</dimen>
   <dimen name="horizontal_page_margin">@dimen/margin_medium</dimen>
   <dimen name="vertical_page_margin">@dimen/margin_medium</dimen>
</resources>
```

Pronto, o básico para criar nossa aplicação utilizando MD já está configurado, agora precisamos desenhar nossa solução de layout.

Nossa aplicação é simples: ela faz uma afirmação e o usuário interage, dizendo se a afirmação é verdadeira ou falsa. Perceba que esse tipo de informação é muito parecido com o desenho da solução da aplicação Google Now e Google Keep. É um tipo de aplicação que funcionará bem no conceito de *cards*.

Um *card* é um elemento do layout que chamamos de *widget*. Widgets são os blocos que compõe a interface com o usuário, podem ser um texto, um botão, uma imagem, etc. Eles podem ser, também, o que chamamos de ViewGroup, que são widgets que organizam e são uma espécie de containers para outros widgets.

Para entender melhor, vamos visualizar a hierarquia da nossa visualização planejada:



Cada elemento tem um conjunto de atributos XML. Cada atributo é uma instrução de como o widget deve ser configurado.

O elemento raiz dessa hierarquia é o primeiro LinearLayout. Como é o elemento raiz, ele deve especificar o namespace do android em http://schemas.android.com/apk/res-auto. O outro namespace está ali porque o CardView é um widget com algumas propriedades customizadas, que precisam de um namespace para serem validadas corretamente.

LinearLayout e o CardView estendem uma classe de visualização do tipo ViewGroup. A utilização de ViewGroups se dá quando há a necessidade de organizar os widgets em uma linha, ou colunas, por exemplo.

A raiz LinearLayout tem um filho CardView, que tem um outro LinearLayout como filho, que tem um TextView e um outro LinearLayout como filhos, e esse último LinearLayout tem mais dois Buttons como filhos.

Vamos falar sobre alguns atributos que utilizamos para configurar nosso layout

- **android:layout_width** e **layout_height**

A largura e altura são necessárias em quase todos os widgets, geralmente são do tipo match_parent ou wrap_content. Para uma explicação mais detalhada, veja o vídeo do capítulo.

- **android:orientation**

A orientação nos LinearLayouts determina como os widgets filhos serão organizados. A ordem que eles são definidos altera a ordem que eles são exibidos na tela.

- **android:text**

O TextView e os botões têm esse atributo que diz qual texto será exibido na tela. Note que os valores não estão de forma literal. Eles são referenciados para recursos do tipo string. Para definir esses recursos, abra o arquivo res/values/strings.xml e faça as seguintes alterações:

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">FutebolQuiz</string>
    <string name="action_settings">Settings</string>
    <string name="cardview_conteudo">Joinville foi campeão da Série C em 2011 em uma campanha
        impecável. Na final, venceu o CRB por 1x3 em Maceió e por 4x0 em Joinville.</string>
    <string name="botao_falso">FALSO</string>
    <string name="botao_verdade">VERDADE</string>
    <string name="toast_errou">Errou!</string>
    <string name="toast_acertou">Acertou</string>
</resources>
```

Podemos, finalmente, escrever nosso layout. Abra activity_futebol_quiz.xml e faça as seguintes modificações:

res/layout/activity_futebol_quiz.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center"
        android:orientation="vertical"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin" >

  <android.support.v7.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/cardview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    card_view:cardElevation="20dp"
    card_view:cardBackgroundColor="@color/primary_light"
    card_view:cardCornerRadius="8dp"
    android:layout_marginLeft="@dimen/margin_large"
    android:layout_marginRight="@dimen/margin_large">

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">
      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/margin_medium"
        android:text="@string/cardview_conteudo" />
      <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
          android:id="@+id/botaoFalso"
          android:layout_width="0dp"
          android:layout_weight="1"
          android:layout_height="wrap_content"
          android:background="@color/accent"
          android:text="@string/botao_falso" />
        <Button
          android:id="@+id/botaoVerdade"
          android:layout_width="0dp"
          android:layout_weight="1"
          android:layout_height="wrap_content"
          android:background="@color/accent"
          android:text="@string/botao_verdade" />
      </LinearLayout>
    </LinearLayout>
  </android.support.v7.widget.CardView>
</LinearLayout>
```

Agora é hora de executar a aplicação e ver que ela está exatamente como mostramos na figura. O que falta é adicionar a funcionalidade a esse layout.

A pergunta que fica é: como os objetos definidos nos layouts serão acessados na nossa classe FutebolQuizActivity?

Abra a classe FutebolQuizActivity na árvore de arquivos do projeto. Chamo aqui a atenção para o método sobrescrito onCreate(Bundle). Perceba que, nesse método, damos à aplicação uma interface com o usuário chamando o método

public void setContentView(int layoutResID)

Esse método *infla* o layout que criamos e o mostra na tela. O layout escolhido é passado utilizando o ID do recurso de layout. Esse método pode inflar na tela qualquer objeto que estenda a classe View.

O layout é um recurso, e um recurso é uma parte de sua aplicação que não é um código java. Para acessar o recurso no java, utilizamos o ID do recurso, *e.g.*, o ID do nosso layout criado é R.layout.activity_futebol_quiz.

O Android gera esse ID para cada arquivo de recurso, mas não gera IDs para os widgets dentro deles, por isso utilizamos o atributo abaixo, por exemplo.

- **android:id="@+id/botaoFalso"**

Esse atributo cria um ID para esse widget, que pode ser acessado via R.id.botaoFalso.

Vamos, agora, instanciar e referenciar nossos botões em nossa atividade.

FutebolQuizActivity

```
public class FutebolQuizActivity extends Activity {

  private Button mBotaoVerdade;
  private Button mBotaoFalso;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_futebol_quiz);

    mBotaoVerdade = (Button)findViewById(R.id.botaoVerdade);
    mBotaoFalso = (Button)findViewById(R.id.botaoFalso);
      }
...
}
```

As reticências na caixa de texto de FutebolQuizActivity aparecerão com frequência e indicam outros pedaços de código na classe – que devem ser desconsiderados.

Numa atividade, podemos referenciar objetos inflados utilizando o método de Activity

public View findViewById(int id).

Agora vamos dar uma funcionalidade a esses botões adicionando listeners a eles. Existem duas formas de se fazer isso, de forma explicita e de forma anônima. Sugiro sempre utilizar a forma explicita, pois é sabido que, dessa forma, ganha-se performance na aplicação. E não queremos sacrificar nem um pouco a performance de nossas aplicações, mesmo que sejam pequenas.

Forma explícita

```
…
private View.OnClickListener mBotaoVerdadeListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      // Alguma funcionalidade do botão quando clicado
    }
};

...

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_futebol_quiz);

    mBotaoVerdade = (Button)findViewById(R.id.botaoVerdade);
    mBotaoVerdade.setOnClickListener(mBotaoVerdadeListener);
…
}

...
```

Forma anônima

```
…
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_futebol_quiz);

        mBotaoVerdade = (Button)findViewById(R.id.botaoVerdade);
        mBotaoVerdade.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Alguma funcionalidade do botão quando clicado
            }
        });
…
```

Em alguns casos, a forma anônima parece conveniente, pois a declaração da funcionalidade vem junto com a adição do listener, e não temos duas coisas separadas. Isso fica a seu critério.

O que falta agora para nossa aplicação é responder, caso o usuário tenha respondido à pergunta corretamente ou não. Iremos criar uma Toast para

responder. Uma toast é uma mensagem que aparecerá para o usuário rapidamente, depois some.

Em nossa atividade, faça as seguintes modificações:

FutebolQuizActivity

```
…
   private Button mBotaoVerdade;
   private Button mBotaoFalso;

   private View.OnClickListener mBotaoVerdadeListener = new View.OnClickListener() {
      @Override
      public void onClick(View v) {
         Toast.makeText(FutebolQuizActivity.this, R.string.toast_acertou,
Toast.LENGTH_SHORT).show();
      }
   };

   private View.OnClickListener mBotaoFalsoListener = new View.OnClickListener() {
      @Override
      public void onClick(View v) {
         Toast.makeText(FutebolQuizActivity.this, R.string.toast_errou,
Toast.LENGTH_SHORT).show();
      }
   };

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_futebol_quiz);

      mBotaoVerdade = (Button)findViewById(R.id.botaoVerdade);
      mBotaoVerdade.setOnClickListener(mBotaoVerdadeListener);
      mBotaoFalso = (Button)findViewById(R.id.botaoFalso);
      mBotaoFalso.setOnClickListener(mBotaoFalsoListener);
   }
…
```

Teste a aplicação e veja se está tudo como deve ser.

Model-View-Controller

Agora colocaremos nossa aplicação no esquema Model-View-Controller que discutimos no vídeo do capítulo.

Clique com o botão direito no pacote com a classe java, crie uma nova classe chamada Pergunta e implemente-a da seguinte forma:

Pergunta

```
public class Pergunta {
    private int mQuestao;

    private boolean mQuestaoVerdadeira;

    public Pergunta(int questao, boolean questaoVerdadeira) {
        mQuestao = questao;
        mQuestaoVerdadeira = questaoVerdadeira;
    }

    public int getQuestao() {
        return mQuestao;
    }

    public void setQuestao(int questao) {
        mQuestao = questao;
    }

    public boolean isQuestaoVerdadeira() {
        return mQuestaoVerdadeira;
    }

    public void setQuestaoVerdadeira(boolean questaoVerdadeira) {
        mQuestaoVerdadeira = questaoVerdadeira;
    }
}
```

Com a classe modelo pronta, podemos atualizar nossa atividade. Primeiro adicionamos um ID ao TextView de conteúdo e removemos o texto dela, para que seja atualizado via atividade.

activity_futebol_quiz.xml

```
…
<TextView
            android:id="@+id/cardviewConteudo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="@dimen/margin_medium"
            android:text="@string/cardview_conteudo" />
…
```

Agora vamos à nossa classe controladora fazer as seguintes modificações

FutebolQuizActivity

```
…
   private Button mBotaoFalso;
    private TextView mConteudoCard;

    private Pergunta[] mPerguntas = new Pergunta[]{
            new Pergunta(R.string.cardview_conteudo,true)
    };

    private int mIndiceAtual = 0;

    private View.OnClickListener mBotaoVerdadeListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(FutebolQuizActivity.this, R.string.toast_acertou,
Toast.LENGTH_SHORT).show();
            checaResposta(true);
        }
    };

    private View.OnClickListener mBotaoFalsoListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(FutebolQuizActivity.this, R.string.toast_errou,
Toast.LENGTH_SHORT).show();
            checaResposta(false);
        }
    };

    private void checaResposta(boolean botaoPressionado) {
        boolean resposta = mPerguntas[mIndiceAtual].isQuestaoVerdadeira();
       int recursoRespostaId = 0;
       if (botaoPressionado == resposta) {
            recursoRespostaId = R.string.toast_acertou;
        } else {
            recursoRespostaId = R.string.toast_errou;
        }

        Toast.makeText(this, recursoRespostaId, Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_futebol_quiz);

        mBotaoVerdade = (Button)findViewById(R.id.botaoVerdade);
        mBotaoVerdade.setOnClickListener(mBotaoVerdadeListener);
        mBotaoFalso = (Button)findViewById(R.id.botaoFalso);
        mBotaoFalso.setOnClickListener(mBotaoFalsoListener);

        mConteudoCard = (TextView)findViewById(R.id.cardviewConteudo);
        int questao = mPerguntas[mIndiceAtual].getQuestao();
        mConteudoCard.setText(questao);
    }
…
```

Pronto, nosso projeto já está de acordo com o Model-View-Controller e preparado para receber novas perguntas, por exemplo:

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">FutebolQuiz</string>
    <string name="action_settings">Settings</string>
    <string name="cardview_conteudo">Joinville foi campeão da Série C em 2011 em uma campanha
        impecável. Na final, venceu o CRB por 1x3 em Maceió e por 4x0 em Joinville.</string>
<string name="cardview_conteudo_joinville">Joinville foi campeão da Série C em 2011 em uma
campanha
    impecável. Na final, venceu o CRB por 1x3 em Maceió e por 4x0 em Joinville.</string>
<string name="cardview_conteudo_cruzeiro">O Cruzeiro foi campeão da Série A em 2014, ganhando
    com 5 rodadas de antecedência.</string>
<string name="cardview_conteudo_gremio">O Grêmio foi campeão do Mundial Interclubes em 1996,
ganhando
    do Ajax por um placar de 2x1.</string>

    <string name="botao_falso">FALSO</string>
    <string name="botao_verdade">VERDADE</string>
    <string name="toast_errou">Errou!</string>
    <string name="toast_acertou">Acertou</string>
</resources>
```

FutebolQuizActivity

```
…
    private Pergunta[] mPerguntas = new Pergunta[]{
            new Pergunta(R.string.cardview_conteudo,true)
    };

private Pergunta[] mPerguntas = new Pergunta[]{
        new Pergunta(R.string.cardview_conteudo_joinville,true),
        new Pergunta(R.string.cardview_conteudo_cruzeiro,false),
        new Pergunta(R.string.cardview_conteudo_gremio,false)
};

...
```

GitHub

Você pode importar fazer um *check out* do projeto via GitHub:

https://github.com/qumaciel/igti_android_activityLifeCycle.git

https://github.com/qumaciel/igti_android_futebolquiz.git

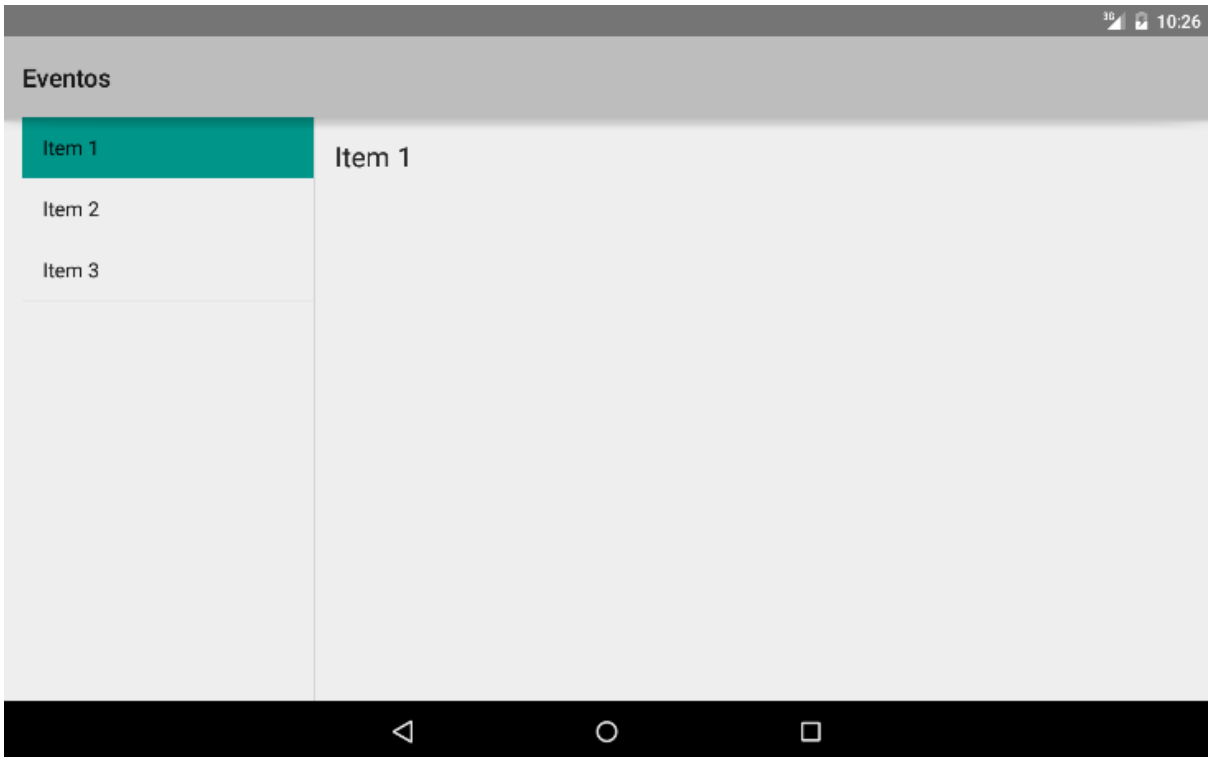
Capítulo 3. Fragmentos

Neste capítulo discutiremos sobre a necessidade de aumentar a flexibilidade da interface com o usuário. Uma aplicação na qual somente a atividade gerencia a IU pode não ser flexível o suficiente, dependendo da necessidade da aplicação.

Flexibilidade

Pense numa aplicação do tipo *list-detail* (*e.g.*, Gmail), como a aplicação abaixo:

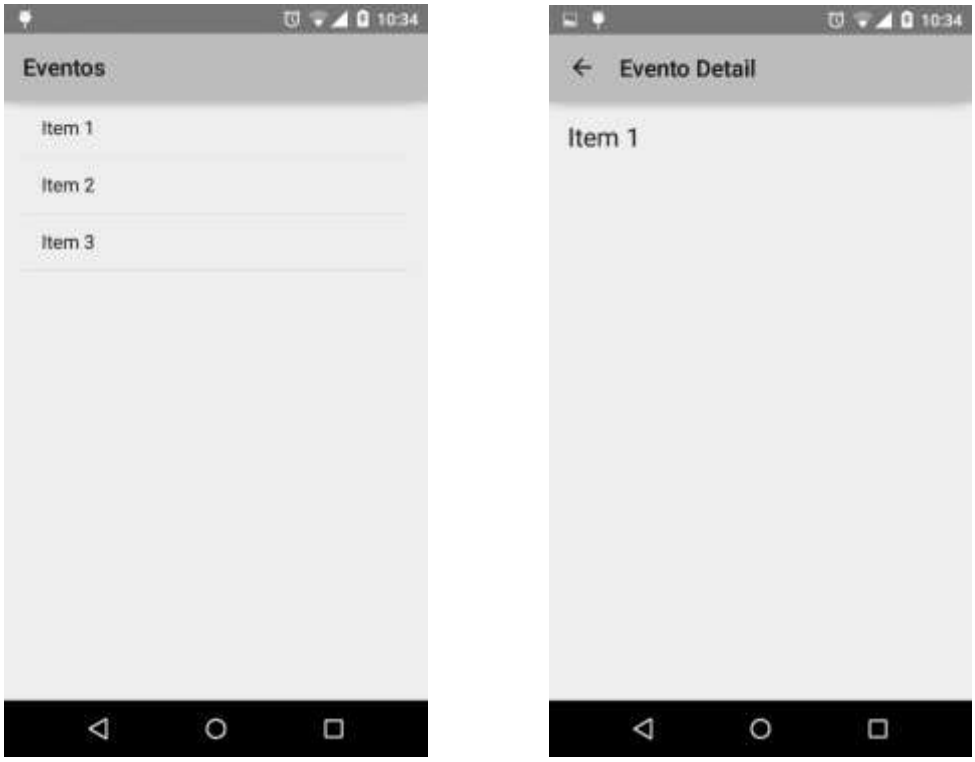
Fig. 3.1.1 - Aplicação tipo list-detail em um tablet



Você já deve ter imaginado que essa aplicação tem, no mínimo, duas atividades: uma para gerenciar a lista dos itens e outra para apresentar os detalhes de cada item.

O primeiro desenho da solução desse tipo de aplicação começa pela interface dependente do tamanho do dispositivo, *i.e.*, caso o usuário esteja num tablet, onde a área é grande, podemos ter uma IU como mostrada acima; caso o usuário esteja num telefone, teremos que *fragmentar* essa interface em lista e detalhe, como mostrado abaixo:

Fig 3.1.2 - Aplicação list-detail em um celular



O que esses dois cenários têm em comum é o fato de os dois necessitarem que a interface com o usuário seja flexível o suficiente para compor e recompor parcialmente, ou totalmente, a visualização, dependendo da necessidade do usuário.

As instâncias de Activity não foram feitas para nos dar essa flexibilidade: assim que inflamos o layout da atividade com o setContentView(…), a atividade fica amarrada em sua visualização *até que a morte os separe*! Essa é a lei do Android.

É possível circundar essa lei do Android transferindo o gerenciamento da IU para um ou mais *fragmentos.*

Um *fragmento* é um objeto controlador que uma atividade pode delegar para executar tarefas. Geralmente delegamos a tarefa de controlar total ou parcialmente a interface com o usuário.

Um fragmento tem uma visualização própria que pode estar associada a um arquivo de layout, assim como era nossa atividade. Como nossa atividade permanece a mesma durante o ciclo de vida da aplicação, nenhuma lei é violada. Pode-se pensar na aplicação da seguinte forma:

Fig 3.1.3 - Atividades e fragmentos

Como vemos na figura 3.1.3, nossa atividade fica composta de dois fragmentos, nos quais, quando o usuário clica no item da lista, o fragmento 2 é removido e substituído por outro fragmento com os detalhes do novo item.
Vamos criar nossa aplicação tipo *list-detail*:

- Application name: Eventos
- Company Domain: android.igti.com.br
- Minimum SDK: API 21 Android 5.0 (Lollipop)
- Template: Blank Activity
- Activity Name: EventoListActivity
- Layout name: activity_evento_list
- Title: Eventos
- Menu resource name: menu_eventos

Primeiramente, temos que ter duas visualizações possíveis: uma para celulares e outra para tablets. Abra nosso layout e substitua pelo seguinte:

layout/activity_evento_list.xml

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/evento_list"
        android:name="br.com.igti.android.eventos.EventoListFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        tools:context=".EventoListActivity"
        tools:layout="@android:layout/list_content"/>
```

Podemos perceber que nesse layout haverá um fragmento com o ID *evento_list*, e utilizamos um layout padrão do Android para fazer um fragmento do tipo lista. Esse layout pode ser personalizado, mas, para esse exemplo, o layout padrão nos serve.

Agora, precisamos da visualização para o tablet. Podemos utilizar um sufixo na pasta layout para associar recursos diferentes a dispositivos com configurações diferentes. No nosso caso, utilizaremos o sufixo “-sw600dp”. O “sw" indica que queremos a propriedade *menor largura* e definimos essa menor largura como 600dp. Seu dispositivo acessará os recursos dessa pasta somente se possuir, *pelo menos*, 600dp na menor largura, *i.e.*, no modo retrato.

Assim, criaremos um layout com o mesmo nome, mas na pasta res/layout-sw600dp/:

layout-sw600dp/activity_evento_list.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:baselineAligned="false"
        android:divider="?android:attr/dividerHorizontal"
        android:orientation="horizontal"
        android:showDividers="middle"
        tools:context=".EventoListActivity">
    <fragment
        android:id="@+id/evento_list"
        android:name="br.com.igti.android.eventos.EventoListFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        tools:layout="@android:layout/list_content"/>

    <FrameLayout
        android:id="@+id/evento_detail_container"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3"/>

</LinearLayout>
```

Perceba que, agora, temos um LinearLayout para organizar os dois fragmentos na tela. O primeiro fragmento, o da lista, foi colocado explicitamente via a tag **fragment**, já o segundo gerenciaremos pelo Java. Para este, reservaremos apenas um quadro com um ID para referenciarmos depois. Colocamos as duas formas aqui

para que vejam que é possível, mas você pode adotar somente a estratégia dos quadros, por exemplo, e gerenciar tudo pelo Java depois.

Hora de pensar no modelo da nossa aplicação. Iremos trabalhar com essa aplicação durante o curso e, neste capítulo, introduziremos, apenas a funcionalidade básica dela. Criaremos uma aplicação tipo TODO-list, *i.e.*, uma lista de coisas a fazer.

A primeira coisa que precisamos criar é o modelo: tudo o que precisamos fazer é um evento, portanto crie uma nova classe chamada Evento com a seguinte implementação:

Evento

```
public class Evento {
    private UUID mId;
    private String mTitulo;

    public Evento(){
        mId   = UUID.randomUUID();
    }

    public Evento(UUID id, String titulo) {
        mId = id;
        mTitulo = titulo;
    }

    public UUID getId() {
        return mId;
    }

    public void setId(UUID id) {
        mId = id;
    }

    public void setTitulo(String titulo) {
        mTitulo = titulo;
```

Evento

```
    }

    @Override
    public String toString() {
        return mTitulo;
    }
}
```

Note que, ao invés do getter tradicional do título, sobrescrevemos o método **toString()**, pois a lista padrão do Android que utilizamos mostra o item na lista de acordo com o que sai desse método.

Como mais para frente iremos adicionar/remover/modificar esses eventos, precisaremos, também, de um gerenciador desses eventos. Por isso, já criaremos a implementação de um gerenciador singleton simples chamado EventosManager:

EventosManager

```
public class EventosManager {

    private ArrayList<Evento> mEventos;

    private static EventosManager sEventos;
    private Context mAppContext;

    private EventosManager(Context appContext) {
        mAppContext = appContext;
        mEventos = new ArrayList<Evento>();

        for(int i =0; i<4; i++)  {
            Evento evento = new Evento();
            evento.setTitulo("Item " + i);
            mEventos.add(evento);
        }
```

EventosManager

```
    }

    public static EventosManager get(Context c) {
        if (sEventos == null) {
            sEventos = new EventosManager(c.getApplicationContext());
        }
        return sEventos;
    }

    public ArrayList<Evento> getEventos() {
        return mEventos;
    }

    public Evento getEvento(UUID id) {
        for(Evento evento : mEventos) {
            if(evento.getId().equals(id)) {
                return evento;
            }
        }
        return null;
    }

    public void addEvento(Evento evento) {
        mEventos.add(evento);
    }

    public void deleteEvento(Evento evento) {
        mEventos.remove(evento);
    }
}
```

Essa é a nossa implementação simples com três itens adicionados no construtor. O Context é passado, pois o gerenciador, como Singleton, deve saber qual é o contexto (Atividade) em que o gerenciador é chamado.

Agora que nossa camada modelo está pronta, vamos à atividade e aos fragmentos, que irão controlar essa aplicação, amarrando a visualização com o modelo.

Primeiro, queremos apenas visualizar os dois fragmentos. Depois, colocaremos uma funcionalidade neles. Primeiro, faremos o fragmento que controlará a lista. Crie uma classe chamada **EventoListFragment**:

EventoListFragment

```
public class EventoListFragment extends ListFragment {
    // Chave utilizada para o onSaveInstanceState
    private static final String KEY_POSICAO_ATUAL = "posicao_atual";

    // uma implementação burra do Callback que não fará nada
    // apenas para quando o fagmento não estiver anexado na atividade
    private static Callbacks sDummyCallbacks = new Callbacks() {
        @Override
        public void onItemSelected(UUID id) {
        }
    };

    // o callback do fragmento que notifica é notificado sobre o clique na
    private Callbacks mCallbacks = sDummyCallbacks;

    // Item atual
    private int mPosicaoAtual = ListView.INVALID_POSITION;

    // O Construtor do fragmento é obrigatório
    public EventoListFragment() {    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setListAdapter(new ArrayAdapter<Evento>(
                getActivity(),  // contexto
                android.R.layout.simple_list_item_activated_1, // template do item
```

EventoListFragment

```
            android.R.id.text1,
            EventosManager.get(getActivity()).getEventos()));
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        // Resolve o problema de girar o celular
        if (savedInstanceState != null
                && savedInstanceState.containsKey(KEY_POSICAO_ATUAL)) {
            setActivatedPosition(savedInstanceState.getInt(KEY_POSICAO_ATUAL));
        }
    }

@Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        // Quando o fragmento é anexado na atividade
        // garantimos que a atividade implementa
        // os Callbacks para os cliques
        if (!(activity instanceof Callbacks)) {
            throw new IllegalStateException("A atividade tem que implementar os Callbacks");
        }

        mCallbacks = (Callbacks) activity;
    }

    @Override
    public void onDetach() {
        super.onDetach();

        // Se o fragmento é desanexado
        // resetamos os callbacks
        mCallbacks = sDummyCallbacks;
```

EventoListFragment

```
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        if (mPosicaoAtual != ListView.INVALID_POSITION) {
            // colocamos a posicao atual
            outState.putInt(KEY_POSICAO_ATUAL, mPosicaoAtual);
        }
    }

    @Override
    public void onListItemClick(ListView listView, View view, int position, long id) {
        super.onListItemClick(listView, view, position, id);

        Evento evento = (Evento)getListAdapter().getItem(position);
        // Notificamos que um item foi selecionado.
        mCallbacks.onItemSelected(evento.getId());
    }

    // configura a lista quando o item é clicado e o "ativa"
    public void setActivateOnItemClick(boolean activateOnItemClick) {
        getListView().setChoiceMode(activateOnItemClick
                ? ListView.CHOICE_MODE_SINGLE
                : ListView.CHOICE_MODE_NONE);
    }

private void setActivatedPosition(int position) {
        if (position == ListView.INVALID_POSITION) {
            getListView().setItemChecked(mPosicaoAtual, false);
        } else {
            getListView().setItemChecked(position, true);
        }

        mPosicaoAtual = position;
    }
```

EventoListFragment

```
    // Uma Callback interface para disparar o evento
    // quando um item for clicado
    public interface Callbacks {
        // Evento para quando o item for selecionado
        public void onItemSelected(UUID id);
    }
}
```

Bom, a classe é maior até agora e vários comentários sobre ela merecem ser feitos. Primeiramente, nossa classe estende a classe **ListFragment** que já é uma classe preparada para lidar com uma lista do tipo que queremos em nosso fragmento. Esse fragmento consiste de um **ListView**.

Depois, declaramos a chave **KEY_POSICAO_ATUAL**, já pensando naquele problema que tivemos no capítulo passado, quando giramos a tela.

Em seguida, temos uma implementação de um **Callback**. Esse callback será responsável por disparar os eventos quando o usuário clica num item da lista. Como estamos num fragmento, ele pode estar anexado à atividade, ou não. Quando ele não estiver anexado à atividade, configuramos esse callback “burro" para que sua aplicação não dê problema.

Vemos mais embaixo, no evento onCreate(), que chamamos o método da lista

public void setListAdapter(ListAdapter).

Nosso ListView pede um adaptador, que será responsável por criar os objetos de visualização necessários, popular a lista com os dados da camada modelo e retornar um objeto View para a nossa lista. Chamamos um **ArrayAdapter<Evento>**

que implementa a interface **Adapter** e sabe lidar com uma lista de objetos do tipo **Evento**. E pedimos essa lista para o nosso gerenciador de eventos.

Depois de lidar com o problema de girar a tela, vemos o método **onAttach(Activity)**, que é específico para fragmentos. Esse evento é disparado quando anexamos o fragmento à atividade. Lembrando que fragmentos são como caixas em nossa atividade, podemos anexar ou remover a visualização deles, sem destruir o objeto.

Os métodos seguintes configuram o comportamento do ListView quando o item é clicado. Por exemplo, no método

ListView.setChoiceMode(int)

Configuramos que o item deve parecer “ativado" quando clicado. Agora vamos configurar nossa atividade para receber esse Fragmento. Abra o **EventoListActivity** e substitua pela seguinte implementação:

EventoListActivity

```
public class EventoListActivity extends Activity implements EventoListFragment.Callbacks {

    private boolean mTwoPane;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_evento_list);

        if (findViewById(R.id.evento_detail_container) != null) {
            // O container só existirá, caso o dispositivo tenha utilizado
            // o recurso de layout-sw600dp
            mTwoPane = true;

            // se estiver no modo “tablet" o item da lista deve permanecer
            // “ativado” quando clicado
            ((EventoListFragment) getFragmentManager()
                    .findFragmentById(R.id.evento_list))
                    .setActivateOnItemClick(true);
        }
    }

    // Implementação do Callback da lista
    @Override
    public void onItemSelected(UUID id) {
        // Sem implementação por enquanto
    }
}
```

Aqui você vê uma chamada ao **FragmentManager**. Falaremos dele daqui a pouco.

Vemos que a atividade serve apenas como um *host* para o fragmento. Perceba que, como a implementação do Callback está na atividade, o fragmento da lista é *completamente independente* do fragmento do detalhe, ele apenas dispara o

evento. Será a atividade a responsável por lidar com esse evento e decidir o que fazer. Quando lidamos com fragmentos, é sempre bom garantir que tenhamos *fragmentos independentes* nesse sentido.

Duas coisas podem ser feitas quando o usuário clica num item da lista: se estiver num celular, precisaremos criar outra atividade e mostrá-la na tela; se o usuário estiver num tablet, devemos anexar o fragmento referente ao detalhe.

Aqui vemos a flexibilidade do fragmento em toda sua glória, já que aproveitaremos o mesmo fragmento nas duas situações, evitando criar uma interface para cada cenário.

Vamos pensar na primeira situação: o celular. Num celular, não haverá espaço para criar o fragmento do detalhe, portanto o melhor a fazer é criar uma nova atividade para esse fragmento.

Vamos criar um template para que você utilize essa estratégia em todas suas aplicações a partir de hoje.

Primeiro, faremos um template para servir de host para o fragmento. Crie um layout chamado **activity_fragment**:

layout/activity_fragment.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".EventoDetailActivity"
        tools:ignore="MergeRootFrame"/>
```

É um layout simples, apenas com um quadro com um ID para referenciarmos depois.

Agora crie uma nova classe chamada **SingleFragmentActivity**, que será um template para atividades com apenas um fragmento.

SingleFragmentActivity

```
public abstract class SingleFragmentActivity extends Activity {
    protected abstract Fragment createFragment();

    protected int getLayoutResId () { return R.layout.activity_fragment; }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(getLayoutResId());
        Fragment fragment = getFragmentManager().findFragmentById(R.id.evento_detail_container);
        if (savedInstanceState == null) {
            getFragmentManager().beginTransaction()
                    .add(R.id.container, (fragment == null) ? createFragment() : fragment)
                    .commit();
        }
    }
}
```

Aqui você vê o **FragmentManager**. O FragmentManager será o responsável por gerenciar seus fragmentos e adicionar a visualização à hierarquia de visualização da atividade.

Ele faz basicamente duas coisas: a primeira é uma lista dos fragmentos e a outra são transações com eles, (via FragmentTransaction). Essas transações são *adicionar*, *remover*, *anexar*, *desanexar*, ou *substituir* um fragmento na lista de fragmentos. Isso é o *coração* de como você utiliza os fragmentos para compor e recompor a interface com o usuário.

Quando chamamos

```
getFragmentManager().beginTransaction()
                .add(R.id.container, (fragment == null) ? createFragment() : fragment)
                .commit();
```

É o mesmo que dizer “Gerenciador, vou fazer uma transação; ela será para adicionar; faça isso, depois, encerre a transação”.

Agora que nosso template está pronto, vamos criar o layout do detalhe. Crie um layout chamado **fragment_evento_detail**

layout/fragment_evento_detail.xml

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/evento_detail"
        style="?android:attr/textAppearanceLarge"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp"
        android:textIsSelectable="true"
        tools:context=".EventoDetailFragment"/>
```

Esse será um layout simples e mostrará apenas o título do evento. Nos capítulos seguintes, enriqueceremos mais esse layout, mas vamos simplificar por enquanto.

Finalmente, podemos criar a atividade do detalhe. Crie uma classe chamada **EventoDetailActivity** com a seguinte implementação:

EventoDetailActivity

```
public class EventoDetailActivity extends SingleFragmentActivity {


   @Override
   protected Fragment createFragment() {
      UUID id = (UUID) getIntent().getSerializableExtra(EventoDetailFragment.ARG_ID_EVENTO);
      return EventoDetailFragment.newInstance(id);    }
```

EventoDetailActivity

```
}
```

Simples assim! Essa atividade fica somente como um host para o fragmento.

Ela já está preparada para receber a informação do ID, que falaremos em seguida.

Para que uma nova atividade possa ser executada em sua aplicação, é necessário registrá-la no seu manifest. Adicione as seguintes tags em seu manifest:

AndroidManifest.xml

```
…
    <activity
        android:name=".EventoListActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name=".EventoDetailActivity"
        android:label="@string/title_evento_detail"
        android:parentActivityName=".EventoListActivity" >
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value=".EventoListActivity" />
    </activity>
…
```

A diferença entre as duas atividades de sua aplicação está no fato de que sua lista é a atividade principal, e o detalhe é uma atividade secundária. Dessa forma, dizemos isso à aplicação utilizando o **meta-data**.

Antes de criar nosso fragmento de detalhe, vamos voltar à atividade da lista e adicionar a funcionalidade que abrirá o detalhe:

```
…
@Override
   public void onItemSelected(UUID id) {
      if (mTwoPane) {
         // No modo tablet, o detalhe será exibido adicionando
         // ou substituindo o fragmento de detalhe
         EventoDetailFragment fragment = EventoDetailFragment.newInstance(id);
         getFragmentManager().beginTransaction()
               .replace(R.id.evento_detail_container, fragment)
               .commit();

      } else {
         // no celular, chamaremos a nova atividade.
         Intent detailIntent = new Intent(this, EventoDetailActivity.class);
         detailIntent.putExtra(EventoDetailFragment.ARG_ID_EVENTO, id);
         startActivity(detailIntent);
      }
   }
…
```

Primeiro vemos o FragmentManager trabalhando, como discutimos anteriormente, só que, dessa vez, a transação é *substituir*. Que irá substituir, caso ele exista, senão ele cria.

Na parte do celular, vemos que o jeito mais simples de iniciar uma atividade é chamando o método

public void startActivity(Intent)

O objeto Intent é um objeto que um componente pode utilizar para se comunicar com o sistema operacional. No nosso caso, o que queremos dizer ao sistema operacional é qual é a atividade a ser criada.

Veja que nós não podemos simplesmente chamar a atividade (fragmento) sem dizer qual é o ID do evento. Por isso, adicionamos essa informação tanto no fragmento quanto na atividade, e lidaremos com ela em seguida.

Finalmente, implementaremos nossa última classe desse capítulo. Crie uma classe chamada **EventoDetailFragment** com a seguinte implementação:

EventoDetailFragment

```
public class EventoDetailFragment extends Fragment {

    // Chave do ID
    public static final String ARG_ID_EVENTO = "evento_id";

    private Evento mEvento;

    // A instancia do detalhe só será possivel com um ID
    public static EventoDetailFragment newInstance(UUID eventoId) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_ID_EVENTO,eventoId);

        EventoDetailFragment fragment = new EventoDetailFragment();
        fragment.setArguments(args);

        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getArguments().containsKey(ARG_ID_EVENTO)) {
            UUID id = (UUID)getArguments().getSerializable(ARG_ID_EVENTO);
```

EventoDetailFragment

```
            mEvento = EventosManager.get(getActivity()).getEvento(id);
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_evento_detail, container, false);

        if (mEvento != null) {
            ((TextView) rootView.findViewById(R.id.evento_detail)).setText(mEvento.toString());
        }

        return rootView;
    }
}
```

Note que criamos um método chamado newInstance(UUID), pois só será possível criar uma nova instância desse fragmento se ele possuir um ID. Portanto, nem permitimos um construtor nessa classe.

No evento onCreate(…), colocamos a implementação que irá ler o ID enviado pela lista e criar a variável membro do evento corretamente.
Pronto, podemos testar nossa aplicação e ver que tudo está funcionando como deveria.

Prós e contras

Como vimos, a utilização de fragmentos permite dividir o trabalho da atividade em blocos gerenciados independentemente.

Mas, claro, pagamos um preço por essa flexibilidade: há um aumento na complexidade da aplicação, já que existem vários blocos independentes, e, claro, muito mais linhas de código.

Mas o preço se paga de forma que a vantagem é maior que o preço que temos a pagar e, por isso, daqui para frente, todas nossas aplicações serão *fragmentadas.*

GitHub

Você pode fazer um *check out* do projeto via GitHub:
https://github.com/qumaciel/igti_android_eventos.git

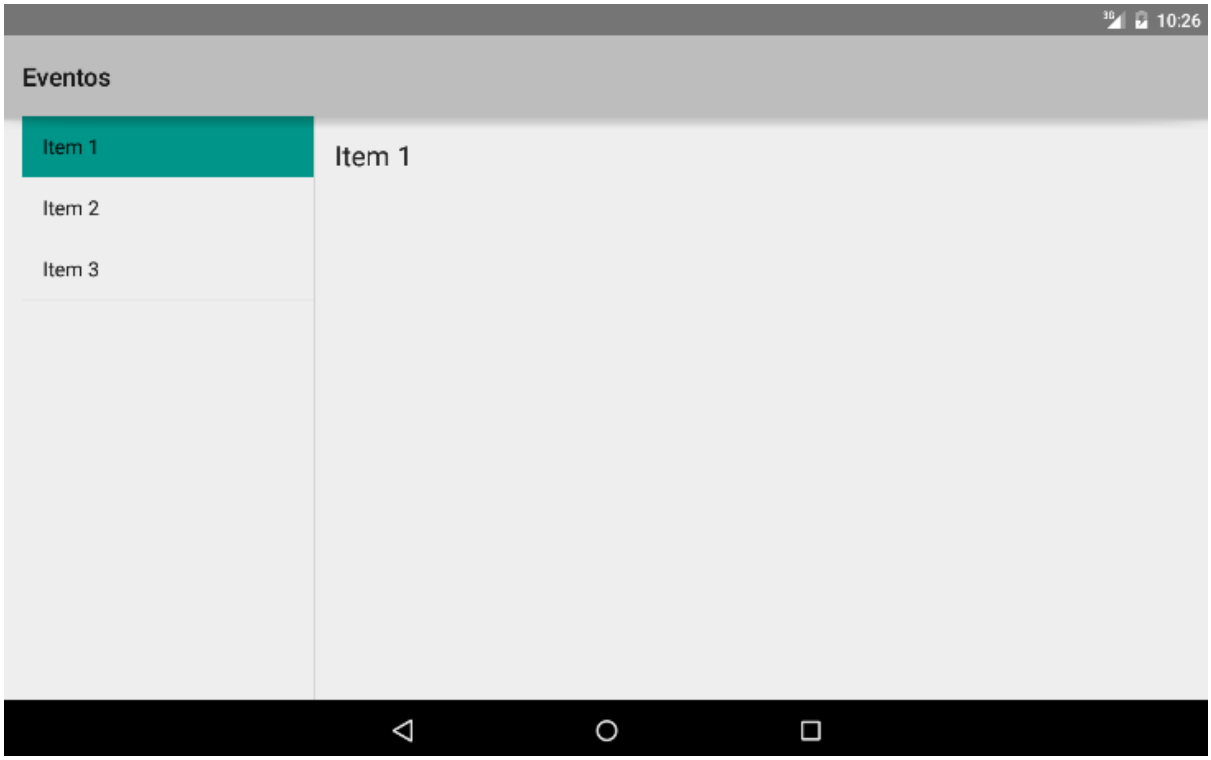
Capítulo 4. Estratégia de Armazenamento de Dados Locais

Neste capítulo discutiremos sobre estratégias de armazenamento de dados locais. Mais especificamente, utilizaremos duas estratégias, primeiro, com arquivos. Para isso utilizaremos nossa aplicação *Eventos* criada no capítulo 3, Fragmentos.

Preparando a IU

Nossa aplicação do capítulo anterior ficou com essa cara:

Fig. 4.1.1 - Aplicação Eventos



Iremos, agora, preparar a interface com o usuário, a fim de que tenhamos uma interface editável, e enriqueceremos mais nossos eventos, para que pareça uma situação real. Primeiro, adicionamos algumas Strings, que estarão presentes em nossa interface.

values/strings.xml

```
…
   <string name="app_name">Eventos</string>
   <string name="title_evento_detail">Evento Detail</string>
   <string name="evento_titulo">Título:</string>
   <string name="evento_desc">Descrição:</string>
…
```

layout/fragment_evento_detail.xml

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:id="@+id/evento_detail"
      style="?android:attr/textAppearanceLarge"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:padding="16dp"
      android:textIsSelectable="true"
      tools:context=“.EventoDetailFragment"/>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:orientation="vertical"
          tools:context=".EventoDetailFragment">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/evento_titulo"/>
   <EditText android:id="@+id/evento_detail"
      android:inputType="text"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:padding="16dp"/>
```

layout/fragment_evento_detail.xml

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/evento_desc"/>
    <EditText android:id="@+id/evento_descricao"
        android:inputType="text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"/>
</LinearLayout>
```

Agora, abra o layout **fragment_evento_detail.xml** e faça as seguintes alterações

O que mudou foi que, agora, ganhamos dois campos editáveis: título e descrição.

Vamos, agora, atualizar nosso modelo para receber essas informações. Abra a classe Evento e adicione o seguinte:

Evento

```
public class Evento {
    private UUID mId;
    private String mTitulo;
    private String mDescricao;

…

    public String getDescricao() {
        return mDescricao;
    }

    public void setDescricao(String descricao) {
        mDescricao = descricao;
    }

…
```

Evento

```
}
```

Agora vamos amarrar o modelo com a visualização. Altere a classe **EventoDetailFragment** da seguinte forma:

EventoDetailFragment

```
public class EventoDetailFragment extends Fragment {

    private EditText mTitulo;
    private EditText mDescricao;

    // uma implementação burra do Callback que não fará nada
    // apenas para quando o fagmento não estiver anexado na atividade
    private static Callbacks sDummyCallbacks = new Callbacks() {
        @Override
        public void onEventoUpdated(Evento evento) {
        }
    };

    // o callback do fragmento que notifica é notificado sobre a edição
    // do evento
    private Callbacks mCallbacks = sDummyCallbacks;

    private TextWatcher mTitutloListener = new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {

        }

        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {
            mEvento.setTitulo(s.toString());
```

EventoDetailFragment

```
            mCallbacks.onEventoUpdated(mEvento);
            getActivity().setTitle(mEvento.toString());
        }

        @Override
        public void afterTextChanged(Editable s) {

        }
    };

    private TextWatcher mDescListener = new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {

        }

        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {
            mEvento.setDescricao(s.toString());
            mCallbacks.onEventoUpdated(mEvento);
        }

        @Override
        public void afterTextChanged(Editable s) {

        }
    };
```

EventoDetailFragment

```
…

@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_evento_detail, container, false);
```

EventoDetailFragment

```
        if (mEvento != null) {
            ((TextView) rootView.findViewById(R.id.evento_detail)).setText(mEvento.toString());
            mTitulo = (EditText) rootView.findViewById(R.id.evento_detail);
            mTitulo.setText(mEvento.toString());
            getActivity().setTitle(mEvento.toString());
            mTitulo.addTextChangedListener(mTitutloListener);

            mDescricao = (EditText) rootView.findViewById(R.id.evento_descricao);
            mDescricao.setText(mEvento.getDescricao());
            mDescricao.addTextChangedListener(mDescListener);
        }

        return rootView;
    }
@Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        // Quando o fragmento é anexado na atividade
        // garantimos que a atividade implementa o Callback
        if (!(activity instanceof Callbacks)) {
            throw new IllegalStateException("A atividade tem que implementar os Callbacks");
        }

        mCallbacks = (Callbacks) activity;
    }

    @Override
    public void onDetach() {
        super.onDetach();

        // Se o fragmento é desanexado
        // resetamos os callbacks
        mCallbacks = sDummyCallbacks;
    }
```

EventoDetailFragment

```
    // Uma Callback interface para disparar o evento
    // quando algo for alterado
    public interface Callbacks {
        // Evento para quando o item for selecionado
        public void onEventoUpdated(Evento evento);
    }
…
```

Em nossas modificações, criamos um Callback para todas as vezes que o título e a descrição forem alterados. A ideia é que, cada vez que alteramos o detalhe, também alteramos a lista, já que podemos alterar o título e essa informação tem que propagar. Abra a classe **EventoListFragment** e implemente o seguinte método:

EventoListFragment

```
…
    public void updateUI() {
        ((ArrayAdapter<Evento>)getListAdapter()).notifyDataSetChanged();
    }
…
```

Esse método do adaptador da lista avisará que houve uma alteração, e o adaptador atualiza a visualização dos itens. Agora, precisamos disparar esse evento na lista. Para isso, implementamos o Callback tanto na atividade do detalhe (caso o usuário esteja num celular), quanto na lista, que é onde faremos a chamada. Abra a classe **EventoDetailActivity:**

EventoListActivity

EventoListActivity

```
~~public class EventoDetailActivity extends SingleFragmentActivity {~~
public class EventoDetailActivity extends SingleFragmentActivity implements
EventoDetailFragment.Callbacks{
…

@Override
   public void onEventoUpdated(Evento evento) {   }
…
```

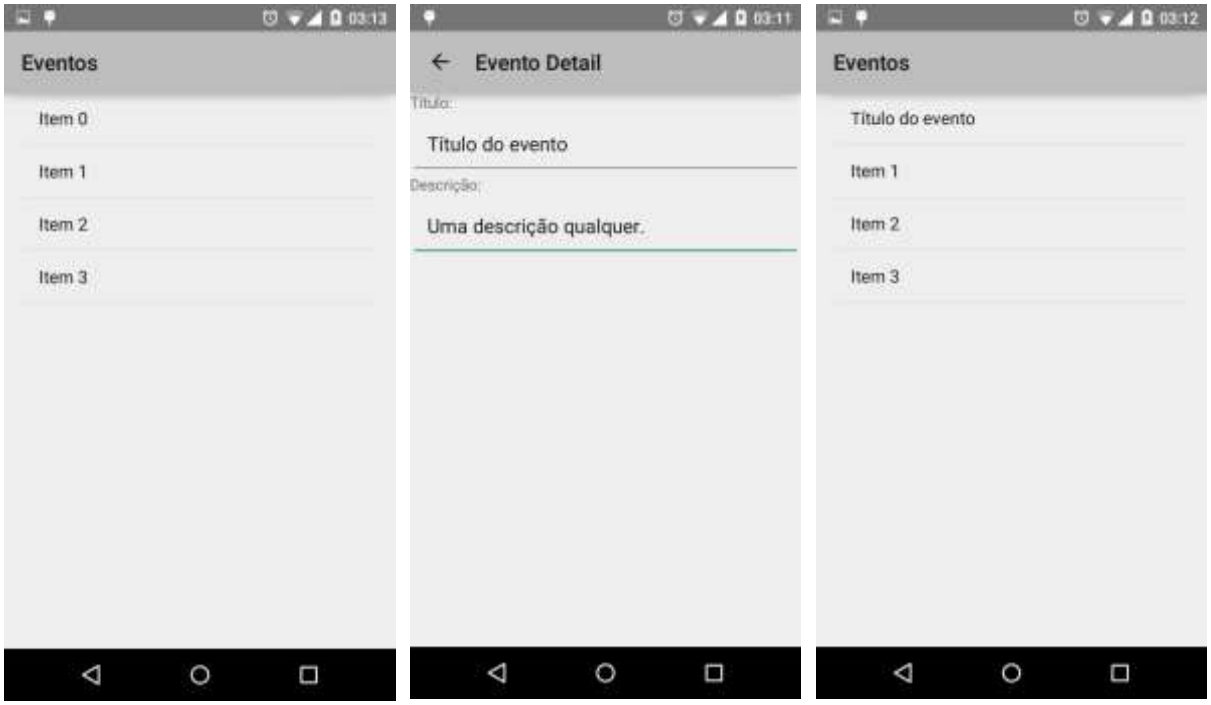
Depois, abra a classe **EventoListActivity** e faça as seguintes modificações:

EventoListActivity

```
~~public class EventoListActivity extends Activity~~
~~    implements EventoListFragment.Callbacks {~~
public class EventoListActivity extends Activity
      implements EventoListFragment.Callbacks, EventoDetailFragment.Callbacks {
…

   @Override
   public void onEventoUpdated(Evento evento) {
      EventoListFragment listFragment = ((EventoListFragment) getFragmentManager()
            .findFragmentById(R.id.evento_list));
      listFragment.updateUI();
   }
…
```

Aqui, pedimos o fragmento da lista ao FragmentManager e invocamos o método. Teste a aplicação. Agora devemos ter o seguinte:

Fig. 4.1.2 - Alterando e visualizando



Pronto, nossa aplicação já está preparada para o desenho de solução de armazenamento de dados.

Crie uma cópia desse projeto chamada EventosJSON.

Estratégia JSON

Como discutimos no vídeo do capítulo, a estratégia JSON envolverá duas etapas: salvar e carregar.

Para salvar, primeiro colocamos nossos dados no formato JSON, depois salvamos o arquivo. Para carregar, lemos o arquivo JSON e transformamos esses dados em variáveis.

Isso envolve alterar nosso modelo para essa estratégia. Abra a classe **Evento** e faça as seguintes alterações:

Evento

```
public class Evento {
    private static final String JSON_ID = "id";
    private static final String JSON_TITULO = "titulo";
    private static final String JSON_DESCRICAO = "descricao";

...

    public Evento(JSONObject json) throws JSONException {
        mId = UUID.fromString(json.getString(JSON_ID));
        mTitulo = json.getString(JSON_TITULO);
        mDescricao = json.getString(JSON_DESCRICAO);
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        json.put(JSON_ID, mId.toString());
        json.put(JSON_TITULO, mTitulo);
        json.put(JSON_DESCRICAO, mDescricao);
        return json;
    }

...
```

Tanto o construtor via um Objeto JSON, quando um método toJSON() para os dados foram adicionados ao modelo, resolvendo metade do problema. A outra metade do problema é lidar com os métodos I/O para salvar e carregar os arquivos.

Para isso, crie uma classe **EventosJSONSerializer** com a seguinte implementação:

Evento

Evento

```
public class EventosJSONSerializer {

   private Context mContext;
   private String mFilename;

   public EventosJSONSerializer (Context c, String f) {
      mContext = c;
      mFilename = f;
   }

   public void salvarEventos(ArrayList<Evento> eventos) throws JSONException,
IOException {
      // Cria o array JSON com os eventos
      JSONArray array = new JSONArray();
      for(Evento evento : eventos) {
         array.put(evento.toJSON());
      }

      Writer writer = null;
      try {
         // Abre um arquivo na sandbox com permissão privada
         OutputStream out = mContext.openFileOutput(mFilename, Context.MODE_PRIVATE);
         writer = new OutputStreamWriter(out);
         writer.write(array.toString());
      } finally {
         if (writer != null) {
            writer.close();
         }
      }
   }

...

  public ArrayList<Evento> carregarEventos() throws IOException, JSONException {
      ArrayList<Evento> eventos = new ArrayList<Evento>();
      BufferedReader reader = null;
```

Evento

```
        try {
            // Abre o arquivo
            InputStream in = mContext.openFileInput(mFilename);
            reader = new BufferedReader(new InputStreamReader(in));
            StringBuilder jsonString = new StringBuilder();
            String line = null;
            while ((line = reader.readLine()) != null) {
                jsonString.append(line);
            }
            // Parse JSON string
            JSONArray array = (JSONArray) new JSONTokener(jsonString.toString()).nextValue();
            //Popula o Array de eventos
            for(int i=0; i<array.length(); i++) {
                eventos.add(new Evento(array.getJSONObject(i)));
            }
        } catch (FileNotFoundException e) {
            // caso ele não encontre o arquivo, criamos alguns novos
            for(int i =0; i<4; i++)  {
                Evento evento = new Evento();
                evento.setTitulo("Item " + i);
                evento.setDescricao("Descrição" + i);
                eventos.add(evento);
        } finally {
            if (reader != null) {
                reader.close();
            }
        }
        return eventos;
    }
}
```

Essa implementação resolve a outra metade do problema: salvar e carregar os dados num arquivo JSON.

Como todas as etapas estão preparadas, basta apenas chamar os eventos. Quem ficará responsável por esses eventos será o gerenciador deles. Abra a classe **EventosManager** e adicione o seguinte:

EventosManager

```
public class EventosManager {

    private static final String FILENAME = "eventos.json";

    private ArrayList<Evento> mEventos;
    private CriminalIntentJSONSerializer mSerializer;

    private static EventosManager sEventos;
    private Context mAppContext;

    private EventosManager(Context appContext) {
        mAppContext = appContext;
        mEventos = new ArrayList<Evento>();

        mSerializer = new EventosJSONSerializer(mAppContext,FILENAME);
        try {
            // tentamos carregar o JSON
            mEventos = mSerializer.carregarEventos();
        } catch (Exception e) {
            // caso ele não exista, ou dê algum erro,
            // criamos nossos dados falsos
            mEventos = new ArrayList<Evento>();
            for(int i =0; i<4; i++)  {
                Evento evento = new Evento();
                evento.setTitulo("Item " + i);
                evento.setDescricao("Descrição" + i);
                mEventos.add(evento);
            }
        }

        for(int i =0; i<4; i++)  {
```

EventosManager

```
        Evento evento = new Evento();
        evento.setTitulo("Item " + i);
        mEventos.add(evento);
    }
  }

  public boolean salvarEventos(){
    try {
      mSerializer.salvarEventos(mEventos);
      return true;
    } catch (Exception e) {
      return false;
    }
  }

...
```

E, como discutimos no vídeo do capítulo, o método

public boolean salvarEventos()

Deverá ser chamado no evento onPause() do fragmento que faz a alteração. Portanto, abra a classe **EventoDetailFragment** e insira essa chamada:

EventoDetailFragment

```
...

  @Override
  public void onPause() {
    super.onPause();
    EventosManager.get(getActivity()).salvarEventos();
  }
```

EventoDetailFragment
...

Agora teste a aplicação, modifique alguns itens, destrua a aplicação e abra novamente. Pronto, você salvou os dados na *sandbox* da aplicação de forma bem simples, utilizando arquivos JSON.

GitHub

Você pode importar fazer um *check out* do projeto via GitHub:
https://github.com/qumaciel/igti_android_eventosJSON.git

Capítulo 5. Comunicação Remota e Background Tasks

Neste capítulo discutiremos sobre comunicação remota via HTTP. Para entendermos como essas requisições funcionam com o Android, criaremos uma aplicação chamada PhotoGallery, que irá buscar e visualizar as fotos mais recentes do Flickr.

PhotoGallery

Crie um projeto com as seguintes configurações:

- Application name: PhotoGallery
- Company Domain: android.igti.com.br
- Minimum SDK: API 21 Android 5.0 (Lollipop)
- Template: Blank Activity
- Activity Name: PhotoGalleryActivity
- Layout name: fragment_photo_gallery
- Title: PhotoGallery
- Menu resource name: menu_photo_gallery

Copie a classe **SingleFragmentActivity** e o layout **activity_fragment.xml** do nosso projeto Eventos, que são nosso template para aplicações simples. Agora, complete o template, abrindo a classe **PhotoGalleryActivity** e escrevendo a seguinte implementação:

PhotoGalleryActivity

```
public class PhotoGalleryActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return PhotoGalleryFragment.newInstance();
    }
```

PhotoGalleryActivity

```
}
```

Que faz referência ao fragmento que iremos criar em seguida. Primeiro, vamos preparar a visualização do fragmento. Abra o layout **fragment_photo_gallery.xml** e faça as seguintes alterações:

layout/fragment_photo_gallery.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        android:paddingBottom="@dimen/activity_vertical_margin"
        tools:context=".PhotoGalleryActivity">
    <TextView
      android:text="@string/hello_world"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
</RelativeLayout>

<GridView xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/gridView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:columnWidth="120dp"
        android:numColumns="auto_fit"
        android:stretchMode="columnWidth"
        tools:context=".PhotoGalleryActivity">
</GridView>
```

O **GridView** é um **AdapterView.** Ele funciona de forma muito parecida com o **ListView** que utilizamos na aplicação *Eventos*. Dividimos esse GridView em colunas de 120dp e distribuímos os elementos.

Crie uma classe chamada **PhotoGalleryFragment** e implemente da seguinte forma:

PhotoGalleryFragment

```
public class PhotoGalleryFragment extends Fragment {

   private GridView mGridView;

   public static PhotoGalleryFragment newInstance() {
      PhotoGalleryFragment fragment = new PhotoGalleryFragment();

      return fragment;
   }

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setRetainInstance(true);
   }

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
      View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

      mGridView = (GridView)v.findViewById(R.id.gridView);

      return v;
   }
}
```

Uma das vantagens de se utilizar fragmentos é que podemos contornar o problema da visualização ser destruída quando giramos o celular, isso é desejável no nosso caso, já que não gostaríamos de ter que carregar todas as fotos outra vez só porque o usuário girou o telefone. Isso é feito utilizando o método

setRetainInstance(true)

Quando a instância do fragmento é retida, todas as variáveis-membros do fragmento mantém seus valores. Agora criaremos a classe responsável por lidar com as requisições. Crie uma classe chamada **FlickrFetchr** com a seguinte implementação:

FlickrFetchr

```
public class FlickrFetchr {

    byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();

        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();

            if(connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                return null;
            }

            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while ((bytesRead = in.read(buffer)) > 0) {
                out.write(buffer,0,bytesRead);
            }
            out.close();
            return out.toByteArray();
        } finally {
            connection.disconnect();
```

FlickrFetchr

```
        }
    }

    public String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}
```

Começamos de maneira simples com nossa implementação, com dois métodos. O método getUrlBytes busca o dado cru da URL e a transforma num Array de bytes. Ele cria um objeto URL a partir de uma String e depois cria uma conexão apontando para essa URL.

Esse objeto HttpURLConnection representa uma conexão, mas não conecta de fato. Só irá conectar quando você informar como irá fazer isso, utilizando o método getInputStream(), ou getOutputStream(), dependendo se queremos fazer um download ou upload nessa URL.

Depois de tudo criado, chamamos o método read() repetidamente até que toda a informação seja processada. Quando os dados terminam, fechamos e colocamos esses dados num ByteArrayOutputStream.

Para que tudo isso seja possível, precisamos pedir permissão ao SO para utilizar a internet em nossa aplicação. Abra o manifest e adicione a seguinte permissão:

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.igti.android.photogallery" >

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
```

AndroidManifest.xml

```
        android:allowBackup="true"
...
```

Background Threads

Como discutimos no vídeo do capítulo, o Android não nos permite fazer requisições de rede na thread principal, precisamos colocá-la em segundo plano. O jeito mais fácil de fazer isso é com a classe **AsyncTask**: ela cria essa thread em segundo plano e executa o código no método **doInBackground**(…).

Faremos uma classe interna à **PhotoGalleryFragment** chamada **FetchItemsTask** que irá fazer a comunicação com o site.

PhotoGalleryFragment

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";
    private GridView mGridView;

…

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
        new FetchItemsTask().execute();
    }

    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
```

PhotoGalleryFragment

```
            String result = new FlickrFetchr().getUrl("https://www.google.com/");
            Log.i(TAG,"Conteúdo da URL: " + result);
        } catch (IOException ioe) {
            Log.e(TAG,"Não foi possível baixar o conteúdo",ioe);
        }
        return null;
    }
  }
}
```

Rode a aplicação e veja o conteúdo da página do Google no log.

Agora que nossa aplicação já consegue lidar com as requisições, vamos prepará-la para a comunicação com a API do Flickr. Veja a documentação da API em www.flickr.com/services/api. Nós utilizaremos um dos métodos GET dessa API, o método **flickr.photos.getRecent**, que retornará uma lista das últimas publicações feitas na plataforma. Esse método só tem um campo obrigatório, uma API key. Essas keys são muito utilizadas em Webservices públicos, e geralmente exigem um cadastro para que essa chave seja disponibilizada.

No link acima, procure por API keys e peça sua chave não comercial. Você precisará de uma conta Yahoo! para isso. Com a chave em mãos poderemos fazer a requisição. Abra a classe **FlickrFetchr**

FlickrFetchr

```
public class FlickrFetchr {
    private static final String TAG = "FlickrFetchr";

    private static final String ENDPOINT = "https://api.flickr.com/services/rest/";
    private static final String KEY = "sua_API_key_aqui";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
    private static final String PARAM_EXTRAS = "extras";
```

FlickrFetchr

```
   private static final String EXTRA_SMALL_URL = “url_s";

…

public void fetchItems() {
      try {
         String url = Uri.parse(ENDPOINT).buildUpon()
               .appendQueryParameter("method",METHOD_GET_RECENT)
               .appendQueryParameter("api_key",KEY)
               .appendQueryParameter(PARAM_EXTRAS,EXTRA_SMALL_URL)
               .build().toString();
         String xmlString = getUrl(url);
         Log.i(TAG, "Xml recebido: " + xmlString);
      } catch (IOException ioe) {
         Log.e(TAG,"Falhou: ",ioe);
      }
   }
}
```

Primeiro adicionamos algumas constantes referentes ao que faremos e queremos do Flickr. Pedimos um parâmetro extra, url_s, para que o Flickr nos dê a URL da versão pequena da imagem.

Depois utilizamos a Uri.Builder para montar a URL. É uma forma conveniente de montar URLs com parâmetros extras.

Agora, volte ao **PhotoGalleryFragment**

PhotoGalleryFragment

```
private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            String result = new FlickrFetchr().getUrl("https://www.google.com/");
            Log.i(TAG,"Conteúdo da URL: " + result);
        } catch (IOException ioe) {
            Log.e(TAG,"Não foi possível baixar o conteúdo",ioe);
        }
        new FlickrFetchr().fetchItems();
        return null;
    }
}
```

Rode a aplicação e veja o XML aparecer no log.

Observando esse XML, podemos criar o modelo para nossa aplicação agora. Crie uma classe chamada **GalleryItem** com a seguinte implementação:

GalleryItem

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;
    private String mOwner;

    public String getCaption() {
        return mCaption;
    }

    public void setCaption(String caption) {
        mCaption = caption;
    }
```

GalleryItem

```
    public String getId() {
        return mId;
    }

    public void setId(String id) {
        mId = id;
    }

    public String getUrl() {
        return mUrl;
    }

    public void setUrl(String url) {
        mUrl = url;
    }

    public String getOwner() {
        return mOwner;
    }

    public void setOwner(String owner) {
        mOwner = owner;
    }

    public String toString() {
        return  mCaption;
    }

    public String getPhotoPageUrl() {
        return "https://www.flickr.com/photos/"+mOwner+"/"+mId;
    }
}
```

Com o modelo feito, podemos preenchê-lo com os dados vindos do XML com a interface **XmlPullParser**. Abra a classe **FlickrFetchr**

FlickrFetchr

```
public class FlickrFetchr {

…
   private static final String XML_PHOTO = “photo";

…

   public void parseItems(ArrayList<GalleryItem> items, XmlPullParser parser) throws
XmlPullParserException, IOException {
      int eventType = parser.next();

      while (eventType != XmlPullParser.END_DOCUMENT) {
         if(eventType == XmlPullParser.START_TAG &&
XML_PHOTO.equals(parser.getName())) {
            String id = parser.getAttributeValue(null,"id");
            String caption = parser.getAttributeValue(null,"title");
            String smallUrl = parser.getAttributeValue(null,EXTRA_SMALL_URL);
            String owner = parser.getAttributeValue(null,"owner");

            GalleryItem item = new GalleryItem();
            item.setId(id);
            item.setCaption(caption);
            item.setUrl(smallUrl);
            item.setOwner(owner);
            items.add(item);
         }
         eventType = parser.next();
      }

   }

   public void fetchItems() {
      try {
         String url = Uri.parse(ENDPOINT).buildUpon()
               .appendQueryParameter("method", METHOD_GET_RECENT)
               .appendQueryParameter("api_key", KEY)
```

FlickrFetchr

```
            .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
            .build().toString();
        String xmlString = getUrl(url);
        Log.i(TAG, "Xml recebido: " + xmlString);
    } catch (IOException ioe) {
        Log.e(TAG,"Falhou: ",ioe);
    }
}
...

...
public ArrayList<GalleryItem> downloadGalleryItems(String url) {
    ArrayList<GalleryItem> items = new ArrayList<GalleryItem>();
    try {
        String xmlString = getUrl(url);
        Log.i(TAG,"Received xml: " + xmlString);

        XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
        XmlPullParser parser = factory.newPullParser();
        parser.setInput(new StringReader(xmlString));
        parseItems(items,parser);
    } catch (IOException ioe) {
        Log.e(TAG,"Failed to fetch items: ",ioe);
    } catch (XmlPullParserException xppe) {
        Log.e(TAG, "Failed to parse items", xppe);
    }
    return items;
}

public ArrayList<GalleryItem> fetchItems() {
    String url =
Uri.parse(ENDPOINT).buildUpon().appendQueryParameter("method",METHOD_GET_RECEN
T).appendQueryParameter("api_key",KEY).appendQueryParameter(PARAM_EXTRAS,EXTRA
_SMALL_URL).build().toString();
    return downloadGalleryItems(url);
}
```

Pense no XmlPullParser como se ele tivesse um dedo apontando para o XML, e ele vai andando por ele passo a passo, lendo o conteúdo. Para caminhar pelo XML, chamamos o método next().

Vamos fazer nosso GridView mostrar alguma coisa agora. Em **PhotoGalleryFragment**, faça o seguinte:

PhotoGalleryFragment

```
…
   private GridView mGridView;
   private ArrayList<GalleryItem> mItems;

…

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
      View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

      mGridView = (GridView)v.findViewById(R.id.gridView);
      setupAdapter();
      return v;
   }

   void setupAdapter() {
      if(getActivity() == null || mGridView ==null) {
         return;
      }

      if (mItems !=null) {
         mGridView.setAdapter(new
ArrayAdapter<GalleryItem>(getActivity(),android.R.layout.simple_gallery_item,mItems));
      } else {
         mGridView.setAdapter(null);
      }
   }
```

PhotoGalleryFragment

```
~~private class FetchItemsTask extends AsyncTask<Void,Void,Void> {~~
private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
    @Override
    ~~protected Void doInBackground(Void... params) {~~
    protected ArrayList<GalleryItem> doInBackground(Void... params) {
~~        new FlickrFetchr().fetchItems();~~
~~        return null;~~
        return new FlickrFetchr().fetchItems();

    }

    @Override
    protected void onPostExecute(ArrayList<GalleryItem> galleryItems) {
        mItems = galleryItems;
        setupAdapter();
    }
}
```

Como o **GridView** não tem uma classe **GridViewFragment**, teremos que implementar um adaptador. Utilizamos um layout padrão do Android com apenas um **TextView** e como nosso modelo sobrescreve o método **toString()** para mostrar o Caption da imagem, é o que aparecerá em nosso grid.

Depois, precisamos popular o grid quando as informações são baixadas. Utilizamos o método onPostExecute(…) que será disparado quando o método doInBackground(…) termina, *i.e.*, quando já fizemos o parse do XML, podemos atualizar a IU. Rodando a aplicação, veremos o seguinte:

Fig. 5.2.1 – PhotoGallery

Agora que o XML já foi baixado e transformado em variáveis do modelo, iremos mostrar as imagens.

Para que as imagens sejam mostradas no GridView, precisaremos de um novo adaptador. Começamos criando um novo layout chamado gallery_item.xml

layout/gallery_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/gallery_item_imageView"
        android:layout_width="match_parent"
        android:layout_height="120dp"
        android:layout_gravity="center"
        android:scaleType="centerCrop">
</ImageView>
```

Agora vamos criar nosso adaptador customizado. Em **PhotoGalleryFragment**, faça as seguintes alterações:

PhotoGalleryFragment

```
…
   void setupAdapter() {
      if(getActivity() == null || mGridView ==null) {
         return;
      }

      if (mItems !=null) {
         mGridView.setAdapter(new
ArrayAdapter<GalleryItem>(getActivity(),android.R.layout.simple_gallery_item,mItems));
         mGridView.setAdapter(new GalleryItemAdapter(mItems));
      } else {
         mGridView.setAdapter(null);
      }
   }
…

   private class GalleryItemAdapter extends ArrayAdapter<GalleryItem> {
      public GalleryItemAdapter (ArrayList<GalleryItem> items) {
         super(getActivity(),0,items);
      }

      @Override
      public View getView(int position, View convertView, ViewGroup parent) {
         if(convertView == null) {
            convertView =
getActivity().getLayoutInflater().inflate(R.layout.gallery_item,parent,false);
         }

         ImageView imageView =
(ImageView)convertView.findViewById(R.id.gallery_item_imageView);
         imageView.setImageResource(android.R.drawable.ic_menu_gallery);

         return convertView;
      }
```

PhotoGalleryFragment

```
  }
...
```

Rode a aplicação e veja que, quando o conteúdo carrega, já temos uma miniatura do Android de imagem, ao invés do TextView.

Agora precisaremos de outra estratégia para baixar as imagens. Se fizermos tudo do doInBackground teríamos que baixar todas as imagens de uma vez e depois visualizar. Sem contar que baixariam todas as 100 imagens da lista.

Queremos algo mais inteligente, baixar e mostrar a miniatura assim que ela estiver disponível e fazer isso somente naquelas que estão na área visível do dispositivo.

Para isso, ao invés da AsyncTask, utilizaremos threads em segundo plano dedicadas. Colocaremos algumas coisas na fila para serem baixadas.

Uma thread que utiliza essa fila, é chamada *message loop*. Esses *loop*s ficam constantemente procurando por novas mensagens na fila.

Nossa thread principal também tem um looper desses, e o que faremos é uma thread em segundo plano para lidar com isso. Crie uma nova classe chamada ThumbnailDownloader que será uma subclasse de HandlerThread.

ThumbnailDownloader

```
public class ThumbnailDownloader<Token> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    Handler mHandler;
    Map<Token,String> requestMap = Collections.synchronizedMap(new HashMap<Token,
String>());
    Handler mResponseHandler;
```

ThumbnailDownloader

```
  Listener mListener;

  public void setListener(Listener<Token> listener) {
     mListener = listener;
  }


  public ThumbnailDownloader(Handler responseHandler) {
     super(TAG);
     mResponseHandler = responseHandler;
  }

  @Override
  protected void onLooperPrepared() {
     mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
           if(msg.what == MESSAGE_DOWNLOAD) {
              Token token = (Token)msg.obj;
              Log.i(TAG,"Got a request for url: " + requestMap.get(token));
              handleRequest(token);
           }
        }
     };
  }

  public void queueThumbnail(Token token,String url) {
     Log.i(TAG, "Got an URL: " + url);
     requestMap.put(token,url);

     mHandler.obtainMessage(MESSAGE_DOWNLOAD,token).sendToTarget();
  }

 …
…
  private void handleRequest(final Token token) {
```

ThumbnailDownloader

```
        try {
            final String url = requestMap.get(token);
            if(url == null) {
                return;
            }

            byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
            final Bitmap bitmap =
BitmapFactory.decodeByteArray(bitmapBytes,0,bitmapBytes.length);
            Log.i(TAG,"Bitmap created");

            mResponseHandler.post(new Runnable() {
                @Override
                public void run() {
                    if(requestMap.get(token) != url) {
                        return;
                    }
                    requestMap.remove(token);
                    mListener.onThumbnailDownloaded(token,bitmap);
                }
            });
        } catch (IOException ioe) {
            Log.e(TAG,"Error downloading image",ioe);
        }
    }

    public void clearQueue() {
        mHandler.removeMessages(MESSAGE_DOWNLOAD);
        requestMap.clear();
    }

    public  interface Listener<Token> {
        void onThumbnailDownloaded(Token token, Bitmap thumbnail);
    }
}
```

Nossa classe identifica cada download utilizando um objeto genérico **Token**. Depois lidamos com a requisição de baixar e criar o bitmap da foto.

Agora vamos colocá-lo em **PhotoGalleryFragment**:

PhotoGalleryFragment

```
…
   private ArrayList<GalleryItem> mItems;
   private ThumbnailDownloader<ImageView> mThumbnailThread;
…

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setRetainInstance(true);
      setHasOptionsMenu(true);
      updateItems();

      mThumbnailThread = new ThumbnailDownloader<ImageView>(new
Handler());
      mThumbnailThread.setListener(new
ThumbnailDownloader.Listener<ImageView>() {
         @Override
         public void onThumbnailDownloaded(ImageView imageView, Bitmap
thumbnail) {
            if(isVisible()) {
               imageView.setImageBitmap(thumbnail);
            }
         }
      });
      mThumbnailThread.start();
```

PhotoGalleryFragment

```
        mThumbnailThread.getLooper();
        Log.i(TAG, "Background thread iniciada!");
    }
    public void updateItems() {
        new FetchItemsTask().execute();
    }
…

    @Override
    public void onDestroy() {
        super.onDestroy();
        mThumbnailThread.quit();
        Log.i(TAG, "Background thread destruída!");
    }

   @Override
    public void onDestroyView() {
        super.onDestroyView();
        mThumbnailThread.clearQueue();
    }
…
```

Lembre-se sempre de chamar o quit() na thread. Senão ela fica no modo zombie no SO. Um Handler funciona como uma espécie de carteiro no sistema operacional, ele lidará com as mensagens do Looper e também precisa ser liberado dessa obrigação utilizando o clearQueue();

Agora só falta atualizar o adaptador:

PhotoGalleryFragment

```
    private class GalleryItemAdapter extends ArrayAdapter<GalleryItem> {
        public GalleryItemAdapter (ArrayList<GalleryItem> items) {
            super(getActivity(),0,items);
        }

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            if(convertView == null) {
                convertView = getActivity().getLayoutInflater().inflate(R.layout.gallery_item,parent,false);
            }

            ImageView imageView =
(ImageView)convertView.findViewById(R.id.gallery_item_imageView);
            imageView.setImageResource(android.R.drawable.ic_menu_gallery);
            GalleryItem item = getItem(position);
            mThumbnailThread.queueThumbnail(imageView,item.getUrl());

            return convertView;
        }
    }
```

Rode sua aplicação e veja a galeria funcionando como deveria.

GitHub

Você pode importar fazer um *check out* do projeto via GitHub:
https://github.com/qumaciel/igti_android_photoGallery.git

Capítulo 6. GPS

Neste capítulo discutiremos sobre um recurso do dispositivo que podemos utilizar em nossas aplicações, mais precisamente o **GPS**.

GPS

Vamos agora à estratégia de posicionamento do dispositivo via GPS. Iremos criar uma aplicação chamada OCorredor, que irá acompanhar o trajeto do dispositivo. No próximo capítulo, enriqueceremos essa aplicação com a API do Google Maps para mostrar o trajeto num mapa.

Crie um novo projeto:

- Application name: OCorredor
- Company Domain: android.igti.com.br
- Minimum SDK: API 21 Android 5.0 (Lollipop)
- Template: Blank Activity
- Activity Name: OCorredorActivity
- Layout name: fragment_ocorredor
- Title: OCorredor
- Menu resource name: menu_ocorredor

Copie a classe **SingleFragmentActivity** e o layout **activity_fragment.xml** do nosso projeto Eventos, que é nosso template para aplicações simples. Agora, complete o template, abrindo a classe **OCorredorActivity** e escrevendo a seguinte implementação:

OCorredorActivity

```
public class OCorredorActivity extends SingleFragmentActivity {

   @Override
   protected Fragment createFragment() {
      return OCorredorFragment.newInstance();
   }

}
```

Que faz referência ao fragmento que iremos criar em seguida. Primeiro, vamos preparar a visualização do fragmento. Abra o layout **fragment_ocorredor.xml** e faça as seguintes alterações:

layout/fragment_ocorredor.xml

```
 <?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   >
   <TableRow>
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:gravity="right"
         android:paddingRight="5sp"
         android:text="@string/iniciada"
         />
      <TextView android:id="@+id/corrida_iniciadaTextView"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         />
   </TableRow>
   <TableRow>
```

layout/fragment_ocorredor.xml

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:paddingRight="5sp"
        android:text="@string/latitude"
        />
    <TextView android:id="@+id/corrida_latitudeTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />
  </TableRow>
  <TableRow>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:paddingRight="5sp"
        android:text="@string/longitude"
        />
    <TextView android:id="@+id/corrida_longitudeTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />
  </TableRow>
 <TableRow>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:paddingRight="5sp"
        android:text="@string/altitude"
        />
    <TextView android:id="@+id/corrida_altitudeTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

layout/fragment_ocorredor.xml

```
        />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="right"
            android:paddingRight="5sp"
            android:text="@string/tempo_decorrido"
            />
        <TextView android:id="@+id/corrida_duracaoTextView"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:textSize="25sp"
            />
    </TableRow>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_height="wrap_content"
        >
        <Button android:id="@+id/corrida_iniciarButton"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/inicia"
            />
        <Button android:id="@+id/corrida_pararButton"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/para"
            />
        <Button
            android:id="@+id/corrida_mapaButton"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
```

layout/fragment_ocorredor.xml

```
        android:layout_weight="1"
        android:text="@string/mapa"/>
    </LinearLayout>
</TableLayout>
```

Esse layout pede algumas strings que adicionaremos agora. Adicionaremos, também, as Strings para o próximo capítulo.

values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">OCorredor</string>
    <string name="action_settings">Settings</string>
    <string name="iniciada">Iniciada:</string>
    <string name="latitude">Latitude:</string>
    <string name="longitude">Longitude:</string>
    <string name="altitude">Altitude:</string>
    <string name="tempo_decorrido">Tempo Decorrido:</string>
    <string name="inicia">Iniciar</string>
    <string name="para">Parar</string>
    <string name="gps_habilitado">GPS Habilitado</string>
    <string name="gps_desabilitado">GPS Desabilitado</string>
    <string name="cell_text">Corrida em %1$s</string>
    <string name="nova_corrida">Nova Corrida</string>
    <string name="mapa">Mapa</string>
    <string name="corrida_inicia">Início da Corrida</string>
    <string name="corrida_iniciada_em_format">Corrida iniciada em %s</string>
    <string name="corrida_finaliza">Fim da Corrida</string>
    <string name="corrida_finalizada_em_format">Corrida finalizada em %s</string>

</resources>
```

Agora, crie uma classe chamada **OCorredorFragment** que amarrará essa visualização com a seguinte implementação:

OCorredorFragment

```
public class OCorredorFragment extends Fragment {

   private Button mIniciarButton, mPararButton, mMapaButton;
   private TextView mIniciadaTextView, mLatitudeTextView,
         mLongitudeTextView, mAltitudeTextView, mDuracaoTextView;


   public static OCorredorFragment newInstance() {
      OCorredorFragment fragment = new OCorredorFragment();
      return  fragment;
   }

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setRetainInstance(true);

   }

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                     Bundle savedInstanceState) {
      View view = inflater.inflate(R.layout.fragment_ocorredor, container, false);

      mIniciadaTextView = (TextView)view.findViewById(R.id.corrida_iniciadaTextView);
      mLatitudeTextView = (TextView)view.findViewById(R.id.corrida_latitudeTextView);
      mLongitudeTextView = (TextView)view.findViewById(R.id.corrida_longitudeTextView);
      mAltitudeTextView = (TextView)view.findViewById(R.id.corrida_altitudeTextView);
      mDuracaoTextView = (TextView)view.findViewById(R.id.corrida_duracaoTextView);

      mIniciarButton = (Button)view.findViewById(R.id.corrida_iniciarButton);

      mPararButton = (Button)view.findViewById(R.id.corrida_pararButton);
```

OCorredorFragment

```
        mMapaButton = (Button)view.findViewById(R.id.corrida_mapaButton);


        return view;
    }
}
```

Rode a aplicação e veja que temos uma interface simples para trabalhar.

Os dados de posição no Android são dados pelo **LocationManager**. Como discutido no vídeo do capítulo, esse serviço do sistema manda a informação de posição a todos os interessados de mais de uma maneira. Nós utilizaremos a **API PendingIntent**.

Utilizando **PendingIntent** é como se pedíssemos ao **LocationManager** que nos envie um tipo de intenção (objeto que comunica com o SO) no futuro. Assim, independente do estado de sua aplicação o LocationManager continuará enviando essas intenções, até que alguém peça para parar.

Para gerenciar os detalhes sobre a corrida, crie uma classe singleton chamada **CorridaManager** e implemente da seguinte forma:

CorridaManager

```
public class CorridaManager {
    private static final String TAG = "CorridaManager";

    public static final String ACTION_LOCATION =
"br.com.igti.android.ocorredor.ACTION_LOCATION";

    private static CorridaManager sCorridaManager;
    private Context mAppContext;
    private LocationManager mLocationManager;

    private  CorridaManager(Context appContext) {
        mAppContext = appContext;
        mLocationManager =
(LocationManager)mAppContext.getSystemService(Context.LOCATION_SERVICE);
    }

    public static CorridaManager get(Context c) {
        if(sCorridaManager == null) {
            sCorridaManager = new CorridaManager(c.getApplicationContext());
        }
        return sCorridaManager;
    }

    private PendingIntent getLocationPendingIntent(boolean shouldCreate) {
        Intent broadcast = new Intent(ACTION_LOCATION);
        int flags = shouldCreate ? 0 : PendingIntent.FLAG_NO_CREATE;
        return PendingIntent.getBroadcast(mAppContext,0,broadcast,flags);
    }

    public void startLocationUpdates() {
        String provider = LocationManager.GPS_PROVIDER;

        // inicia a requisição de Localização
        PendingIntent pi = getLocationPendingIntent(true);
        mLocationManager.requestLocationUpdates(provider,0,0,pi);
```

CorridaManager

```
    }

    public void stopLocationUpdates() {
        PendingIntent pi = getLocationPendingIntent(false);
        if(pi != null) {
            mLocationManager.removeUpdates(pi);
            pi.cancel();
        }
    }

    public boolean isTracking() {
        return getLocationPendingIntent(false) != null;
    }

}
```

Perceba que nosso gerenciador tem três métodos públicos, um para iniciar, um para parar e um para verificar se a posição está sendo monitorada. O método

requestLocationUpdates(…)

requer alguns parâmetros para o menor tempo de espera (em milisegundos) e menor distância (em metros) a ser coberta antes de enviar uma nova atualização.

Esses valores devem ser ajustados a fim de serem os maiores possíveis, tais que sua aplicação tenha um comportamento bom, pois, dependendo dos valores, utiliza uma grande quantidade de bateria do dispositivo.

No nosso caso, queremos a maior precisão possível e uma maior frequência de atualização.

O método getLocationPendingIntent(…) cria uma intenção a ser utilizada no broadcast quando a atualização de posição acontece. Utilizamos um nome personalizado para identificar o evento na aplicação e o argumento shouldCreate diz se devemos criar um novo PendingIntent no sistema, ou não. Se o PendingIntent já foi criado, não há necessidade de criá-lo novamente.

Agora que a posição será divulgada, devemos implementar como iremos recebê-la. Crie uma classe chamada **LocationReceiver** da seguinte forma:

LocationReceiver

```
public class LocationReceiver extends BroadcastReceiver {
   private static final String TAG = "LocationReceiver";

   @Override
   public void onReceive(Context context, Intent intent) {
      Location loc =
(Location)intent.getParcelableExtra(LocationManager.KEY_LOCATION_CHANGED);
      if (loc != null) {
         onLocationReceived(context,loc);
      }
      if (intent.hasExtra(LocationManager.KEY_PROVIDER_ENABLED)) {
         boolean enabled =
intent.getBooleanExtra(LocationManager.KEY_PROVIDER_ENABLED, false);
         onProviderEnabledChanged(enabled);
      }
   }

   protected void onLocationReceived(Context context, Location location) {
      Log.d(TAG,this + " recebeu a posição de " + location.getProvider() + ": " +
location.getLatitude() + ", " + location.getLongitude());
   }

   protected void onProviderEnabledChanged(boolean enabled) {
      Log.d(TAG,"Provider " + (enabled ? "habilitado" : "desabilitado"));
   }
```

LocationReceiver

```
}
```

Como podemos perceber, quando o evento onReceive(…) é disparado, o LocationManager empacota no Intent algumas informações extras, que são as chaves de interesse: no caso, **LocationManager.KEY_LOCATION_CHANGED**, para disparar o evento onLocationReceived(…) e as chaves para habilitar/desabilitar o serviço.

Para que tudo isso funcione, devemos pedir permissão para utilizar esses recursos. Altere seu manifest da seguinte forma:

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.igti.android.ocorredor" >

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <uses-feature android:required="true" android:name="android.location" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="br.com.igti.android.ocorredor.OCorredorActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

AndroidManifest.xml

```
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <receiver android:name=".LocationReceiver" android:exported="false">
        <intent-filter>
            <action android:name="br.com.igti.android.ocorredor.ACTION_LOCATION"/>
        </intent-filter>
    </receiver>
  </application>

</manifest>
```

Agora, temos toda a funcionalidade pronta, falta amarrar com a visualização. Abra a classe **OCorredorFragment**

OCorredorFragment

```
...
  private CorridaManager mCorridaManager;

  private View.OnClickListener mIniciarListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      mCorridaManager.startLocationUpdates();
      updateUI();
    }
  };

  private View.OnClickListener mPararListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      mCorridaManager.stopLocationUpdates();
      updateUI();
    }
  };

...
```

OCorredorFragment

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    mCorridaManager = mCorridaManager.get(getActivity());
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
     ...

    mIniciarButton = (Button)view.findViewById(R.id.corrida_iniciarButton);
    mIniciarButton.setOnClickListener(mIniciarListener);

    mPararButton = (Button)view.findViewById(R.id.corrida_pararButton);
    mPararButton.setOnClickListener(mPararListener);

    mMapaButton = (Button)view.findViewById(R.id.corrida_mapaButton);

    updateUI();
    return view;
}

private void updateUI() {
    boolean iniciada = mCorridaManager.isTracking();

    mIniciarButton.setEnabled(!iniciada);
    mPararButton.setEnabled(iniciada);
}
```

Agora teste a aplicação e veja a posição no Log. Perceba que até o GPS sincronizar a primeira vez pode demorar um pouco. Principalmente em ambientes fechados.

Agora vamos construir uma variável modelo para nossa aplicação, que nos auxiliará mais para frente, se quisermos fazer uma lista dessas corridas e etc. Crie uma classe chamada **Corrida** da seguinte forma:

Corrida

```
public class Corrida {
    private long mId;
    private Date mDataInicio;

    public Corrida() {
        mId = -1;
        mDataInicio = new Date();
    }

    public long getId() {
        return mId;
    }

    public void setId(long id) {
        mId = id;
    }

    public Date getDataInicio() {
        return mDataInicio;
    }

    public void setDataInicio(Date dataInicio) {
        mDataInicio = dataInicio;
    }

    public int getDurationSeconds(long endMillis) {
        return (int)((endMillis-mDataInicio.getTime()) / 1000);
    }

    public static String formatDuration(int durationSeconds) {
        int segundos = durationSeconds % 60;
```

Corrida

```
        int minutos = ((durationSeconds - segundos) / 60) % 60;
        int horas = (durationSeconds - (minutos * 60) - segundos) / 3600;
        return String.format("%02d:%02d:%02d",horas,minutos,segundos);
    }
}
```

Agora, vamos implementar o modelo no **OCorredorFragment**

OCorredorFragment

```
…
    private Corrida mCorrida;
    private Location mUltimaLocalizacao;

    private BroadcastReceiver mLocationReceiver = new LocationReceiver() {
        @Override
        protected void onLocationReceived(Context context, Location location) {
            if (!mCorridaManager.isTracking()) {
                return;
            }
            mUltimaLocalizacao = location;
            if (isVisible()) {
                updateUI();
            }
        }

        @Override
        protected void onProviderEnabledChanged(boolean enabled) {
            int toastText = enabled ? R.string.gps_habilitado : R.string.gps_desabilitado;
            Toast.makeText(getActivity(), toastText, Toast.LENGTH_LONG).show();
        }
    };

    private View.OnClickListener mIniciarListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
```

OCorredorFragment

```
            mCorridaManager.startLocationUpdates();
            mCorrida = new Corrida();
            updateUI();
        }
    };

...

    @Override
    public void onStart() {
        super.onStart();
        getActivity().registerReceiver(mLocationReceiver,new
IntentFilter(CorridaManager.ACTION_LOCATION));
    }

    @Override
    public void onStop() {
        getActivity().unregisterReceiver(mLocationReceiver);
        super.onStop();
    }

...
private void updateUI() {
        boolean iniciada = mCorridaManager.isTracking();

        mIniciarButton.setEnabled(!iniciada);
        mPararButton.setEnabled(iniciada);

        if (mCorrida != null) {
            mIniciadaTextView.setText(mCorrida.getDataInicio().toString());
        }

        int duracaoSegundos = 0;
        if (mCorrida != null && mUltimaLocalizacao != null) {
            duracaoSegundos = mCorrida.getDurationSeconds(mUltimaLocalizacao.getTime());
            mLatitudeTextView.setText(Double.toString(mUltimaLocalizacao.getLatitude()));
```

OCorredorFragment

```
            mLongitudeTextView.setText(Double.toString(mUltimaLocalizacao.getLongitude()));
            mAltitudeTextView.setText(Double.toString(mUltimaLocalizacao.getAltitude()));
            mDuracaoTextView.setText(Corrida.formataDuracao(duracaoSegundos));
            mIniciadaTextView.setEnabled(!iniciada);
        }
    }

...
```

Criamos um LocationReceiver anônimo para salvar a localização atual e atualizar na IU. Também registramos esse receiver nos eventos onStart() e onStop().

Pronto, temos uma aplicação simples que é capaz de receber a posição atual do dispositivo.

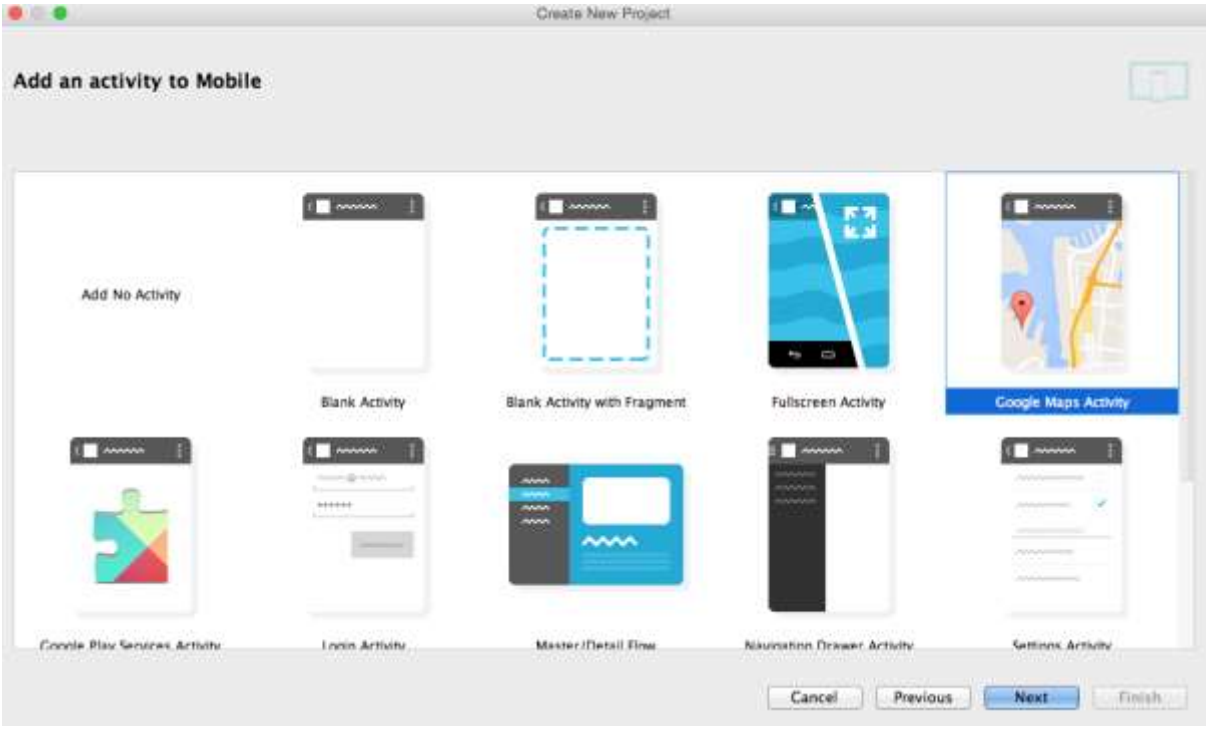
Google Play Services

Antes de incorporar o Google Maps, precisamos de algumas coisas. Da mesma forma que precisamos de uma API Key para utilizar o Flickr, também precisaremos de uma chave para utilizar a API do Google Maps.

Como discutimos no video do capítulo, esse processo em vias normais é bastante trabalhoso, mas temos um atalho para facilitar.

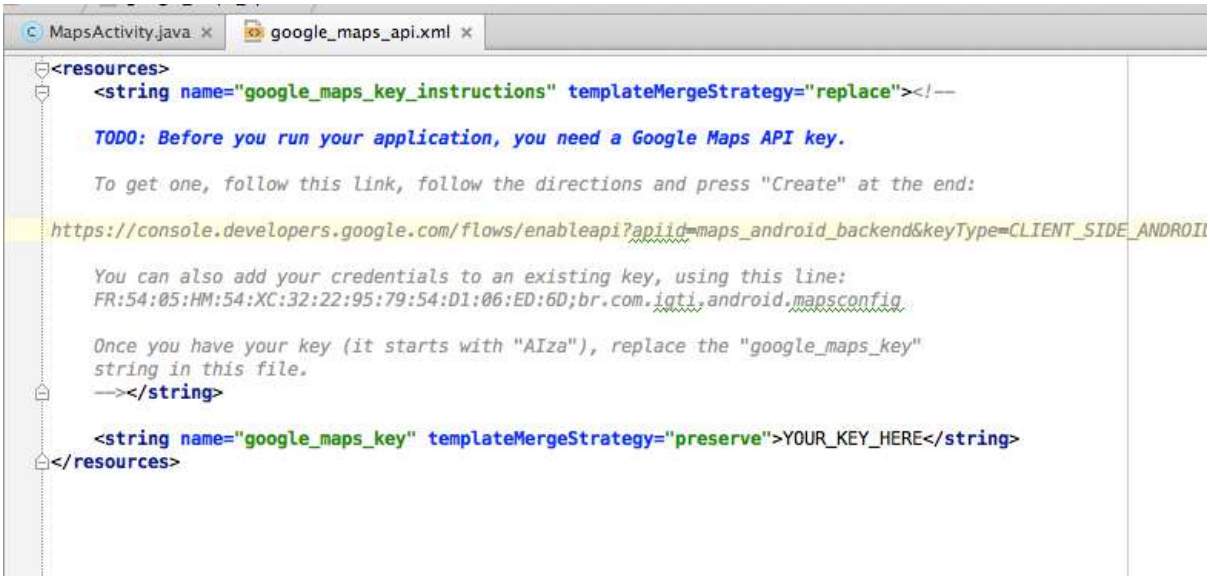
Crie um projeto, por exemplo, chamado MapsConfig e escolha o template **Google Maps Activity** como mostrado na figura abaixo:

Fig. 6.2.1 - Template do Google Maps Activity



Quando o projeto é criado, aparece a seguinte tela para você:

Fig. 6.2.2 - google_maps_api.xml



Copie e cole esse link destacado no navegador e siga as instruções. Na hora de cadastrar a aplicação ele pedirá algo do tipo:

FR:54:05:HM:54:XC:32:22:95:79:54:D1:06:ED:6D;br.com.igti.android.mapsconfig

Esse lugar é onde você registra as aplicações que poderão utilizar a API. Registre também nossa aplicação OCorredor, adicionando uma nova linha:

FR:54:05:HM:54:XC:32:22:95:79:54:D1:06:ED:6D;br.com.igti.android.mapsconfig
FR:54:05:HM:54:XC:32:22:95:79:54:D1:06:ED:6D;br.com.igti.android.ocorredor

Salve as alterações. Pronto, o google deve ter criado uma chave para você: uma String que começa com “AIza”. Guarde essa chave, feche esse projeto e abra o projeto **OCorredor**. Vamos incorporar o GoogleMaps a nossa aplicação agora.

Atualizando OCorredor

Ao final do capítulo, gostaríamos de mostrar o trajeto feito no mapa. Para isso, precisamos adicionar algumas coisas ao nosso projeto OCorredor. Abra a classe **CorridaManager** e adicione o seguinte:

CorridaManager

```
…
   private LocationManager mLocationManager;
   private ArrayList<Location> mTrajeto;

   private  CorridaManager(Context appContext) {
      mAppContext = appContext;
      mLocationManager =
(LocationManager)mAppContext.getSystemService(Context.LOCATION_SERVICE);
      mTrajeto = new ArrayList<Location>();
   }

…

   public void addLocationTrajeto (Location location) {
      mTrajeto.add(location);
```

CorridaManager

```
}

public ArrayList<Location> getTrajeto() {
    return mTrajeto;
}
```

E atualizamos, também, **OCorredorFragment**:

OCorredorFragment

```
…

private BroadcastReceiver mLocationReceiver = new LocationReceiver() {
        @Override
        protected void onLocationReceived(Context context, Location location) {
            if (!mCorridaManager.isTracking()) {
                return;
            }
            mUltimaLocalizacao = location;
            mCorridaManager.addLocationTrajeto(location);
            if (isVisible()) {
                updateUI();
            }
        }

        @Override
        protected void onProviderEnabledChanged(boolean enabled) {
            int toastText = enabled ? R.string.gps_habilitado : R.string.gps_desabilitado;
            Toast.makeText(getActivity(), toastText, Toast.LENGTH_LONG).show();
        }
    };

…
```

Agora, vamos criar uma atividade chamada **CorridaMapaActivity** com a seguinte implementação:

CorridaMapaActivity

```
package br.com.igti.android.ocorredor;

import android.Manifest;
import android.content.pm.PackageManager;
import android.content.res.Resources;
import android.location.Location;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentActivity;
import android.view.Display;

import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.LatLngBounds;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.PolylineOptions;

import java.util.ArrayList;
import java.util.Date;

public class CorridaMapaActivity extends FragmentActivity implements
OnMapReadyCallback {

   private GoogleMap mGoogleMap;
   private ArrayList<Location> mTrajeto;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_maps);
```

CorridaMapaActivity

```
        MapFragment fragment = (MapFragment)
getFragmentManager().findFragmentById(R.id.map);
        fragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap map) {
        mGoogleMap = map;

        // Coloca o mapa na posição atual do dispositivo
        if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            mGoogleMap.setMyLocationEnabled(true);
        }
        mGoogleMap.getUiSettings().setZoomControlsEnabled(true);

        // Pede a lista do trajeto da corrida
        mTrajeto = CorridaManager.get(this).getTrajeto();

        updateUI();
    }
```

CorridaMapaActivity

```
private void updateUI() {
    if (mGoogleMap == null || mTrajeto == null) {
        return;
    }

    PolylineOptions line = new PolylineOptions();
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder();
    for (Location loc : mTrajeto) {
        LatLng latLng = new LatLng(loc.getLatitude(), loc.getLongitude());
        Resources r = getResources();
        if (mTrajeto.indexOf(loc) == 0) {
            String startDate = new Date(loc.getTime()).toString();
            MarkerOptions starMarkerOptions = new MarkerOptions().position(latLng)
                    .title(r.getString(R.string.corrida_inicia))
                    .snippet(r.getString(R.string.corrida_iniciada_em_format, startDate));
            mGoogleMap.addMarker(starMarkerOptions);
        } else if (mTrajeto.indexOf(loc) == mTrajeto.size() - 1) {
            String endDate = new Date(loc.getTime()).toString();
            MarkerOptions finishMarkerOptions = new MarkerOptions().position(latLng)
                    .title(r.getString(R.string.corrida_finaliza))
                    .snippet(r.getString(R.string.corrida_finalizada_em_format, endDate));
            mGoogleMap.addMarker(finishMarkerOptions);
        }
        line.add(latLng);
        latLngBuilder.include(latLng);
    }
    mGoogleMap.addPolyline(line);
```

CorridaMapaActivity

```
        Display display = this.getWindowManager().getDefaultDisplay();
        LatLngBounds latLngBounds = latLngBuilder.build();
        CameraUpdate movement = CameraUpdateFactory.newLatLngBounds(latLngBounds,
display.getWidth(), display.getHeight(), 15);
        mGoogleMap.moveCamera(movement);
    }
}
```

Com referência ao layout que iremos criar agora, **activity_maps.xml**

res/layout/activity_maps.xml

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.MapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="br.com.igti.android.ocorredor.CorridaMapaActivity" />
```

Essa atividade traz primeiro uma instância do mapa do Google Maps, depois ele centraliza na posição atual do dispositivo com o método

mGoogleMap.setMyLocationEnabled(true),

depois, ele carrega a lista de localizações do CorridaManager.

Na interface com o usuário, desenhamos o trajeto utilizando a classe PolylineOptions, marcamos o início e o fim da corrida e centralizamos o trajeto no mapa.

Com tudo pronto, precisamos disparar essa atividade quando o usuário clicar no botão mapa em **OCorredorFragment**:

OCorredorFragment

```
...

   private View.OnClickListener mMapaListener = new View.OnClickListener() {
      @Override
      public void onClick(View v) {
         Intent i = new Intent(getActivity(), CorridaMapaActivity.class);
         startActivity(i);
      }
   };

...

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
      View view = inflater.inflate(R.layout.fragment_ocorredor, container, false);

   ...

      mMapaButton = (Button)view.findViewById(R.id.corrida_mapaButton);
      mMapaButton.setOnClickListener(mMapaListener);

   ...
    }
```

Para finalizar, adicionamos as permissões necessárias da API do Google Maps no manifest e registramos a atividade do mapa:

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.com.igti.android.ocorredor" >
```

AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-feature android:required="true" android:name="android.location" />
<uses-feature android:required="true" android:glEsVersion="0x00020000"/>

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="br.com.igti.android.ocorredor.OCorredorActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".CorridaMapaActivity"
            android:label="@string/app_name"/>
    <receiver android:name=".LocationReceiver" android:exported="false">
        <intent-filter>
            <action android:name="br.com.igti.android.ocorredor.ACTION_LOCATION"/>
        </intent-filter>
    </receiver>
    <meta-data
        android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
```

AndroidManifest.xml

```
    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="SUA_API_KEY_AQUI" />
  </application>
</manifest>
```

Onde você deve colocar sua chave na última tag meta-data.

GitHub

Você pode importar fazer um *check out* do projeto via GitHub:
https://github.com/qumaciel/igti_android_ocorredor.git

Capítulo 7. Câmera

Neste capítulo discutiremos sobre um recurso do dispositivo. Mais precisamente a **Câmera**.

Camera2

Falaremos primeiramente da API Camera2, introduzida no Android L. Ela veio para substituir a API antiga, nos dando maior controle das câmeras. Isso foi necessário pela própria evolução delas nos dispositivos atuais, gerando uma necessidade de algo mais elaborado.

Como esta API será implementada em todos os dispositivos Android a partir de agora, não falaremos da API antiga e só discutiremos essa, através do exemplo do próprio google de como utilizá-la.

Faremos isso pois é uma API muito nova e ainda tem alguns bugs quando implementamos em dispositivos mais antigos e emuladores. Por exemplo, até o presente momento em que escrevo essa apostila, essa API ainda não funciona nos emuladores, quando utilizamos a Webcam para simular a câmera. Ao que parece, há um problema de renderização ao lidar com câmeras tipo *legacy.*

Isso deverá ser resolvido num futuro próximo, por isso, analisaremos o exemplo do Google que terá sempre a implementação correta e atual da API, caso alguma modificação seja necessária em futuras implementações.

Faça uma cópia e importe o exemplo localizado em $sdk/samples/android-21/media/Camera2Basic/. Onde $sdk é o caminho em que seu SDK foi instalado. [**NOTA**: esse exemplo apareceu disponível recentemente na biblioteca do Android. Durante o processo de gravação do vídeo do capítulo ele ainda não estava

disponível e só tinhamos acesso ao exemplo do Android L. Ele agora é importado sem nenhum problema de compatibilidade.]

A utilização completa possível com a câmera também inclui dois outros exemplos: o Camer2Video, para gravações de video e o HdrViewfinder, para câmeras novas com alta taxa de atualização.

Depois de importado, vamos analisar os passos para utilização da câmera. Começamos pelo evento onResume(), nele, é chamado o método startBackgroundThread(), que irá iniciar uma thread em segundo plano e um Handler a ela (de forma similar como fizemos no capítulo passado para lidar com as miniaturas). Essa thread ficará responsável pela comunicação com a câmera.

Em seguida, iniciamos o processo de configuração da visualização da câmera. A idéia é ter um *preview* numa instância de **TextureView**.

Depois que esse mSurfaceTextureListener é adicionado, ele pode disparar o evento onSurfaceTextureAvailable(…), que é uma maneira de se certificar que o processamento desse preview da câmera só ocorrerá quando a visualização estiver disponível, e nesse método já sabemos também qual é o tamanho dessa visualização. Daí passamos esse parâmetro para tentar abrir a câmera no método openCamera(…).

A primeira coisa que temos nesse método é a chamada de um outro método, setUpCameraOutputs(…). Nesse método, pedimos uma instância de CameraManager e verificamos as configurações de todas as câmeras disponíveis, utilizando o manager.getCameraCharacteristics(cameraId).

O bloco que vem em seguida é uma sequência de comandos com a finalidade de verificar e utilizar a maior resolução possível dessa câmera. Essas resoluções, inicialmente, são acessadas via:

StreamConfigurationMap map = characteristics.get(

CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);

Feita a conta da melhor resolução, configuramos o tamanho da Texture view no aspectRatio correto utilizando o método setAspectRatio(…).

Isso é feito porque, caso a visualização não esteja configurada de forma compatível com a imagem que a câmera envia, isso disparará um erro na hora do preview (dimensões incompatíveis).

Voltando ao openCamera, temos, agora, a chamada do método configureTransform(…). Esse método configurará onde a textura que será processada irá aparecer dentro da TextureView. Falaremos mais sobre posicionamento de objetos renderizados no capítulo 10.

Finalmente, fazemos uma tentativa de abrir essa câmera junto ao CameraManager, utilizando o:

manager.openCamera(mCameraId, mStateCallback, mBackgroundHandler);

Esse método diz que iremos tentar abrir a câmera que foi configurada em setUpCameraOutputs(…), diz quem implementa os Callbacks e quem irá ser responsável pela comunicação. No caso, implementamos os Callbacks (onOpened(…), onDisconnected(…) e onError()) da câmera que tentaremos abrir no objeto mStateCallback e dizemos que a conversa com essa câmera será dada via uma thread em segundo plano com seu respectivo Handler e Looper que criamos no onResume().

Caso tenha sido possível abrir a câmera, o evento onOpened(…) é disparado e podemos tentar mostrar a visualização dela, chamando o método createCameraPreviewSession().

Nesse método, configuramos a superfície em que essa imagem é mostrada.

Imagine que o que é mostrado na tela é uma sequência quadro a quadro de fotos, e esse quadro é essa superfície, que deve ser do mesmo tamanho da imagem da câmera, claro. Daí configuramos via o método setDefaultBufferSize(…).

Em seguida, temos que dizer qual é a intenção de utilizar essa superfície, criando uma requisição de captura do tipo TEMPLATE_PREVIEW. I.e., dizemos a câmera que a intenção é ter essa imagem quadro a quadro mostrada naquela superfície para exibir um preview da câmera.

Com a requisição feita, podemos tentar iniciar uma sessão de captura. Essa sessão pede quais são as superfícies envolvidas, pede os Callbacks e um Handler (que, no caso, não utilizaremos por enquanto, já que definimos um Handler para a câmera).

Se for possível iniciar uma sessão de captura com essas superfícies configuradas, o evento onConfigured(…) é disparado e podemos processar aquele requisição tipo preview com o mPreviewRequestBuilder.build().

Como queremos algo que é atualizado quadro a quadro, precisamos pedir o esse quadro o tempo todo, daí temos o método

```
mCaptureSession.setRepeatingRequest(mPreviewRequest,
                                    mCaptureCallback, mBackgroundHandler),
```

que fará a captura repetidamente, disparando Callbacks e conversando via esse Handler em segundo plano.

No evento process(…) desse Callback é onde podemos processar a imagem crua em tempo real. Essa é uma das vantagens dessa nova API, que nos dá a liberdade de processar uma textura, por exemplo, na imagem capturada de forma *on the fly*!

Pronto! Essa foi uma resenha do processo — longo, porém muito melhor implementado — para um preview da câmera utilizando a nova API.

Entendido esse processo, o resto da utilização da câmera, seja para bater fotos ou gravar um video é muito semelhante. Para maiores informações, sugiro olhar também os exemplos Camer2Video e HdrViewfinder. Caso queiram utilizar a API antiga, vejam em http://developer.android.com/training/camera/ .

Capítulo 8. Vendas

Neste capítulo discutiremos sobre a monetização de um aplicativo Android.

Aplicativos Free vs. Pró

Temos, nessa estratégia, dois aplicativos: um simples e de graça e outro completo e pago. Essa estratégia de monetização é muito comum em diversos seguimentos de aplicativos e existem alguns prós e contras na utilização disso.

Essa estratégia pode facilitar a organização do projeto, já que existem duas linhas de desenvolvimento: a do aplicativo grátis e a do pago. A escolha do que será de graça e o que será pago é algo muito importante nessa solução. Lembramos que o usuário só irá investir no aplicativo Pró se o Free já resolve o problema dele e ele quer algo a mais. Se a expectativa não for atendida em um certo nível já no aplicativo gratuito, o usuário não irá comprar o Pró.

Podemos afirmar que o usuário Android, culturalmente, não é muito aderente a esse tipo de solução. Lembre-se que há alguns anos, levando em consideração a diversidade de dispositivos Android, os celulares/tablets não possuíam muita memória, além da possibilidade de serem dispositivos mais simples e baratos. Isso gera uma resistência dupla para a absorção desses aplicativos. O usuário não tem muito espaço para mais uma aplicação ou não está disposto a pagar caro pela versão Pró. Caso queira mais detalhes sobre a precificação de aplicativos na Google Play, veja [1].

Por isso, desencorajamos a utilização dessa estratégia.

Compras in-app

As compras *in-app* podem ser vistas incialmente como uma loja dentro da aplicação (*e.g.*, Google Play). O usuário escolhe e compra somente o que quer. Dessa maneira, essa estratégia já é bem estabelecida e os usuários de maneira geral já estão confortáveis em buscar itens dentro de uma loja móvel. Portanto, não entraremos muito em detalhes sobre essa utilização.

Para uma utilização moderna e adequada dessa solução, podemos extrapolar essa ideia e o problema da estratégia anterior. Por que não aumentar os recursos do seu aplicativo na medida em que ele necessita?

Perceba que essa estratégia é muito delicada, pois, dada uma certa quantidade de recursos grátis, você terá que convidar o usuário a expandir as funcionalidades.

Dependendo do tipo de convite, por exemplo, propagandas o tempo inteiro na tela, isso pode gerar um certo desconforto na experiência do usuário e ele desistir de utilizar o aplicativo: seja por falta de recursos nativamente disponíveis ou pelo excesso de propagandas.

Devemos considerar, também, que essa estratégia aumenta a complexidade da aplicação. Já que ela se torna uma espécie de quebra cabeças, dependendo de quais recursos extras o usuário comprou.

Um exemplo de bons aplicativos seriam jogos dos quais existe a possibilidade de progressão *lenta* caso o usuário não deseje comprar nada, mas tenha a opção de progressão *mais rápida* caso ele pague. Isso não deve interferir na experiência do jogo a fim de ser impossível progredir caso o usuário não pague por isso.

Por outro lado, ou exemplo ruim seria oferecer uma expansão de recursos *in-app* num aplicativo Pró, por exemplo. O usuário já pagou para sair da aplicação Free e utilizar a Pró, ele não quer comprar mais nada para resolver o problema dele.

Encorajamos esse tipo de solução, mas chamamos a atenção para a sensibilidade do convite feito ao usuário para expandir seus recursos, ou comprar o plano Pró. O principal é que seu aplicativo cumpra a expectativa de resolver o problema do usuário num certo nível. Essa compra tem que ser um recurso extra.

Como essa estratégia funciona na prática? Você pode utilizar o Play in-app Billing (cf. [2]) para vender itens e recursos adicionais ou para remover anúncios. Com a vantagem da conversão automática para preços locais, com opções para arredondar para padrões de preços locais ou definir preços locais por si mesmo, modelos de preços, códigos de promoção, e a In-app Billing Sandbox no app, com a capacidade de vender bens virtuais duráveis e consumíveis.

Você também pode utilizar a Play in-app Billing para oferecer aos usuários acesso contínuo a conteúdo ou serviços por uma taxa recorrente (e.g., Netflix). Use recursos como frequências de faturamento flexíveis, avaliações gratuitas, precificação introdutória e local, períodos de carência de pagamento, atualizações e downgrades, análises de conversão e relatórios e painéis de cobrança.

Anúncios

Você pode ganhar dinheiro por exibir anúncios da AdMob, incluindo recursos como anúncios nativos que permitem que você correlacione anúncios à aparência do seu aplicativo, o que pode ser uma boa ideia.

Os anúncios são uma maneira fácil e eficaz de gerar receita com seus aplicativos. A AdMob é uma plataforma de monetização inteligente para aplicativos que ajuda você a maximizar a receita de anúncios e compras no aplicativo. Mais de 1 milhão

de aplicativos usam a AdMob para gerar um fluxo de receita confiável. Alguns aplicativos são difíceis de monetizar, o que faz esssa estratégia interessante em alguns casos.

Tudo o que você precisa fazer é se inscrever na AdMob (cf. [3]) e usar o SDK dos anúncios para dispositivos móveis do Google para colocar anúncios em seu aplicativo com apenas algumas linhas de código. Você é pago rapidamente sem taxas cobradas pela AdMob. Além disso, a integração da AdMob com o Google Play Services atualiza as melhorias de desempenho automaticamente para os aplicativos sem alterações adicionais no SDK (diga “oi” para seu mais novo backdoor!).

Capítulo 9. Privacidade

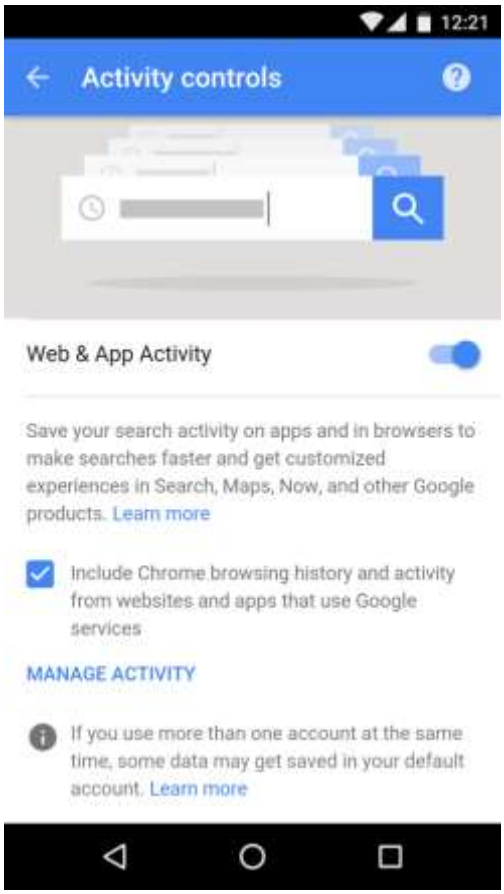
Discutiremos neste capítulo um assunto de muita importância: a privacidade.

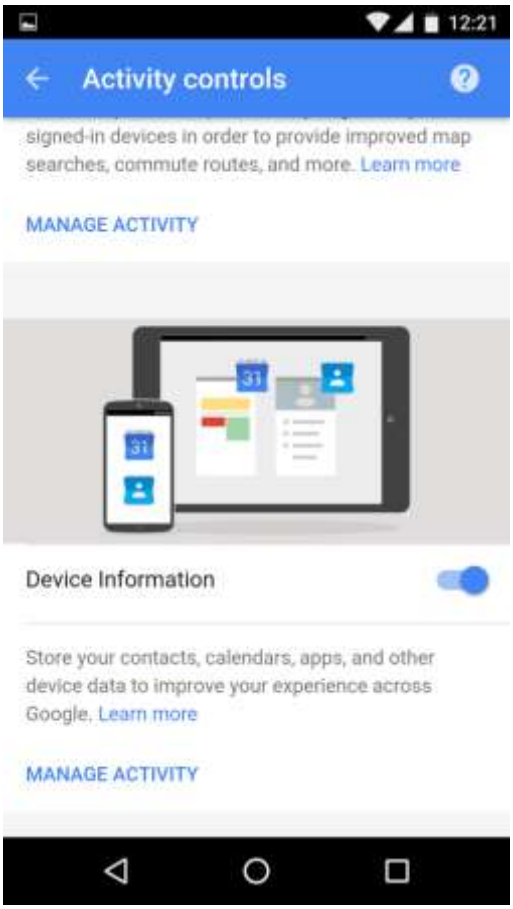
Rainer Kuhlen define privacidade não apenas como proteção de dados ou como o direito de ser deixado em paz, mas também como autonomia informacional [11], *i.e.*, a capacidade de escolher e utilizar o conhecimento e a informação autonomamente, em um ambiente eletrônico, e de determinar quais atributos de si serão usados por outros.

Note o quão delicado esse "determinar quais atributos de si serão usados por outros" pode ser.

Há alguns anos (pré-Snowden), a maioria dos termos de privacidade digitais eram encontrados no momento do cadastro, com aquele check-box "Li e aceito os termos". Culturalmente, ninguém lia os termos e acabávamos por aceitar tudo o que estivesse lá. Isso abre brecha para inúmeras indiscrições por meio desses coletores de informações. Nós desencorajamos esse tipo de estratégia.

Hoje em dia, a maneira mais adequada de se coletar dados é dar o controle para o usuário, *e.g.*, o Google Activity (https://myactivity.google.com):





Perceba a diferença dessa estratégia. Antes, essa coleta era prevista num termo que ninguém lia e ninguém poderia fazer nada. Hoje, temos explicitamente o que será coletado e a possibilidade de controlar a coleta.

Tenho que incluir uma política de privacidade no meu aplicativo?

Isso depende do que seu aplicativo está fazendo. Considere que você está sempre do lado mais seguro, incluindo um link ou uma visualização de página inteira da sua política de privacidade.

É muito provável que você seja obrigado por lei a incluir uma política de privacidade em seu aplicativo Android. Em 2014, a legislação brasileira ganhou uma regulação específica, o Marco Civil da Internet (cf. [13]).

Especificamente em relação à privacidade dos usuários, a lei determina especificamente a proteção da privacidade, dos dados pessoais e da segurança das redes.

Além disso, há previsão específica de responsabilidade civil dos agentes de acordo com suas atividades e obrigações, i.e., provedores de aplicação devem manter o sigilo das comunicações trocadas entre usuários e de seus dados pessoais.

Eles também devem guardar essas informações por um prazo mínimo de seis meses, caso sejam demandados judicialmente a fornecer essas informações às autoridades, mediante ordem judicial. Sem uma ordem judicial, dados pessoais, comunicações e padrões de navegação dos usuários não podem ser publicados. Vale a pena mencionar também o conteúdo do artigo 7º do Marco Civil, que garante aos usuários o direito de ter acesso a “informações claras e completas sobre coleta, uso, armazenamento, tratamento e proteção de seus dados pessoais”.

Além disso, essas informações somente poderão ser utilizadas a fim que “justifiquem sua coleta, que não sejam vedadas pela legislação e que estejam especificadas nos contratos de prestação de serviços ou em termos de uso de aplicações de internet”.

Ou seja, a política de privacidade é realmente o contrato que regula a relação entre usuário e provedor.

Tenho que incluir minha política de privacidade na Google Play?

Você pode até se safar de não postar uma política de privacidade, mas não se engane, isso não significa que não seja necessário em sua situação. Se você usar permissões sensíveis, como câmera, contatos, áudio, contas e estado do telefone, você receberá um e-mail do Google. Pode ter certeza!

Desde fevereiro de 2017, o Google aplica uma regra rigorosa de política de privacidade em aplicativos que solicitam permissões confidenciais e dados do usuário.

Existem alguns lugares na documentação do Google Play Store que indicam esse requisito. Se você quiser ler as declarações do Google em sua documentação e termos, você pode encontrá-los seguindo os links ou lendo os trechos mostrados (cf. [14], [15], [16] e [17]).

Como criar uma política de privacidade do zero?

Recomendo, inicialmente, que utilizem o gerador de políticas da Iubenda (cf. [18]). Lá você consegue gerar seu termo com apenas alguns cliques.

Como adicionar e editar essa política de privacidade na Play Store?

Fig. 9.1.1 – Google Play Develper Console

- Faça login no seu Console do desenvolvedor do Google Play (em [18]);
- Em seguida, selecione Todos os aplicativos e selecione o aplicativo cuja política de privacidade você deseja editar;
- Depois disso, selecione Listagem de loja;
- Em seguida, vá até a seção Política de Privacidade e insira o URL em que você tem a política de privacidade hospedada on-line – gere sua política de privacidade aqui;

- Por fim, não se esqueça de clicar em Salvar ou atualizar.

Capítulo 10. Aplicativos em 3D

Neste capítulo faremos uma introdução ao OpenGL ES e algumas visualizações em 2D e 3D.

Configurando o Renderizador

Para iniciarmos o estudo do OpenGL ES, crie a seguinte aplicação:

- Application name: OGLESBasic
- Company Domain: android.igti.com.br
- Minimum SDK: API 21 Android 5.0 (Lollipop)
- Template: Blank Activity
- Activity Name: OGLESActivity
- Layout name: activity_evento_list
- Title: OGLESBasic
- Menu resource name: menu_oglesbasic

Dê a seguinte implementação à classe **OGLESBasicActivity**:

OGLESBasicActivity

```
public class OGLESBasicActivity extends Activity {

   private GLSurfaceView mGLSurfaceView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      mGLSurfaceView = new GLSurfaceView(this);
      mGLSurfaceView.setRenderer(new GLRendererEx());

      setContentView(mGLSurfaceView);
   }
```

OGLESBasicActivity

```
    @Override
    protected void onPause() {
        super.onPause();
        mGLSurfaceView.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
        mGLSurfaceView.onResume();
    }
}
```

Estamos interessados em uma aplicação renderizada via OpenGL, portanto nota-se que não iremos inflar nenhum layout XML, quem será responsável pela visualização será o objeto **GLSurfaceView** e delegamos a renderização à classe **GLRendererEx**, que criaremos logo em seguida.

Como esse processo de renderização é custoso, tanto em termos de memória, quanto em termos de processamento, devemos nos certificar de que essa renderização é devidamente pausada e continuada nos eventos onPause() e onResume().

Agora vamos criar um renderizador para o nosso GLSurfaceView. Crie uma classe chamada **GLRendererEx** da seguinte forma:

GLRendererEx

```
public class GLRendererEx implements GLSurfaceView.Renderer {

    public GLRendererEx() {
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glDisable(GL10.GL_DITHER);
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
        gl.glClearColor(.5f,.8f,.5f,1);
        gl.glClearDepthf(1f);


    }

    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        gl.glViewport(0,0,width,height);
        float ratio = (float) width/height;
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        gl.glFrustumf(-ratio,ratio,-1,1,1,25);

    }

    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glDisable(GL10.GL_DITHER);
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    }
}
```

Essa classe implementa três métodos da interface **GLSurfaceView.Renderer**: onSurfaceCreated(…), onSurfaceChanged(…) e onDrawFrame(…).

No método onSurfaceCreated(…) é de boa prática, colocar as configurações padrões da superfície.

Quando iniciamos essa renderização, existem algumas opções que vêm por *default*, e as nossas duas primeiras linhas são para modificar algumas delas: desabilitar o dither e baixar a qualidade da correção de perspectiva, deixando a renderização mais rápida, mas com menor qualidade. Coloquei isso ali para que vocês se lembrem que sempre é possível fazer isso.

Os outros dois comandos são definições de cor e profundidade quando a gente limpa os buffers no evento onDrawFrame, lembrando que quando falamos em cor, geralmente, é do formato RGBa, com o A sendo a transparência.

No método onSurfaceChanged(…) é onde configuramos a área útil de visualização. O método GL10.glViewport(…) é o que define a área útil na tela. Depois definimos o sistema de coordenadas da visualização padrão, centralizado no meio da tela, sem nenhuma rotação.

Depois, definimos o Frustum. O frustum é a área virtual que iremos renderizar, por exemplo, se quisermos renderizar um quarto, o frustum seria dado pelas coordenadas das paredes, onde começa e termina uma espécie de caixa.

Finalmente, chegamos no onDrawFrame(…), que é onde a diversão acontece. Nesse momento, a única coisa que fizemos foi garantir que o dither e a correção de perspectiva está nos valores adequados e depois limpamos os buffers, deixando a tela com seu plano de fundo, no caso, verde.

Gráficos 2D

Como explicamos no video do capítulo, vamos utilizar o OpenGL primeiro num cenário 2D, por ser mais simples, desenhando um triângulo na tela.

Crie uma classe chamada **GLTrianguloEx** e implemente da seguinte forma:

GLTrianguloEx

```
public class GLTrianguloEx {
   private float mVertices[] = {
         0f, 1f, // ponto 0
         1f, -1f, // ponto 1
         -1f, -1f // ponto 1
   };

   private FloatBuffer mVerticesBuffer;

   private short[] mIndicesVertices = { 0, 1, 2};

   private ShortBuffer mIndicesBuffer;

   public GLTrianguloEx() {
      ByteBuffer verticesByteBuffer = ByteBuffer.allocateDirect(mVertices.length * 4);
      verticesByteBuffer.order(ByteOrder.nativeOrder());
      mVerticesBuffer = verticesByteBuffer.asFloatBuffer();
      mVerticesBuffer.put(mVertices);
      mVerticesBuffer.position(0);

      ByteBuffer indiceByteBuffer = ByteBuffer.allocateDirect(mIndicesVertices.length * 2);
      indiceByteBuffer.order(ByteOrder.nativeOrder());
      mIndicesBuffer = indiceByteBuffer.asShortBuffer();
      mIndicesBuffer.put(mIndicesVertices);
      mIndicesBuffer.position(0);
   }
```

GLTrianguloEx

```
    public void draw (GL10 gl) {
        // Vamos escolher os pontos no sentido horário
        gl.glFrontFace(GL10.GL_CW);
        // Habilita a sessão pra ligar os pontos via vertices
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        // Dá o pointer dos vértices e diz que temos um ponto
        // para cada duas coordenadas
        gl.glVertexPointer(2,GL10.GL_FLOAT,0,mVerticesBuffer);
        // Desenha, cobrindo a superfice por triângulos
gl.glDrawElements(GL10.GL_TRIANGLES,mIndicesVertices.length,GL10.GL_UNSIGNED_SH
ORT,mIndicesBuffer);
        // fecha os pontos
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    }
}
```

Nessa classe, primeiro definimos as coordenadas dos pontos em mVertices[]. No construtor, alocamos memória para essas coordenadas e para os vértices que associaremos a elas.

O Buffer para coordenadas tipo float consome 4 bytes por coordenada, e coordenadas tipo short custam 2 bytes.

Um comentário sobre o método verticesByteBuffer.order(ByteOrder.nativeOrder()), que é necessário para que o garbage collector não “coma" essa região alocada por acidente.

No método draw(…) temos o desenho de fato. Primeiro dizemos que faremos uma cobertura de pontos no sentido horário, depois abrimos a sessão de posicionar vértices, depois damos o pointer, dizendo a região de memória e desenhamos. Com o objeto desenhado, fechamos a sessão.

Veja o vídeo do capítulo com mais informações sobre esse fluxo de desenho.

Agora, vamos adicionar esse triângulo a visualização, na classe **GLRendererEx**:

GLRendererEx

```
public class GLRendererEx implements GLSurfaceView.Renderer {

    private GLTrianguloEx mTrianguloEx;

    public GLRendererEx() {
        mTrianguloEx = new GLTrianguloEx();
    }

…

    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glDisable(GL10.GL_DITHER);
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        GLU.gluLookAt(gl,0,0,-5,0,0,0,0,2,0);

        mTrianguloEx.draw(gl);
    }
}
```

Posicionamos uma câmera com o método

GLU.gluLookAt(GL10, eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ),

que recebe onde posicionamos a câmera, para onde ela olha e o vetor UP dessa câmera. No nosso caso, botamos uma câmera posicionada atrás da visualização,

i.e., em cima do plano do celular, e olhando para o meio da tela do celular. Depois, desenhamos o triângulo.

Rode a aplicação e você terá a seguinte tela:

Fig. 10.2.1 - Triângulo desenhado com OpenGL ES

Com o triângulo na tela, podemos colorir também. Abra a classe **GLTrianguloEx** e adicione o seguinte:

GLTrianguloEx

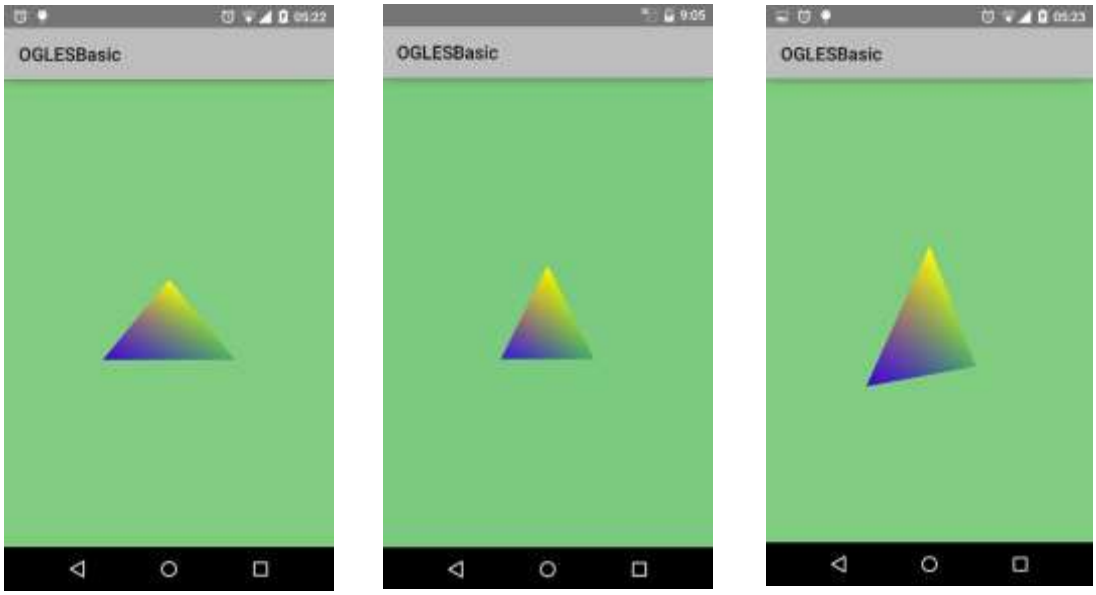
```
…

   private float mRgba[] = {
         1, 1, 0, .5f, // ponto 0
         .25f, 0, .85f, 1, // ponto 1
         .3f, .7f, .4f, 1 // ponto 1
   };

   private FloatBuffer mColorBuffer;

   public GLTrianguloEx() {
```

GLTrianguloEx

```
…

        ByteBuffer colorByteBuffer = ByteBuffer.allocateDirect(mRgba.length * 4);
        colorByteBuffer.order(ByteOrder.nativeOrder());
        mColorBuffer = colorByteBuffer.asFloatBuffer();
        mColorBuffer.put(mRgba);
        mColorBuffer.position(0);
    }

    public void draw (GL10 gl) {
        // Vamos escolher os pontos no sentido horário
        gl.glFrontFace(GL10.GL_CW);
        // Habilita a sessão pra ligar os pontos via vertices
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        // Habilita para colorir
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        // Dá o pointer dos vértices e diz que temos um ponto
        // para cada duas coordenadas
        gl.glVertexPointer(2,GL10.GL_FLOAT,0,mVerticesBuffer);
        // Dá o pointer dos vértices  para cor e diz que temos um ponto
        // para cada quatro coordenadas
        gl.glColorPointer(4,GL10.GL_FLOAT,0,mColorBuffer);
        // Desenha, cobrindo a superfice por triângulos

gl.glDrawElements(GL10.GL_TRIANGLES,mIndicesVertices.length,GL10.GL_UNSIGNED_SHORT
,mIndicesBuffer);
        // fecha a coloração
        gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
        // fecha os pontos
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    }
}
```

Nossa aplicação fica desse jeito:

Fig. 10.2.2 - Triângulo colorido em diferentes perspectivas



Agora que nosso triângulo está pronto, sugiro a vocês que brinquem um pouco com o posicionamento da câmera.

Gráficos 3D

Vamos criar um cubo agora. Crie uma classe chamada **GLCuboEx**, o cubo agora terá vertices com coordenadas (x,y,z) e será implementada da seguinte forma:

GLCuboEx

```
public class GLCuboEx {
    private float mVertices[] = {
            1, 1,-1, // ponto 0
            1,-1,-1, // ponto 1
           -1,-1,-1, // ponto 2
           -1, 1,-1, // ponto 3
            1, 1, 1, // ponto 4
```

GLCuboEx

```
        1,-1, 1, // ponto 5
       -1,-1, 1, // ponto 6
       -1, 1, 1, // ponto 7

  };

  private FloatBuffer mVerticesBuffer;

  // Um cubo é composto de 12 triângulos
  private short[] mIndicesVertices = {
                              3,4,0,  0,4,1,  3,0,1,
                              3,7,4,  7,6,4,  7,3,6,
                              3,1,2,  1,6,2,  6,3,2,
                              1,4,5,  5,6,1,  6,5,4};

  private ShortBuffer mIndicesBuffer;

  public GLCuboEx() {
     ByteBuffer verticesByteBuffer = ByteBuffer.allocateDirect(mVertices.length * 4);
     verticesByteBuffer.order(ByteOrder.nativeOrder());
     mVerticesBuffer = verticesByteBuffer.asFloatBuffer();
     mVerticesBuffer.put(mVertices);
     mVerticesBuffer.position(0);

     ByteBuffer indiceByteBuffer = ByteBuffer.allocateDirect(mIndicesVertices.length * 2);
     indiceByteBuffer.order(ByteOrder.nativeOrder());
     mIndicesBuffer = indiceByteBuffer.asShortBuffer();
     mIndicesBuffer.put(mIndicesVertices);
     mIndicesBuffer.position(0);
  }
...
```

```
...
public void draw (GL10 gl) {
        // Vamos escolher os pontos no sentido horário
        gl.glFrontFace(GL10.GL_CW);
        // Não renderizar a face de trás dos triângulos
        gl.glEnable(GL10.GL_CULL_FACE);
        gl.glCullFace(GL10.GL_BACK);
        // Habilita a sessão pra ligar os pontos via vertices
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        // Dá o pointer dos vértices e diz que temos um ponto
        // para cada três coordenadas
        gl.glVertexPointer(3,GL10.GL_FLOAT,0,mVerticesBuffer);
        // Desenha, cobrindo a superfice por triângulos

gl.glDrawElements(GL10.GL_TRIANGLES,mIndicesVertices.length,GL10.GL_
UNSIGNED_SHORT,mIndicesBuffer);
        // fecha os pontos
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        // Desfaz o cull_face
        gl.glDisable(GL10.GL_CULL_FACE);
    }
}
```

Quando ganhamos uma dimensão vemos uma diferença nos índices dos vértices. Queremos desenhar um cubo, mas como utilizamos

gl.glFrontFace(GL10.GL_CW) e gl.glDrawElements(GL10.GL_TRIANGLES….)

Temos que desenhá-lo utilizando triângulos e cobrindo no sentido horário. Um cubo pode ser dividido em quatro tetraedros e, consequentemente, 12 triângulos. Rotulamos eles corretamente para que, no fim, tenhamos um cubo preenchido.

A outra diferença aqui está em

gl.glEnable(GL10.GL_CULL_FACE) e gl.glCullFace(GL10.GL_BACK),

que dizem ao renderizador para que não renderize a face de dentro do cubo, já que não visualizaremos a face de dentro, podemos ignorá-la e economizar recursos.

Agora vamos aplicar algumas rotações nesse cubo. Abra a classe **GLRendererEx**:

GLRendererEx

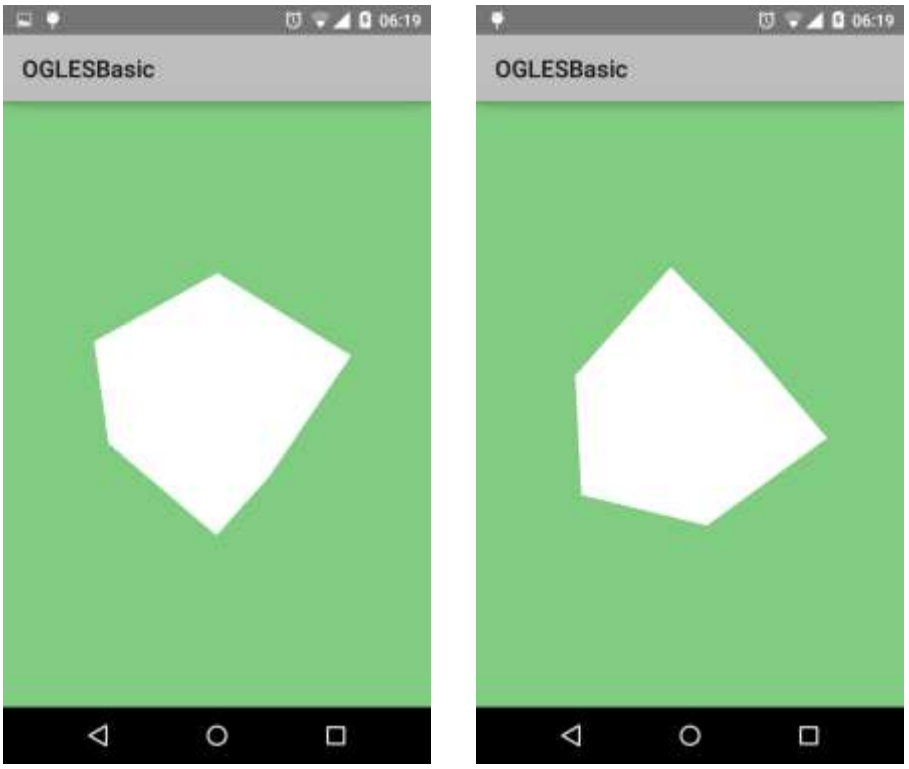
```
public class GLRendererEx implements GLSurfaceView.Renderer {

    private GLTrianguloEx mTrianguloEx;
    private GLCuboEx mCuboEx;

    public GLRendererEx() {
        mTrianguloEx = new GLTrianguloEx();
        mCuboEx = new GLCuboEx();
    }

…

    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glDisable(GL10.GL_DITHER);
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        GLU.gluLookAt(gl,0,0,-3,0,0,0,0,2,0);

       long time = SystemClock.uptimeMillis() % 4000L;
        float angle = .090f * ((int)time);
```

GLRendererEx

```
        gl.glRotatef(angle,1,0,1);

        mTrianguloEx.draw(gl);
        mCuboEx.draw(gl);
    }
}
```

O que fizemos foi, basicamente, pegar o tempo como referência para atualizar o ângulo que iremos rodar no eixo-xz.

Fig. 10.3.1 - Cubo girando em 3D



Pronto, essa foi nossa aplicação simples para a introdução ao OpenGL ES.

GitHub

Você pode importar fazer um *check out* do projeto via GitHub:

https://github.com/qumaciel/igti_android_oglesbasic.git

Capítulo 11. Performance e UX Review

Neste capítulo, discutiremos sobre performance e como lidar com opiniões sobre a aplicação, sejam qualitativas, ou quantitativas.

Performance

Quando seu aplicativo vai a Google Play, os usuários dão notas baixas devido à performance numa quantidade três vezes maior quando comparada a outros motivos.

Por isso, não tenha medo das ferramentas de *profiling* disponíveis. Temos três principais ferramentas:

- TraceView – Verifica problemas de processamento;
- Memory monitor - Monitora a utilização da memória;
- HeapViewer – Verifica problemas de computação antes que algo seja desenhado na tela.

Além delas, podemos utilizar o depurador de overdraw, que está nas opções de desenvolvedor do seu Android:

Fig. 11.1.1 - O depurador de overdraw da GPU.

Esse depurador é muito útil para resolver um dos problemas mais comuns em aplicações Android: desenhar coisas desnecessárias na tela.

Sua aplicação deve rodar a 60 quadros por segundo. Isso nos dá 16,666… ms/quadro para fazer toda a computação necessária até a próxima atualização. Se demorar mais que isso, o Android pula um quadro, deixando sua aplicação com uma fluidez menor.

Discutimos melhor esse problema no vídeo do capítulo.

Bateria

Outra coisa muito importante a considerar é o consumo de bateria da sua aplicação. Afinal, de nada adianta ter o melhor aplicativo do mundo se o usuário não tem bateria para utilizá-lo.

Existem três grandes vilões da bateria, são eles:

- Tela;
- GPS;
- Rede.

A tela já é um vilão conhecido. Basta abrir as configurações de bateria de seu dispositivo Android e verá (na média 60-70% do consumo). Não há muito o que fazer nesse caso, exceto controlar o overdrawing e a quantidade de pixels brancos.

O GPS é outro vilão, porém mais difícil de controlar. A sugestão aqui é estudar com cuidado a frequência e a precisão da aquisição. Quanto mais frequente você quiser pedir a localização (por exemplo, uma requisição a cada 100ms) e mais preciso (utilizando GPS, Wi-Fi e antena de telefonia), mais intensivo será, consumindo mais recursos.

A rede também é colocada como um vilão, pois, quanto mais intensa é a transferência de dados, seja via 2G/3G/4G ou Wi-Fi, maior será o consumo. Principalmente se o sinal estiver fraco.

Feedback qualitativo e quantitativo

Tanto as avaliações dos usuários quanto o UX Review do Google devem ser tratados seriamente. Em geral, a pergunta que fica é: Como fazer dessas informações algo útil para desenvolver a melhor aplicação possível?

Dividimos esse feedback em duas categorias: quantitativo e qualitativo.

O feedback quantitativo traz informação através de números, por exemplo, quantas vezes o usuário abre minha aplicação por dia?

Uma boa prática para esse tipo de feedback é a utilização da API do Google Analytics.

O qualitativo é feito, geralmente, utilizando comunicação direta entre usuários e desenvolvedores. Uma espécie de entrevista: “Como foi sua experiência com esta aplicação? Comente.”.

Seu aplicativo passa por uma revisão quando você submete para a Google Play. Esse relatório é feito por desenvolvedores experientes sobre diferentes aspectos do aplicativo, e.g., design, padrões de interação, acessibilidade e etc.

Capítulo 12. Android em tudo

Neste capítulo, discutiremos sobre o Android em todos os lugares. Ao invés de uma exposição dissertativa sobre o assunto, deixaremos perguntas em aberto. Faremos isso porque o sistema operacional e os aplicativos para essas plataformas ainda precisam de maturação.

Sobre TVs inteligentes

As TVs inteligentes sofrem com a usabilidade. O controle remoto é péssimo para quem precisa digitar algo, por exemplo.

Quais seriam as melhores práticas para solucionar esse problema? Um aplicativo no celular?

Talvez esse seja um fator limitante do dispositivo.

Sobre óculos inteligentes

Os óculos inteligentes surgem com uma grande desconfiança dos usuários por vários motivos (apelo visual, preço e etc.). Tanto que o Google Glass foi um projeto descontinuado por tempo ilimitado.

Como esse dispositivo pode ser melhor absorvido? Quais seriam as necessidades para esse tipo de dispositivo?!

Com a absorção da realidade aumentada, seria a hora de voltar com o projeto?

Sobre carros inteligentes

Já não é de hoje que as montadoras vendem carros com computadores de bordo. Mas a interface desses computadores ainda não é boa.

Como um SO tipo o Android pode ser melhor absorvido num carro? Quais seriam suas melhores utilidades?

Aplicações voltadas a navegação e entretenimento (música e vídeo) ainda são as mais simples, porém podemos ir além e pensar no sistema operacional controlando outras coisas do carro, por exemplo alerta de pneus murchos ou temperatura alta.

Sobre relógios inteligentes

Os relógios inteligentes já despertam um grande interesse em seu lançamento.

Como esse dispositivo pode ser melhor absorvido? Quais seriam as necessidades para esse tipo de dispositivo?

Atualmente, aplicações de notificações (lembretes, IM, etc.) e para o público fit, com interface com monitor de frequência cardíaca são os de maior absorção.

Sobre casas inteligentes

As casas inteligentes ainda parecem estar mais longe de serem absorvidas por diversos motivos, apesar de já entrarem em alguns dispositivos da casa, como as TVs.

Quais seriam os desafios para ambientes controlados e eletrodomésticos inteligentes?

Em termos de acessibilidade, isso seria ótimo. Ter ambientes controlados por voz ou gestos resolveria diversos problemas para quem tem limitações motoras, por exemplo.

Capítulo 13. Os três pilares

Neste capítulo discutiremos sobre os três pilares de uma aplicação Android de sucesso. Segundo o Google, o Android é desenvolvido em cima de uma visão criativa organizada em três pilares: encante-me, simplifique minha vida e torne-me fantástico.

Encante-me

Neste primeiro pilar, temos a ideia de que nossas aplicações têm que se mostrar e se comportar naturalmente.

Isso não é atingido apenas com uma interface com o usuário bem pensada, mas, também, com uma experiência boa com a aplicação.

Experiência boa ao ponto de deixar uma sensação de que ela foi feita especificamente para aquele usuário. Essa identidade é muito importante para que o usuário se sinta parte da aplicação e queira utilizá-la

Simplifique minha vida

No segundo pilar está o apelo para que a aplicação seja intuitiva e fácil de entender, tanto para um usuário que acaba de instalar a aplicação e tenta aprender o que ela faz quanto para o usuário que retorna a ela.

As diretrizes do material design aqui são evidentes: principalmente no que toca a ter a mesma experiência em diferentes dispositivos (como comentamos no Capítulo 1).

Torne-me fantástico

No último pilar temos a oportunidade de utilizar as vantagens do Android em toda sua glória. O aplicativo tem que ser mais que simplesmente a soma das partes, ele tem a versatilidade de integrar várias aplicações e conta com mais de um ponto de entrada.

Você implementa a visão desse terceiro pilar adequadamente quando o usuário utiliza e quer voltar para sua aplicação sempre que possível e necessário (até quando ele não sabe que é necessário).

Referências

[1] MAHAPATRA, Lisa. “Android Vs. iOS: What’s The Most Popular Mobile Operating System In Your Country?”. Disponível em: <www.ibtimes.com/android-vs-ios-whats- most-popular-mobile-operating-system-your- country-1464892>. Acesso em: 2 Mai. 2018.

[2] ELMER-DEWITT, Philip. “Don’t mistake Apple’s market share for its installed base”. Disponível em: <http://fortune.com/2014/01/10/dont-mistake-apples-market-share-for-its-installed-base/>. Acesso em: 2 Mai. 2018.

[3] YAROW, Jay. “This Chart Shows Google’s Incredible Domination of the World’s Computing Platforms”. Disponível em: <http://www.businessinsider.in/This-Chart-Shows-Googles-Incredible-Domination-Of-The-Worlds-Computing-Platforms/amp_articleshow/32869933.cms>. Acesso em: 2 Mai. 2018.

[4] AMADEO, Ron. “The history of Android”. Disponível em: <https://arstechnica.com/gadgets/2016/10/building-android-a-40000-word-history-of-googles-mobile-os/>. Acesso em: 2 Mai. 2018.

[5] Oracle. *Java JDK Downloads*. Disponível em: <http://www.oracle.com/technetwork/java/javase/downloads/>. Acesso em: 2 Mai. 2018.

[6] *Android Studio*. Disponível em: <http://developer.android.com/sdk/installing/studio.html>. Acesso em: 2 Mai. 2018.

[7] Exemplos do Google. Disponível em: <http://developer.android.com/samples/new/>. Acesso em: 2 Mai. 2018.

[8] Android Auto comes to more than 100 car models for 2017. Disponível em: <https://www.usatoday.com/story/money/cars/2016/10/11/android-auto-comes-more-than-100-car-models-2017/91884366/>. Acesso em: 2 Mai. 2018.

[9] “A long awaited privacy awakening is here”. Disponível em: <http://money.cnn.com/2018/03/28/technology/facebook-data-awakening/index.html>. Acesso em: 2 Mai. 2018.

[10] Architecture. Disponível em: <https://source.android.com/devices/architecture/>. Acesso em: 2 Mai. 2018.

[11] HAL. Disponível em: <https://source.android.com/devices/architecture/hal>. Acesso em: 2 Mai. 2018.

[12] Material Design. Disponível em: <http://www.google.com/design/spec/material-design/introduction.html>. Acesso em: 2 Mai. 2018.

[11] KUHLEN, Rainer. *Informationsethik*: Umgang mit Wissen und Information in elektronischen Räumen. Universitätsverlag Konstanz, 2004.

[12] The Need for Privacy Policies in Mobile Apps – an Overview. Disponível em: <https://www.iubenda.com/blog/the-need-for-privacy-policies-in-mobile-apps-an-overview/>. Acesso em: 2 Mai. 2018.

[13] Marco Civil da Internet. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2014/lei/l12965.htm?utm_source=blog&utm_campaign=rc_blogpost>. Acesso em: 2 Mai. 2018.

[14] Help center. Disponível em: <https://support.google.com/googleplay/android-developer/#topic=2364761>. Acesso em: 2 Mai. 2018.

[15] Developer contente policy. Disponível em: <https://play.google.com/about/developer-content-policy/#!?modal_active=none>. Acesso em: 2 Mai. 2018.

[16] Privacy and Security. Disponível em: <https://play.google.com/about/privacy-security-deception/>. Acesso em: 2 Mai. 2018.

[17] Contrato de distribuição do desenvolvedor do Google Play. Disponível em: <https://play.google.com/about/developer-distribution-agreement.html>. Acesso em: 2 Mai. 2018.

[18] Have a mobile app? You need a privacy policy. Disponível em: <https://www.iubenda.com/en/mobile>. Acesso em: 2 Mai. 2018.

[19] Google Play Console. Disponível em: <https://play.google.com/apps/publish/signup/>. Acesso em: 2 Mai. 2018.

PHILLIPS, B., HARDY, B. *Android Programming: The Big Nerd Ranch Guide.* Big Nerd Ranch Inc. Pearson 2013.