Desenvolvimento para Dispositivos Móveis

Desenvolvimento para Dispositivos Móveis

Hudson Cadan Scheffer

Dados Internacionais de Catalogação na Publicação (CIP)

Scheffer, Hudson Cadan
S316d Desenvolvimento para dispositivos móveis / Hudson Cadan Scheffer. – Londrina : Editora e Distribuidora Educacional S.A., 2018.
256 p.

ISBN 978-85-522-1107-5

1. Programação mobile. 2. Android. 3. Android Studio. I. Scheffer, Hudson Cadan. II. Título.

CDD 005.3

Thamiris Mantovani CRB-8/9491

2018
Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

Palavras do autor

Seja bem-vindo à disciplina de Desenvolvimento para Dispositivos Móveis.

Estamos vivendo uma época de revolução digital, em que usuários utilizam seus dispositivos móveis tanto profissionalmente quanto no âmbito pessoal. Segundo o site *International Data Corporation* (2017), estima-se que aproximadamente 85% do mercado mundial de *smartphones* está destinado ao sistema Android. Diante do disposto, nesta disciplina iremos focar em desenvolvimento de aplicativos para *smartphones* com sistema operacional Android. Em particular, o objetivo deste material didático é apresentar desde conceitos básicos até recursos avançados para a programação de aplicativos Android.

Você irá conhecer conceitos e recursos, e aplicará técnicas para o desenvolvimento de quatro aplicativos, que serão apresentados cada um em uma unidade.

Na primeira unidade desta disciplina, estudaremos a instalação, configuração e toda a preparação necessária do ambiente para o desenvolvimento de um aplicativo Android. Para que você possa conhecer a estrutura de um projeto Android, desenvolveremos, no decorrer da unidade, um aplicativo que realizará cálculos específicos.

Para a segunda unidade, serão apresentados novos componentes visuais para serem integrados aos aplicativos e o conceito de armazenamento local com estrutura de chave-valor, um recurso muito utilizado para personalização de aplicativos.

Vamos continuar com nosso conceito de armazenamento local, porém, agora, vamos trabalhar com banco de dados relacional. Na terceira unidade, você será desafiado a usar a criatividade para desenvolver um jogo com recursos de banco de dados SQL.

Na quarta unidade, abordaremos o conceito de banco de dados *NoSQL* na nuvem. Agora é hora de se conectar ao mundo. Você conhecerá ferramentas para o desenvolvimento de aplicativos com conexão em banco de dados na nuvem.

Seja sempre curioso para explorar novos recursos. Esperamos que este material didático o guie até o mercado de trabalho.

Tecnologias de desenvolvimento para dispositivos móveis

Convite ao estudo

Seja bem-vindo à primeira unidade de estudo de desenvolvimento para dispositivos móveis. Os conteúdos que serão apresentados nesta unidade nos permitirão conhecer e compreender a estrutura de um projeto Android e criar aplicativos. Devemos, primeiramente, conhecer todas as ferramentas necessárias para o desenvolvimento de aplicativos Android e, após a instalação dessas ferramentas, criaremos o nosso primeiro aplicativo.

Vejamos o caso de uma empresa que atua no ramo de logística e possui filiais em diversos estados brasileiros para oferecer um serviço mais ágil aos seus clientes. Essas filiais estão encarregadas de encaminhar as mercadorias para o seu destino final, dentro do respectivo estado em que atuam.

De acordo com a legislação brasileira, é previsto a cada estado arrecadar o Imposto sobre Circulação de Mercadorias e Prestação de Serviços - ICMS. Desta forma, cada estado brasileiro está autorizado a legislar sobre o ICMS, definindo qual porcentagem deve ser cobrada em relação ao valor das mercadorias e dos serviços que circulam dentro do estado.

Para a emissão de uma nota fiscal, devem ser incluídos todos os impostos previstos em lei, inclusive o ICMS. Sabe-se que as mercadorias não podem circular sem nota fiscal, então as filiais também estão encarregadas de emiti-las com o valor do ICMS já incluído para que as mercadorias possam finalmente ser encaminhadas.

Com o avanço da tecnologia, os diretores da empresa sabem que os seus colaboradores estão sempre conectados aos seus

smartphones. Assim, observaram que poderia ser vantajoso contratar uma equipe de programadores especializados em dispositivos móveis para desenvolverem um aplicativo que calculasse o valor do ICMS para todas as filiais.

Com a vaga de programador aberta, você foi selecionado para atuar com a equipe de desenvolvimento e deverá criar um aplicativo para dispositivos móveis com Android.

A sua colaboração com a equipe de desenvolvimento será dividida em três etapas. Inicialmente, você deverá preparar todas as ferramentas e configurações necessárias para iniciar o desenvolvimento do aplicativo para a empresa de logística. O próximo passo será criar a tela de interação com o usuário. Nesse momento, você deverá definir quais componentes e elementos estarão disponíveis para que os usuários possam interagir com o aplicativo. E, por fim, você acrescentará a lógica do aplicativo. Você aplicará técnicas de desenvolvimento para finalizar a entrega do primeiro aplicativo. Vamos para o nosso primeiro desafio?

Seção 1.1

Android Studio: IDE oficial

Diálogo aberto

Caro aluno, seja bem-vindo à primeira seção de desenvolvimento para dispositivos móveis. Vamos iniciar os estudos conhecendo as ferramentas necessárias para que possamos criar aplicativos Android. Ao longo do conteúdo, seguiremos todos os passos necessários nos processos de instalações dessas ferramentas, bem como compreenderemos a estrutura de um projeto no Android Studio.

Uma empresa de logística acaba de contratá-lo para desenvolver um aplicativo para dispositivos móveis com o sistema operacional Android e você deverá atuar com a equipe de desenvolvimento. Agora é hora de se reunir com a equipe e definir os requisitos que o aplicativo deve atender.

Para iniciar as atividades, é importante conhecer o público-alvo do aplicativo e coletar as informações necessárias para poder tomar decisões sobre a configuração do mesmo. Recomenda-se definir um relatório para que todos os envolvidos no projeto estejam cientes da estrutura do aplicativo e possam trabalhar dentro de um padrão, assim cada programador poderá focar no que lhe foi delegado.

Nessa seção, você deverá instalar e configurar o *Java SE Development Kit* – JDK (Kit de Desenvolvimento Java), a *Integrated Development Environment* – IDE (Ambiente de Desenvolvimento Integrado) oficial e, por fim, o *Software Development Kit* – SDK (Kit de Desenvolvimento de Software).

Após a instalação e configuração, defina junto com a equipe de desenvolvimento para qual versão Android o aplicativo será projetado e compilado, qual a versão mínima do Android no *smartphone* aceita para a instalação do aplicativo, como será feito o controle de versões e quaisquer outros itens que você e a equipe de desenvolvimento julgarem necessários.

Essa seção é fundamental para desenvolver qualquer aplicativo Android. Você criará e organizará toda a base da estrutura para o

desenvolvimento de futuros aplicativos. Siga os passos mencionados nessa seção e as mensagens de orientação que o Android Studio fornecer durante qualquer processo de instalação. Seja bem-vindo ao mundo Android!

Não pode faltar

Caro aluno, antes de iniciarmos os nossos trabalhos, gostaria de alertá-lo que, durante a elaboração desse material didático, foram utilizadas as versões mais recentes disponíveis das ferramentas de desenvolvimento. Você poderá encontrar passos diferentes dos apresentados nesse material, pois as ferramentas de desenvolvimento estão em constantes atualizações, a fim de oferecer um melhor serviço.

De acordo com Tanenbaum e Bos (2016), o Android é um sistema operacional projetado para dispositivos móveis baseado no núcleo Linux. Ainda segundo os mesmos autores, esse sistema operacional foi inicialmente desenvolvido pela empresa Android Inc., mas, em 2005, a empresa Google o comprou. Em 2008 foi lançado o primeiro dispositivo comercial com Android.

"Os aplicativos Android são desenvolvidos com Java – uma das linguagens de programação mais usadas do mundo" (DEITEL; DEITEL; WALD, 2016, p.4). A Google fornece um serviço de distribuição de aplicativos, jogos, filmes, músicas e outros conteúdos digitais conhecido como *Google Play*. Também é possível acessar o serviço por meio do aplicativo *Play Store*, disponível em dispositivos móveis com Android. O Google Play lista os aplicativos mais baixados no serviço, entre eles "Netflix" e "Spotify Music", com mais de cem milhões de instalações, e "WhatsApp Messenger" e "Instagram", com mais de 1 bilhão de instalações.

O Android possui várias versões de lançamento e, atualmente, está na versão 8.1, lançada em dezembro de 2017.

Pesquise mais

Acesse o painel com as versões lançadas do Android e analise seus nomes, níveis de API e a porcentagem de usuários em cada versão do

sistema. Um fato curioso sobre as versões é que todas elas recebem um nome de doce. Veja mais informações em: <https://developer.android.com/about/dashboards/index.html>. Acesso em: 1 mar. 2018. Para que você possa visualizar as *Platform versions* (Versões da plataforma), configure, na parte inferior direita da página, a exibição para inglês.

Agora é hora de começarmos a preparar a base da estrutura para o desenvolvimento de aplicativos para dispositivos móveis com sistema operacional Android.

As três primeiras etapas que seguiremos serão executadas na seguinte ordem:

1. Instalação do JDK – *Java SE Development Kit*, ou, Kit de Desenvolvimento Java.
2. Instalação da IDE Android Studio – *Integrated Development Environment*, ou, Ambiente de Desenvolvimento Integrado para Android Studio.
3. Instalação do SDK – *Software Development Kit*, ou, Kit de Desenvolvimento de Software.

Etapa 1: o JDK, conforme o próprio nome sugere, é um kit de desenvolvimento que permite aos programadores desenvolverem aplicações em Java. A empresa Oracle Corporation o fornece para download, com suporte a diversos sistemas operacionais. Por padrão, a empresa Oracle recomenda o uso do termo JDK para se referir ao *Java SE Development Kit*, disponível para download em <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Uma simples consulta do termo "JDK" em sites de pesquisa retornará esse *link*. Como alternativa, é possível acessar o site da empresa Oracle (www.oracle.com) e navegar pelos menus até a página de download do *Java SE Development Kit*. Certifique-se de fazer o download do JDK e, após finalizá-lo, execute o arquivo e siga os passos de instalação. Não há segredos nesse processo. A instalação não deve demorar para ser concluída.

Iniciaremos agora a segunda etapa da nossa configuração. Nela, instalaremos o Android Studio, o IDE oficial para desenvolvimento de aplicativos Android, fornecido pela Google e que, atualmente,

está na versão 3.1. O download do Android Studio está disponível em <https://developer.android.com/studio/index.html>. Como alternativa, pesquise por "Android Studio" em algum site de pesquisa e ele retornará a página oficial.

Dica

Para um melhor aproveitamento da disciplina, sugiro que leia todo o conteúdo na página do Android Studio. Nela, há diversas informações disponíveis, tais como novidades da plataforma, requisitos do sistema, dicas de desenvolvimento, entre muitas outras informações úteis.

Nesse momento, você passará pelos seguintes passos:

- Passo 1: acesse o site oficial do Android Studio.

- Passo 2: selecione *Download Android Studio*.

- Passo 3: você será solicitado a aceitar o termo de licença e condições.

- Passo 4: após concordar com o termo, você será redirecionado para a página de "Instruções de Instalação". Essa página contém um vídeo explicando passo a passo os procedimentos necessários para concluir com sucesso a instalação do Android Studio.

Também não há segredos nesse momento, todo o processo será guiado pelo assistente de instalação do Android Studio.

Após concluir a instalação, chega o momento de iniciar a terceira etapa: a instalação do SDK – *Software Development Kit*. O SDK permite aos programadores desenvolverem de forma nativa aplicativos para a plataforma Android.

Assimile

Sempre que a empresa Google lança uma nova versão do sistema Android, um SDK correspondente também é lançado. Se você desejar sempre utilizar os recursos mais recentes disponíveis para o Android, mantenha o seu SDK atualizado. O processo de instalação e gerenciamento do SDK é realizado no ambiente do Android Studio, por meio do *SDK Manager* (Gerenciador do SDK). Em alguns momentos, será necessário acessar o *SDK Manager* para fazer download de novos componentes ou atualizar os componentes já instalados.

Conforme mostrado no vídeo da página "Instruções de Instalação", no passo 4 da etapa 2, ao executar o Android Studio pela primeira vez, será solicitada a configuração de mais alguns passos:

Passo 1: importar ou não configurações já existentes. Selecione a opção "*Do not import settings*" ("Não importar configurações"), pois, como estamos configurando pela primeira vez, não possuímos configurações para serem importadas.

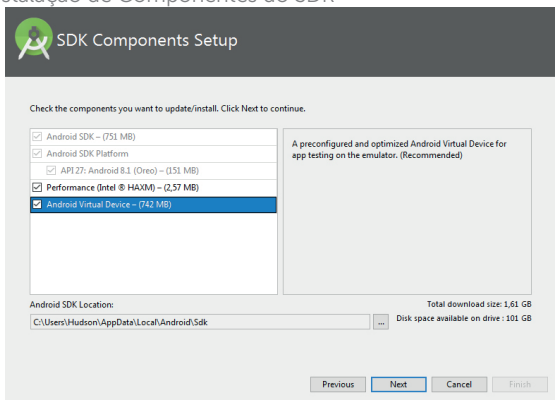
Passo 2: selecionar o tipo de instalação *Standard* (Padrão), pois queremos a instalação padrão.

Passo 3: selecionar a cor do *layout* do IDE Android Studio. Escolha a que mais lhe agrada.

Passo 4: selecionar os componentes para a instalação. Certifique-se de manter selecionada a opção *Android Virtual Device* (AVD), conforme exibida na Figura 1.1. Nessa opção, criamos um perfil de características de um *smartphone* ou *tablet* com Android, ou seja, como o próprio nome sugere, é a criação de um Dispositivo Virtual com Android. É importante criá-lo para que você possa simular e executar testes dos aplicativos que você irá desenvolver. Caso um AVD não seja instalado, você terá que executar os aplicativos diretamente no seu dispositivo com Android. Posteriormente, você poderá criar quantos AVDs julgar necessários com o *AVD Manager* (Gerenciador AVD), disponível no IDE Android Studio, acessando "*Tools*" > "Android" > "*AVD Manager*".

Passo 5: resumo de instalação, em que é exibido tudo o que será instalado. Esse processo pode levar algum tempo, aguarde!

Figura 1.1 | Instalação de Componentes do SDK



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Pesquise mais

Para reforçar o conceito de AVD e entender o funcionamento de criação e gerenciamento com o *AVD Manager*, recomendamos a leitura da documentação oficial, no site do Android Studio.

ANDROID. **Criar e gerenciar dispositivos virtuais**. Disponível em: <https://developer.android.com/studio/run/managing-avds.html>. Acesso em: 2 mar. 2018.

Pronto, toda a base da estrutura está configurada! Podemos então criar o nosso primeiro projeto e, para isso, seguiremos os passos a seguir:

Passo 1: selecione a opção "*Start a new Android Studio project*" ("Iniciar um novo projeto Android Studio").

Passo 2: conforme exibido na Figura 1.2, nesse segundo passo, você será solicitado a informar:

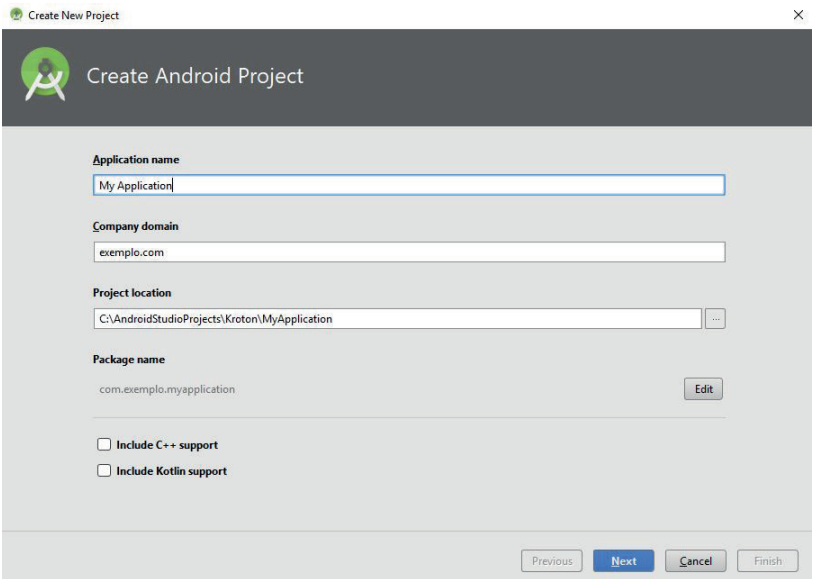
- *Application name* (Nome da aplicação): nome do projeto que será desenvolvido.
- *Company domain* (Domínio da companhia): costuma-se informar o site da empresa ou, para fins didáticos, utilize "exemplo.com".
- *Project location* (Localização do projeto): local onde o projeto será armazenado no disco rígido.
- *Package name* (nome do pacote): por padrão, o Android Studio define o nome do pacote automaticamente, invertendo o campo "*Company domain*" e acrescentando o campo "*Application name*". Esse padrão é adotado por que o *Package name* será utilizado para criar um identificador único para o seu aplicativo, ou seja, não poderá existir outro aplicativo com o mesmo identificador.
- E, por fim, se o aplicativo usará outras linguagens aceitas além do Java. Não marque outras linguagens neste momento.

Passo 3: você é solicitado a informar para qual tipo de dispositivo será desenvolvido o projeto e o SDK mínimo aceito, ou seja, qual versão mínima aceita do Android para que o aplicativo possa ser instalado. Observe que, nesse momento, o Android Studio exibe a porcentagem de usuários abrangida pelo seu aplicativo.

Passo 4: criar uma *Activity*. Este conceito será abordado em outra sessão, mas, de forma sucinta, para cada tela disponível no aplicativo, é criada uma *Activity*. Por padrão, cria-se uma "*Empty Activity*" ("Atividade Vazia").

Passo 5: e, por fim, com a criação de uma *Activity*, deve-se informar o seu nome e o nome do recurso de *layout*. Um projeto Android apresenta em arquivos distintos a classe Java com o código fonte e a estrutura visual que será apresentada para o usuário.

Figura 1.2 | Criar projeto Android



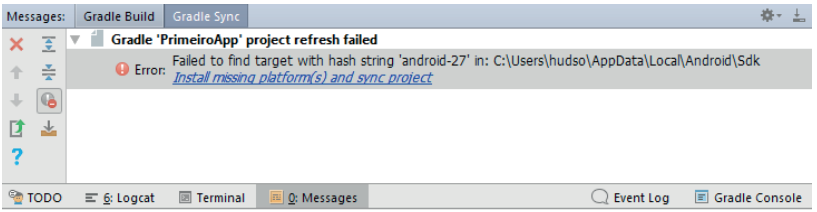
Fonte: captura de tela do Android Studio, elaborada pelo autor.

Reflita

Cada lançamento de versão do Android demanda meses até que o dispositivo móvel do usuário seja atualizado. Ao criar um projeto no Android Studio, o IDE fornece dados sobre a abrangência de usuários para cada versão disponível do sistema. Para o seu aplicativo, o que será mais importante: manter recursos atualizados com as SDKs mais recentes, porém com um público limitado, ou atingir um público maior de usuários com SDKs mais antigas, porém com menos recursos disponíveis?

Após a criação do primeiro projeto, é possível que seja solicitado o download de outros componentes ou, ainda, de outra versão do SDK. Observe na Figura 1.3 que a própria notificação "*Install missing platform(s) and sync project*" ("Instalar plataforma ausente e sincronizar projeto") permite clicar no link e fazer a instalação automaticamente.

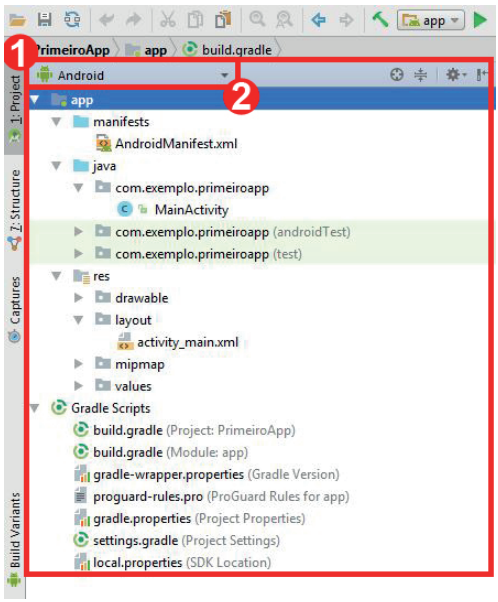
Figura 1.3 | Solicitação de download de novos componentes



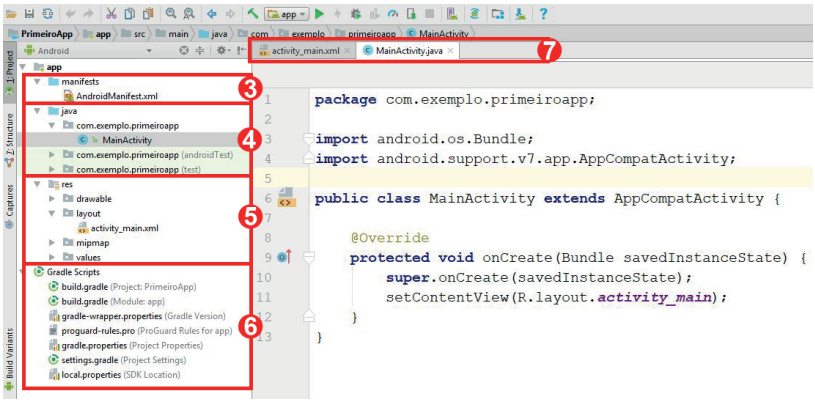
Fonte: captura de tela do Android Studio, elaborada pelo autor.

Agora, vamos entender como o projeto está estruturado np IDE Android Studio, conforme exibido nas Figuras 1.4 (a) e 1.4 (b).

Figura 1.4 | IDE Android Studio 3.0.1



(a) Janela *Projetct* da IDE Android Studio 3.0.1



b) IDE Android Studio 3.0.1

Fonte: captura de tela do Android Studio, elaborada pelo autor.

Vejamos o que representa cada número na figura:

1- Janela de Projeto: exibe de forma hierárquica todos os arquivos que compõe o projeto Android.

2- Filtragem dos arquivos: é uma lista suspensa, com várias opções que modificam a forma como os arquivos do projeto são exibidos. Por padrão, o IDE a configura como "Android", assim são exibidos apenas os arquivos mais importantes para o desenvolvimento do projeto. Para ver a estrutura real de arquivos, selecione a opção "*Project*".

3- Pasta com o arquivo "AndroidManifest.xml". Esse arquivo apresenta uma estrutura em formato XML, com configurações essenciais de um projeto. Na próxima seção, esse arquivo será apresentado de forma detalhada.

4- Pasta "java", utilizada para armazenar todo o código fonte do projeto. É possível organizar todas as classes dentro de pacotes específicos. No nosso exemplo temos a classe "MainActivity" dentro do pacote "com.exemplo.primeiroapp", que está localizado na pasta "java".

5- Pasta res: abreviação para "*resources*" ("recursos"), é a pasta em que estão localizados todos os recursos que não são código

fonte do seu projeto Android. São criados subdiretórios para cada tipo de recurso, sejam eles, ícones, imagens, *layouts* XML, *strings de UI*, menus e quaisquer outros recursos disponíveis na plataforma. Observe no nosso exemplo que existe um recurso de *layout* XML, na pasta "*res/layout*", chamado "activity_main.xml", que está vinculado à classe *MainActivity*.

6- *Gradle Scripts:* sempre que você executar o aplicativo que estiver desenvolvendo, será necessário criar um arquivo com extensão ".apk". Esse procedimento, conhecido como processo de compilação, envolve diversas ferramentas e etapas. O *Gradle* é um sistema de automatização de *builds*, ou seja, o *Gradle* é responsável por unir todas as etapas e recursos necessários para a geração do arquivo .apk.

7- Ao selecionar um arquivo na Janela de Projeto, com dois cliques, o arquivo será aberto e carregado nessa barra. No nosso exemplo, os arquivos "activity_main.xml" e "MainActivity.java" estão abertos.

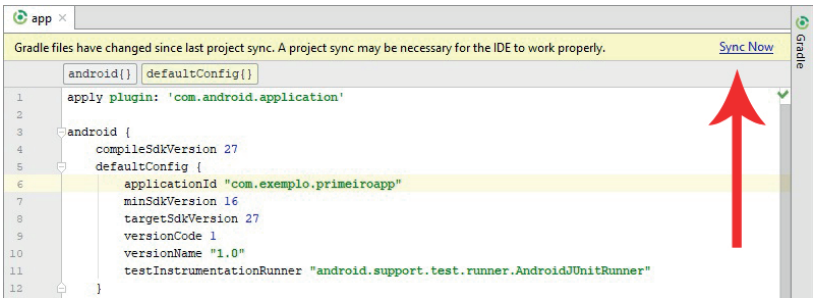
Ao trabalhar com o *Gradle*, é possível realizar ajustes para que possa ser gerado um arquivo ".apk" conforme a necessidade. Para isso, o arquivo "*build.gradle (Module: app)*" permite modificar diversas configurações de compilação. Vejamos as principais:

- *compileSdkVersion:* versão do SDK que o aplicativo será compilado.
- *applicationId:* identificador único para o aplicativo. Por padrão, o Android estabelece o mesmo nome de pacote.
- *minSdkVersion:* versão mínima aceita do SDK para a execução do aplicativo.
- *targetSdkVersion:* versão para a qual o aplicativo é direcionado.
- *versionCode:* código da versão do aplicativo. A cada nova versão do aplicativo, deve ser atribuído um código superior a essa variável para que o usuário seja notificado sobre a atualização disponível para o aplicativo.
- *versionName:* controle interno da empresa com os nomes das versões.
- *dependencies*: local para incluir bibliotecas externas no aplicativo. As bibliotecas externas permitem utilizar recursos e funcionalidades desenvolvidos por outros programadores no seu

aplicativo. Essas bibliotecas devem ser compatíveis com a mesma versão do *compileSdkVersion* e do *targetSdkVersion*.

Após a modificação de qualquer configuração do *Gradle*, é necessário sincronizar o projeto. Para isso, selecione a opção "*Sync Now*" ("Sincronizar Agora") exibida na parte superior direita do arquivo, conforme indicado na Figura 1.5.

Figura 1.5 | Sincronizar Projeto



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Exemplificando

É comum utilizarmos bibliotecas externas para o desenvolvimento do seu aplicativo. Você deve atentar ao fato de que essas bibliotecas devem ser compatíveis com o *compileSdkVersion* e o *targetSdkVersion*. Por padrão, o Android Studio utiliza uma biblioteca de compatibilidade (linha 23) para os recursos da plataforma. Caso haja alguma incompatibilidade nas configurações, o *Gradle* o notificará. Observe abaixo o conteúdo do *Gradle* e verifique a incompatibilidade de versões nas linhas 4 e 23. Na linha 4, é utilizado o nível de API 27 para *compileSdkVersion*, e na linha 23 é utilizada a versão 26 da biblioteca de compatibilidade.

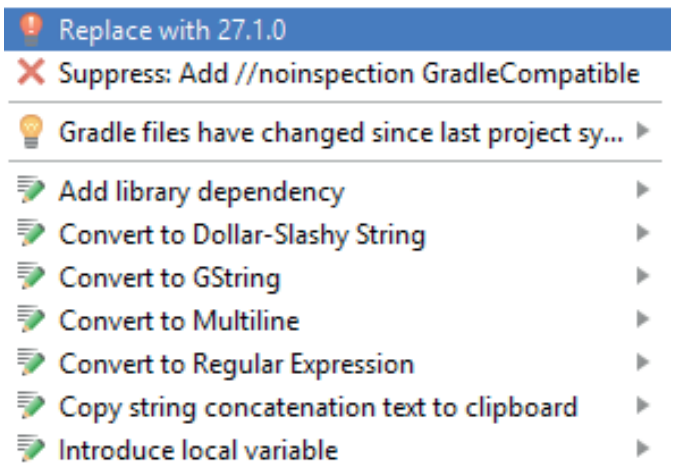
```
1            apply plugin: 'com.android.application'
2
3            android {
4                compileSdkVersion 27
```

```
5               defaultConfig {
6                       applicationId "com.exemplo.
primeiroapp"
7                   minSdkVersion 16
8                   targetSdkVersion 27
9                   versionCode 1
10                  versionName "1.0"
11                      testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
12          }
13              buildTypes {
14                  release {
15                      minifyEnabled false
16                                  proguardFiles
getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
17                  }
18          }
19          }
20
21          dependencies {
22              implementation fileTree(dir: 'libs',
include: ['*.jar'])
23                                  implementation
'com.android.support:appcompat-v7:26.1.0'
24                                  implementation
'com.android.support.constraint:constraint-
```

```
    layout:1.0.2'

25          testImplementation 'junit:junit:4.12'

26                              androidTestImplementation
    'com.android.support.test:runner:1.0.1'

27                              androidTestImplementation
    'com.android.support.test.espresso:espresso-
    core:3.0.1'

28      }
```

Para corrigir o erro apresentado acima, posicione o cursor em cima da linha notificada (linha 23) e pressione as teclas *ALT + ENTER*. Será aberta uma lista suspensa com várias opções disponíveis, conforme exibido na Figura 1.6. Selecione a opção que possa corrigir o erro apresentado. No nosso exemplo, selecionaremos *"Replace with 27.1.0"* ("Substituir com 27.1.0"), pois a biblioteca em destaque deve ser compatível com o *compileSdkVersion* 27 e *targetSdkVersion* 27.

Figura 1.6 | Lista de opções no Gradle



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Observe a notificação na barra inferior da IDE: *"This support library should not use a different version* (26) *than the 'compileSdkVersion'* (27)" ("Esta biblioteca de suporte não deveria usar uma versão diferente (26) do que a *'compileSdkVersion'* (27)").

Concluímos que, para desenvolver um aplicativo Android, é necessária a instalação de três ferramentas: JDK, Android Studio e o SDK.

Após a instalação das ferramentas, devemos criar um projeto no Android Studio que nos fornecerá:

(i) Um arquivo AndroidManifest.xml, com informações essenciais do aplicativo.

(ii) Um diretório "java", com todos os nossos códigos-fontes.

(iii) Um diretório "res", em que serão armazenados todos os recursos que não são códigos-fontes, por exemplo, recursos de *layout*, imagens, ícones, entre outros.

(iv) Um sistema de automatização de *builds*, conhecido como *Gradle*, que nos permitirá definir configurações de compilação.

Pesquise mais

Você pode conhecer detalhadamente a configuração do Gradle no link a seguir:

ANDROID. **Configure sua compilação**. Disponível em: <https://developer.android.com/studio/build/index.html>. Acesso em: 3 mar. 2018.

Chegamos ao final desta seção e temos todas as ferramentas necessárias configuradas para o desenvolvimento de aplicativos Android. Na próxima seção, estudaremos como criar uma interface de interação entre o usuário e o aplicativo. Continue seus estudos para conhecer novos recursos do Android Studio.

Sem medo de errar

Após uma empresa de logística contratá-lo para atuar como integrante da equipe de desenvolvimento de aplicativos para Android, a equipe se reuniu para definir os requisitos do aplicativo.

Para que todos os desenvolvedores possam trabalhar em cima de um mesmo padrão de projeto, você deve elaborar um relatório com todas as informações que você e sua equipe julgarem necessárias.

Para iniciarmos os trabalhos, devemos criar um projeto no Android Studio. Em seguida, acesse o arquivo "*build.gradle (Module: app)*" para definir a configuração em que o aplicativo será compilado.

Analise e modifique todas as configurações de compilação necessárias para que o aplicativo atenda aos requisitos definidos com a equipe de desenvolvimento. Não há resposta certa ou errada, pois a equipe de desenvolvimento pode julgar necessário atender a um público alvo diferenciado.

Vamos considerar a seguinte configuração:

(i) Nome do projeto definido como "ICMS".

(ii) Nome do pacote definido como "empresalogistica.com".

(iii) Criação de uma *EmptyActivity* com o nome padrão sugerido pelo Android Studio.

(iv) Devem ser disponibilizados os recursos mais recentes disponíveis na plataforma Android.

(v) O aplicativo deve atingir o maior número de usuários possíveis.

(vi) O controle de versão será um acumulador inteiro.

(vii) O nome da versão será construído sob a concatenação do ano, mês e dia de lançamento da versão.

Acrescente o conteúdo do arquivo "*build.gradle (Module app)*" e comente as configurações modificadas.

```
1.    apply plugin: 'com.android.application'
2.
3.    android {
4.        compileSdkVersion 27
5.        defaultConfig {
6.            applicationId "com.empresalogistica.
   icms"
7.            minSdkVersion 16
8.            targetSdkVersion 27
9.            versionCode 1
10.                versionName "2018-03-11"
11.                testInstrumentationRunner
   "android.support.test.runner.
   AndroidJUnitRunner"
```

```
12.             }
13.             buildTypes {
14.                 release {
15.                     minifyEnabled false
16.                     proguardFiles
   getDefaultProguardFile('proguard-android.txt'),
   'proguard-rules.pro'
17.                 }
18.             }
19.         }
20.
21.         dependencies {
22.             implementation fileTree(dir:
   'libs', include: ['*.jar'])
23.             implementation 'com.android.
   support:appcompat-v7:27.1.0'
24.             implementation 'com.android.
   support.constraint:constraint-layout:1.0.2'
25.             testImplementation
   'junit:junit:4.12'
26.             androidTestImplementation 'com.
   android.support.test:runner:1.0.1'
27.             androidTestImplementation 'com.
   android.support.test.espresso:espresso-
   core:3.0.1'
28.         }
```

De acordo com o código apresentado acima, podemos concluir que:

(i) Linha 4: a primeira versão do aplicativo será compilada com o SDK 27, que corresponde à versão 8.1.

(ii) Linha 6: o identificador único definido para o aplicativo é "com.empresalogistica.icms", ou seja, nenhum outro aplicativo poderá apresentar esse identificador.

(iii) Linha 7: para que o aplicativo possa ser instalado no dispositivo móvel do usuário, o mesmo deverá ter no mínimo a versão 4.1 do

Android, que corresponde ao nível de API 16.

(iv) Linha 8: o aplicativo também será direcionado ao SDK 27.

(v) Linha 9: o aplicativo está na sua primeira versão.

(vi) inha 10: construção do nome da primeira versão, conforme definido na reunião com a equipe de desenvolvimento;

(vii) Linha 23: definição da versão da biblioteca de compatibilidade, de acordo com o nível do *compileSdkVersion* e do *targetSdkVersion*.

Agora é possível compilar o aplicativo com o SDK 27 e o ele será instalado apenas em dispositivos móveis que possuem no mínimo a versão 4.1 do Android. Com essas configurações, finalizamos a primeira etapa de desenvolvimento do aplicativo para a empresa de logística.

Avançando na prática

Configurações de compilação

Descrição da situação-problema

Uma grande empresa especializada em desenvolvimento de aplicativos com conteúdo multimídia o contratou para desenvolver um aplicativo com recursos disponíveis apenas na nova versão do sistema Android. O "Modo Imagem em Imagem" é um tipo especial que permite exibir várias janelas, usado principalmente para a reprodução de vídeo. Você deve iniciar o seu projeto e configurá-lo para que apenas dispositivos com o sistema Android 8.0 ou superior possam instalar este novo aplicativo.

Resolução da situação-problema

Comece o desenvolvimento do seu aplicativo criando um projeto no Android Studio com uma *Empty Activity*.

A fim de atender ao requisito solicitado pela empresa, você deve permitir apenas aos usuários que possuem a versão 8.0 ou superior do Android a instalação do aplicativo. Para realizar esse ajuste, será necessário modificar as configurações de compilação no arquivo "build.gradle (Module: app)". Após acessar o arquivo, consulte o nível da API correspondente à versão 8.0 no site oficial do Android. Após

realizar a consulta, atribua o valor do nível da API para a configuração *minSdkVersion*. Será necessário também modificar as configurações *compileSdkVersion* e *targetSdkVersion* para um nível de API igual ou superior ao do *minSdkVersion*.

```
1. apply plugin: 'com.android.application'
2.
3.    android {
4.        compileSdkVersion 27
5.        defaultConfig {
6.            applicationId "com.exemplo.
   pictureinpicture"
7.            minSdkVersion 26
8.            targetSdkVersion 27
9.            versionCode 1
10.                versionName "1.0"
11.                testInstrumentationRunner
   "android.support.test.runner.
   AndroidJUnitRunner"
12.            }
13.            buildTypes {
14.                release {
15.                    minifyEnabled false
16.                    proguardFiles
   getDefaultProguardFile('proguard-android.txt'),
   'proguard-rules.pro'
17.                }
18.            }
19.        }
20.
21.        dependencies {
22.            implementation fileTree(dir:
   'libs', include: ['*.jar'])
```

```
23.             implementation 'com.android.
    support:appcompat-v7:27.1.0'
24.             implementation
    'com.android.support.constraint:constraint-
    layout:1.0.2'
25.             testImplementation
    'junit:junit:4.12'
26.             androidTestImplementation
    'com.android.support.test:runner:1.0.1'
27.             androidTestImplementation
    'com.android.support.test.espresso:espresso-
    core:3.0.1'
28.         }
```

Com a configuração do arquivo *"build.gradle (Module: app)"* ajustada conforme o conteúdo acima, você poderá compilar o aplicativo com o SDK 27 e será possível acrescentar ao aplicativo a funcionalidade "Modo Imagem em Imagem", disponível a partir dessa versão do SDK. Você também limitou a instalação do aplicativo para somente os usuários que possuem a versão 8.0 ou superior do Android instalada no dispositivo móvel.

Faça valer a pena

1. Segundo o site oficial do Android Studio, o IDE possui as ferramentas mais rápidas para a construção de aplicativos para qualquer tipo de dispositivo com Android. Ainda segundo o site, "recursos como edição de código de nível global, depuração, ferramentas de desempenho, sistema flexível de compilação e criação/implantação instantâneas permitem que você se concentre na criação de aplicativos exclusivos de alta qualidade." (ANDROID, [s.d.])

ANDROID. **Baixar o Android Studio e as SDK Tools**. [s.l.; s.d.; s.p.]. Disponível em: <https://developer.android.com/studio/index.html>. Acesso em: 24 abr. 2018.

Escolha a opção correta para que o ambiente de desenvolvimento para Android seja configurado corretamente.

a) Devemos instalar apenas IDE Android Studio e os demais recursos necessários serão instalados automaticamente.

b) Devemos instalar primeiramente o JDK, depois o SDK e, finalmente, o Android Studio.
c) Devemos instalar o Android Studio e verificar as notificações de componentes faltantes para então realizar o download do JDK e SDK.
d) Devemos instalar o Java SE Development Kit, depois o Android Studio e, por fim, o SDK.
e) Devemos instalar o Software Development Kit, depois o JDK e, finalmente, o Android Studio.

2. Para que você possa testar o seu aplicativo de forma mais eficiente, é sugerida a criação de um perfil de AVD para cada tipo de dispositivo que seu aplicativo deva suportar. Para realizar a criação e o gerenciamento de AVDs, utiliza-se o AVD Manager, disponível no ambiente Android Studio. Caso o programador decida por não configurar um perfil de AVD, é recomendo utilizar um dispositivo físico.
Sobre o gerenciamento de dispositivos virtuais:
I- É possível criar apenas um perfil de hardware para cada nível de API.
II- O AVD Manager permite realizar o download de imagens de sistema para que você possa realizar o maior número de testes.
III- Um AVD é constituído de um perfil de hardware, uma imagem de sistema e outras propriedades.

Assinale a alternativa correta.

a) Apenas a alternativa I está correta.
b) Apenas a alternativa II está correta.
c) Apenas a alternativa III está correta.
d) As alternativas I e II estão corretas.
e) As alternativas II e III estão corretas.

3. Para que você possa testar, implantar, assinar e distribuir o seu aplicativo é necessário realizar o processo de compilação. Esse processo irá compilar o código fonte com os recursos disponíveis no aplicativo e os empacotará em um arquivo com extensão ".apk". O Android Studio disponibiliza a ferramenta *Gradle* para que esse processo possa ser feito de forma automatizada. Uma grande característica do *Gradle* é que lhe é permitido definir configurações personalizadas de compilação.

Escolha a alternativa que apresenta a(s) configuração(ões) de compilação que o permite(m) definir o nível mínimo da API para que um aplicativo Android possa ser instalado.

a) “compileSdkVersion”,“minSdkVersion” e “targetSdkVersion”.
b) “compileSdkVersion”.
c) “versionCode” e “versionName”.
d) “minSdkVersion”.
e) “targetSdkVersion”.

Seção 1.2

Desenvolvendo UI com *ScrollView* e *LinearLayout*

Diálogo aberto

Caro desenvolvedor, seja bem-vindo à segunda seção de Desenvolvimento para Dispositivos Móveis com Android. Na primeira seção, nós configuramos todo o ambiente de desenvolvimento, bem como conhecemos o arquivo de configuração de compilação de um aplicativo para Android. O próximo passo visa conhecer os componentes e elementos disponíveis para construir uma tela de interação entre o aplicativo e o usuário.

A primeira fase do projeto foi concluída com a configuração da versão do SDK com o qual o aplicativo será compilado. Nessa nova etapa, você será responsável por construir a interface do aplicativo. Para tal função, é importante conhecer o fluxo de processamento que o aplicativo executará.

Sabe-se que cada estado brasileiro possui uma legislação específica com a porcentagem do ICMS definida. O aplicativo funcionará em todas as filiais do Brasil, então a porcentagem do ICMS irá variar de estado para estado, e o usuário deve ter a opção de inserir o estado para o qual deseja realizar o cálculo.

O aplicativo deverá calcular a porcentagem do ICMS do respectivo estado da filial sobre um valor de mercadoria. Após o cálculo, será exibido o valor da mercadoria acrescido o imposto.

Nesta seção, começaremos os estudos compreendendo como o sistema Android reconhece as telas disponíveis em um aplicativo para que o usuário possa interagir. E, para finalizar, serão apresentados elementos para construção de interfaces.

Você tem o poder de definir como os aplicativos devem interagir com os usuários. Lembre-se de que a interface de um aplicativo deve ser sempre intuitiva e de fácil manuseio. É hora de dar um passo à frente e assumir o controle da interação entre o usuário e o aplicativo.

Não pode faltar

Segundo Deitel, Deitel e Wald (2016), os aplicativos Android podem trabalhar com quatro tipos de componentes: (i) atividades; (ii) serviços; (iii) provedores de conteúdo; e (iv) receptores de transmissão por broadcast.

- As *Activities* ("Atividades") são telas que fornecem uma interface para que o usuário interaja com o aplicativo. Por exemplo, "um aplicativo de e-mail pode ter uma atividade que mostra uma lista de novos e-mails, outra atividade que compõe um e-mail e outra ainda que lê e-mails" (DEVELOPER, 2018, p. 1).
- Um serviço é um componente executado em segundo plano e não fornece uma interface para interação. Sua finalidade é realizar longas operações, por exemplo, um *download*. O *Service* ("Serviço") continuará em execução mesmo que o usuário alterne para outro aplicativo instalado no dispositivo móvel.
- Provedores de conteúdo gerenciam o acesso aos dados do seu aplicativo. Por exemplo, para que outro aplicativo acesse os dados do seu aplicativo, é necessário fornecer um provedor de conteúdo.
- O *Broadcast Receiver* recebe mensagens do sistema ou de outros aplicativos para que seja possível executar uma ação dentro do seu aplicativo. Por exemplo, é possível detectar, no seu aplicativo, quando o usuário se conecta à internet e, assim, realizar alguma operação.

Vamos iniciar os estudos com o componente *Activity*. Existem três formas de criarmos uma *Activity*:

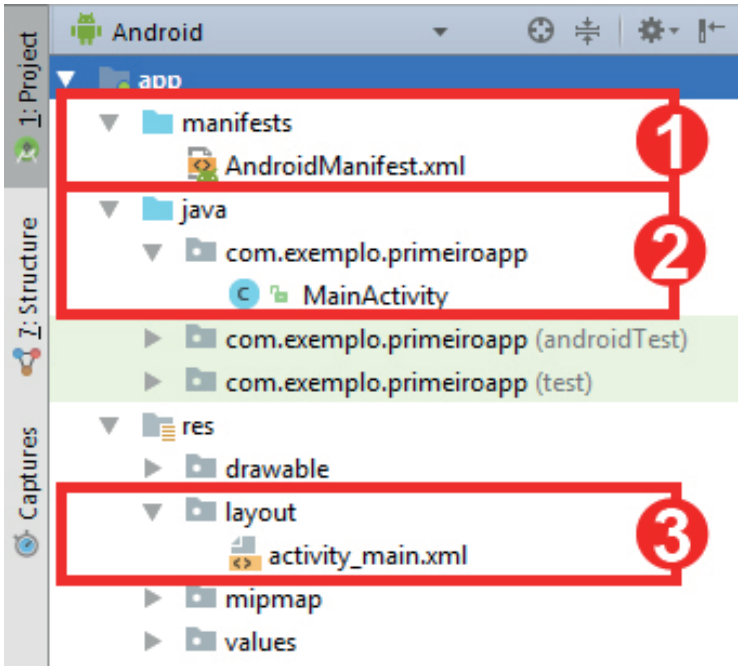
(i) Ao criar um projeto no Android Studio, é permitido criar uma *Activity*.

(ii) Por meio do menu "*File*" > "*New*" > "*Activity*", e selecionando o tipo de *Activity* que deseja criar.

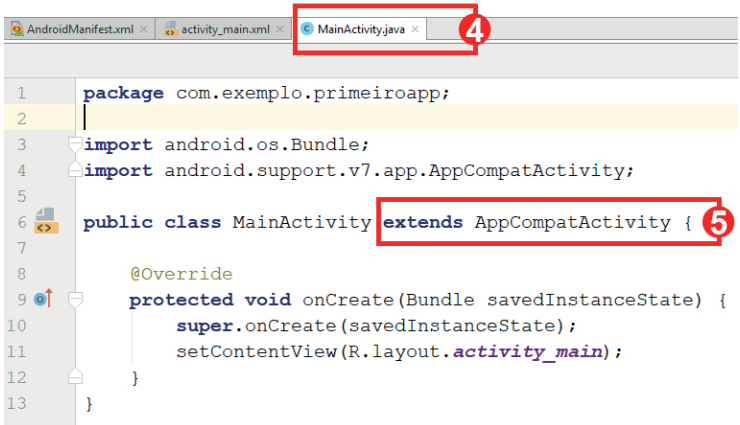
(iii) Manualmente, criando uma classe java e configurando a estrutura do projeto para reconhecer essa *Activity*. Abordaremos essa estrutura logo em frente.

Analise na Figura 1.7 (a) e (b) as características de toda *Activity*.

Figura 1.7 | Características de uma *Activity*



a) Estrutura dos arquivos para a criação de uma *Activity*



b) Arquivo com código-fonte de uma *Activity*

Fonte: captura de tela do Android Studio, elaborada pelo autor.

1. Toda *Activity* deve ser declarada no arquivo "AndroidManifest.xml". Observe na Figura 1.7 (a) onde está localizado o arquivo "AndroidManifest.xml".

2. Cada *Activity* fica armazenada na pasta "java", dentro do seu respectivo *package* ("pacote"). Na Figura 1.7 (a), a localização do arquivo de código-fonte da *MainActivity* é exibida hierarquicamente.

3. Geralmente, possui um recurso de *layout* vinculado. Esse recurso possui a extensão ".xml" e fica armazenado na pasta "res/layout", conforme exibido na Figura 1.7 (a).

4. As *Activities* possuem a extensão ".java". A Figura 1.7 (b) exibe o arquivo de código-fonte da *MainActivity* aberto com a extensão ".java".

5. Uma *Activity* é uma classe java que estende um objeto do tipo *Activity* ou *AppCompatActivity*. A figura 1.7-b exibe a construção da *MainActivity* e estende o objeto *AppCompatActivity*.

Assimile

Você deve estar se perguntando qual a diferença entre as classes *Activity* e *AppCompatActivity*.

Activity é uma classe que está sempre atualizada e apresenta recursos disponíveis apenas para a versão mais nova do Android.

AppCompatActivity é uma classe filha de *Activity* e pertence à biblioteca de suporte do Android, permitindo que os novos recursos funcionem nas versões mais antigas do sistema.

De acordo com W3schools (2018), XML foi projetado para armazenar e transportar dados em uma estrutura legível por humanos e máquinas. Para a construção de qualquer arquivo XML, de forma sucinta, é necessário indicar o início e o fim de cada elemento. Utilizamos a simbologia "<elemento>" para indicar o início, e a simbologia "</elemento>" para indicar o fim. Dentro de cada "<elemento>" pode haver atributos para determinar configurações específicas, bem como outros "<elementos-filhos>".

Pesquise mais

> XML é um texto que pode ser lido por humanos e por computadores e, no contexto do Android, ajuda a especificar os *layouts* e componentes a serem usados e seus atributos, como tamanho, posição, cor, tamanho do texto, margens e preenchimento. (DEITEL; DEITEL; WALD, 2016, p. 38)

O arquivo "AndroidManifest.xml" e outros recursos disponíveis na plataforma Android utilizam a estrutura XML. Para compreender melhor a estrutura de um arquivo XML, recomendo a leitura das regras de sintaxe do XML, disponíveis em: <https://www.w3schools.com/xml/xml_syntax.asp>. Acesso em: 19 mar. 2018.

Todo aplicativo Android deve possuir pelo menos um arquivo "AndroidManifest.xml", pois ele é o responsável por fornecer ao sistema operacional informações essenciais sobre o aplicativo. Observe o conteúdo de um arquivo "AndroidManifest.xml" a seguir:

```
1.          <?xml version="1.0" encoding="utf-8"?>
2.          <manifest xmlns:android=
  "http://schemas.android.com/apk/res/android"
3.              package="com.exemplo.primeiroapp">
4.
5.            <application
6.              android:allowBackup="true"
7.              android:icon="@mipmap/ic_launcher"
8.              android:label="@string/app_name"
9.              android:roundIcon= "@mipmap/ic_
   launcher_round"
10.             android:supportsRtl="true"
11.             android:theme="@style/AppTheme">
```

```
12.                 <activity android:name=".
    MainActivity">
13.                   <intent-filter>
14.                    <action android:name="android.
   intent.action.MAIN" />
15.                  <category android:name="android.
   intent.category.LAUNCHER" />
16.                   </intent-filter>
17.                 </activity>
18.             </application>
19.          </manifest>
```

- Na primeira linha, é configurada a codificação do arquivo XML. Por questões de caracteres internacionais, o atributo *encoding* recebe o valor de "utf-8" para evitar erros.
- O elemento raiz "<*manifest*>" deve conter dois atributos: (i) "*xmlns*", que já vem configurado por padrão pelo sistema, e (ii) "*package*", que indica o pacote em que as classes java estão armazenadas. Esse elemento deve obrigatoriamente conter o elemento-filho "<*application*>".
- Na linha 5, apresentamos o elemento "<*application*>", que possui atributos que afetam todos os seus "<elementos-filhos>". É neste elemento que devemos declarar todos os componentes de um aplicativo Android, conforme definimos no início dessa seção.
- O primeiro componente declarado nesse exemplo é uma *Activity* (linha 12). Esse elemento possui o atributo "android:name=. MainActivity", que indica o nome da *Activity*. Observe que não é necessário declarar o caminho completo do local que a *Activity* está armazenada (com.exemplo.primeiroapp.MainActivity), pois o caminho do *package* já está declarado no elemento raiz "<manifest>" (linha 3).
- Na linha 13, temos o elemento-filho "<*intent-filter*>". Esse elemento especifica como o componente deve se comportar dentro do sistema operacional. Ao utilizar os elementos-filhos "<action android:name="android.intent.action.MAIN" />" e

"<category android:name="android.intent.category.LAUNCHER" />", indicamos ao sistema que essa *Activity* deverá ser iniciada ao executar o aplicativo.

Pesquise mais

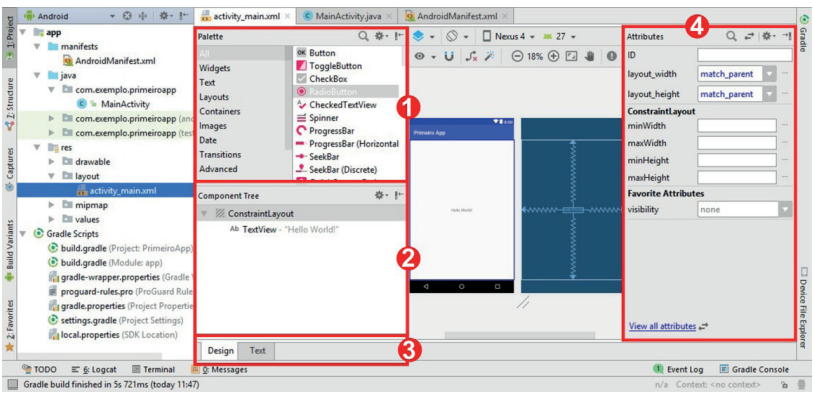
Você poderá conhecer a estrutura completa do arquivo AndroidManifest.xml na documentação oficial.

ANDROID. **Manifesto do aplicativo.** [s.l.; s.d.; s.p.]. Disponível em: <https://developer.android.com/guide/topics/manifest/manifest-intro.html?hl=pt-br>. Acesso em: 21 mar. 2018.

Temos em nosso projeto uma *Activity* declarada no arquivo "AndroidManifest.xml", que será a primeira tela de interação com o usuário ao executar o aplicativo.

É hora de analisarmos como é construída uma interface para o usuário. Você deve acessar o arquivo "activity_main.xml" na janela *Project*, conforme exibido na Figura 1.8.

Figura 1.8 | Editor de Layout



Fonte: captura de tela do Android Studio, elaborada pelo autor.

1: Janela *Palette*: apresenta todos os *Views* disponíveis para serem acrescentados na interface, por exemplo, um *LinearLayout*, uma caixa de texto, um botão ou uma imagem.

2: Janela *Component Tree:* exibe de forma hierárquica todos os elementos que estão sendo utilizados na interface.

3: Abas *Design* e *Text*: permitem selecionar a construção da interface no modo visual ou no modo texto. O modo *Design* permite arrastar e soltar os elementos, enquanto o modo *Text* permite construir a interface digitando os elementos no formato XML.

4: Janela *Attributes*: exibe os atributos disponíveis de acordo com o elemento selecionado na janela *Component Tree*. Por padrão, o Android Studio adiciona um *layout* do tipo *Constraint Layout* e, ao selecionarmos esse elemento no *Component Tree*, os seus atributos são exibidos na janela *Attributes*.

Vamos remover todos os elementos apresentados nesse padrão de interface e iniciar uma nova interface para darmos continuidade ao conteúdo desta seção. Consideremos a construção da Figura 1.9.

Figura 1.9 | Exemplo de uma interface

Fonte: adaptada de <https://commons.wikimedia.org/wiki/File:Novo_Logo_da_Kroton.jpg>. Acesso em: 21 jun. 2018.

A criação da interface é feita acrescentando elementos em XML, e sua estrutura é apresentada de forma hierárquica. Analisemos todos os elementos, selecionando a aba *Text* na parte inferior da IDE

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <ScrollView xmlns:android=
    "http://schemas.android.com/apk/res/android"
3.  xmlns:tools="http://schemas.android.com/
    tools"
4.  android:layout_width="match_parent"
5.  android:layout_height="match_parent"
6.  tools:context=
    "com.exemplo.primeiroapp.MainActivity">
7.
8.    <LinearLayout
9.      android:layout_width="match_parent"
10.     android:layout_height="wrap_content"
11.     android:orientation="vertical">
12.
13.   <ImageView
14.     android:layout_width="match_parent"
15.     android:layout_height="100dp"
16.     android:contentDescription="Kroton"
17.     android:scaleType="fitCenter"
18.     android:src="@drawable/kroton" />
19.
20.   <TextView
21.     android:layout_width="match_parent"
22.     android:layout_height="wrap_content"
23.     android:text="Qual o seu curso?" />
24.
```

```
25.     <EditText
26.       android:id="@+id/editText"
27.       android:layout_width="match_parent"
28.       android:layout_height="wrap_content"
29.       android:hint="Digite aqui"
30.       android:inputType="textCapWords" />
31.
32.     <TextView
33.       android:id="@+id/textView"
34.       android:layout_width="match_parent"
35.       android:layout_height="wrap_content"
36.       android:text="O curso digitado é " />
37.
38.     <Button
39.       android:id="@+id/button"
40.       android:layout_width="match_parent"
41.       android:layout_height="wrap_content"
42.       android:text="Confirmar curso" />
43.     </LinearLayout>
44.  </ScrollView>
```

- Linha 2: primeiramente, devemos escolher um tipo de *layout* que será definido como o elemento raiz. *ScrollView* é um tipo de *layout* que permite acrescentar barra de rolagem a uma tela de aplicativo. É permitido acrescentar apenas um elemento-filho para este tipo de *layout*.
- Linha 8: como é possível acrescentar apenas um elemento-filho para *ScrollView*, inserimos outro elemento também do tipo *layout*, conhecido como *LinearLayout*, para que possamos inserir quantos elementos-filhos quisermos na interface. *LinearLayout* é um *layout* que permite agrupar vários elementos-filhos, e esses elementos são exibidos um após o outro, de forma linear, como o nome sugere. A exibição pode ser horizontal ou vertical.

Os próximos elementos-filhos são *Views* que pertencem ao elemento *LinearLayout*.

- Linha 13: *ImageView* permite exibir uma imagem na interface.
- Linha 20 e 32: *TextView* permite exibir um texto para o usuário.
- Linha 25: *EditText* é uma caixa que permite ao usuário digitar e modificar um texto.
- Linha 38: *Button* adiciona um botão à interface para que o usuário possa clicar.

Reflita

Durante a criação da interface exibida na Figura 1.9, foram utilizados dois elementos do tipo *layout*: *ScrollView* e *LinearLayout*. Qual seria o resultado se você estivesse criando uma interface com diversos elementos-filhos e acrescentasse apenas o elemento *ScrollView*? Ou, ainda, se optasse por utilizar apenas o *LinearLayout*?

Após estudarmos todos os elementos apresentados na Figura 1.9, vamos estudar os seus atributos:

- *android:id:* define um nome para identificar o elemento e, por meio desse nome, podermos acessá-lo na classe "*MainActivity.java*". Devemos seguir a sintaxe exemplificada nas linhas 26, 33 e 39. Mais detalhes sobre como acessar o elemento serão explicados na próxima seção.
- *android:layout_width* e *android:layout_height:* definem, respectivamente, a largura e altura do elemento. É possível determinar um valor fixo, definindo valores numéricos acrescido de "dp". No nosso exemplo, definimos a altura da imagem para "100dp", conforme exemplificado na linha 15. Contudo, os dispositivos móveis com Android apresentam diversos tamanhos de telas e, a fim de evitar distorções durante a construção de uma interface, a plataforma fornece valores padrão para serem utilizados nesses atributos: (i) o valor "*match_parent*" força o elemento a preencher o espaço disponível na tela ou no elemento pai, se houver; e (ii) o valor "*wrap_content*" determina que o tamanho do elemento seja correspondente ao tamanho do seu conteúdo.
- *android:text*: define um texto para ser exibido no elemento.
- *androud:src*: define qual imagem será exibida na interface.

Existem algumas maneiras disponíveis para utilizar esse atributo. Para criarmos a interface apresentada na Figura 1.9, seguiremos os seguintes passos: (i) abra o *Windows Explorer*; (ii) copie uma imagem para a pasta do seu projeto "app/src/main/res/drawable". Para acessar a imagem salva nessa pasta a partir de um arquivo XML, basta utilizar o valor "@drawable/nome_do_arquivo_sem_a_extensao".

- *android:scaleType*: esse atributo controla como a imagem deve ser exibida.
- *android:inputType*: configura o tipo de entrada permitida para o elemento. Podemos citar como exemplos: (i) *numberDecimal*, para entradas de valores numéricos, e (ii) *textCapWords*, para entradas do tipo texto com a primeira letra de cada palavra em maiúsculo.
- *android:orientation*: indica a orientação de exibição dos elementos-filhos.

Exemplificando

Ao definir um atributo, você poderá utilizar as teclas CTRL + ESPAÇO para conhecer os valores pré-definidos pela plataforma para aquele atributo. Por exemplo, adicione ao elemento <ImageView> o atributo android:scaleType="". Posicione o cursor dentro das aspas, pressione CTRL + ESPAÇO e experimente cada valor para alterar como a imagem é exibida na interface. Observe os valores disponíveis para o atributo "android:scaleType", exibidos na figura 1.10.

Figura 1.10 | Valores disponíveis para o atributo *android:scaleType*

Fonte: elaborada pelo autor.

Concluímos o conteúdo desta seção compreendendo que o usuário pode interagir com um aplicativo por meio de uma *Activity*, que deve estar declarada no arquivo "AndroidManifest.xml".

Em seguida, aplicamos técnicas para criar uma interface de uma *Activity* utilizando os elementos de *layout ScrollView* e *LinearLayout*. Também estudamos vários elementos e atributos para que possamos modelar a construção de uma interface.

Esperamos você na próxima seção para que possamos compreender juntos como acessar os elementos de uma interface a partir do código-fonte de uma *Activity*. Até breve!

Sem medo de errar

Caro programador, a primeira etapa do desenvolvimento do aplicativo para a empresa de logística já está concluída. Você definiu com a equipe de desenvolvimento as configurações de compilação do aplicativo. Agora é hora de finalizar a segunda etapa.

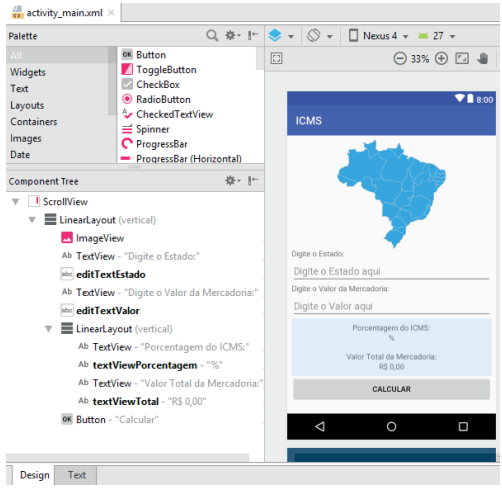
Você é responsável pela tarefa de construir a interface do aplicativo que será disponibilizado para todas as filiais da empresa de logística. Lembre-se que, ao criar o projeto na primeira etapa de desenvolvimento, também foi criada uma *Empty Activity*. Dessa forma, a atividade foi declarada automaticamente no arquivo "AndroidManifest.xml" para que o sistema Android pudesse reconhecê-la.

Para iniciar a construção da interface, abra o projeto criado na seção anterior e acesse o arquivo "activity_main.xml".

Defina quais elementos você utilizará para que os usuários interajam com o aplicativo. Lembre-se de que o usuário deverá inserir duas informações: o estado e o valor da mercadoria para os quais deseja realizar o cálculo. Você também deverá definir como as informações serão exibidas para o usuário após o cálculo do ICMS sobre o valor da mercadoria.

Observe a sugestão de interface apresentada na Figura 1.11:

Figura 1.11 | Criação de interface para o aplicativo de cálculo do ICMS



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Na Janela *Component Tree* são apresentados hierarquicamente todos os elementos utilizados na interface. Faça uma análise desses elementos e observe as seguintes dicas:

(i) O primeiro *View* incluído é um *layout ScrollView* que irá adicionar barra de rolagem na *Activity*.

(ii) O segundo elemento adicionado é um *LinearLayout*, com orientação definida em vertical. Dentro desse elemento, você poderá adicionar quantos *TextView*, *EditText*, *ImageView* e *Button* achar necessários para a interface atender aos requisitos do aplicativo.

(iii) Uma técnica muito utilizada é incluir um elemento *LinearLayout* (*oitavo elemento*) dentro de outro *LinearLayout* (segundo elemento). Essa técnica é utilizada para definir atributos específicos para os elementos-filhos. Por exemplo, caso você queria alterar a cor de fundo apenas de uma parte da interface, você poderá incluir o atributo *android:background="#e5efff"* no segundo *LinearLayout* incluído (oitavo elemento) e, assim, definirá uma cor de fundo apenas para os seus elementos-filhos.

(iv) Adicione os atributos *android:padding="8dp"* ou android:layout_margin="8dp" para alterar a distância da borda ou da margem entre os elementos. Esses atributos permitem apresentar uma finalização

mais sofisticada na interface. Experimente esses atributos e modifique os seus valores para explorar novos resultados.

(v) O atributo *android:textAlignment="center"* permite centralizar o texto em um elemento *TextView*.

Agora veja como os elementos e os atributos estão construídos, na aba *"Text"*, na parte inferior do IDE.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <ScrollView
    xmlns:android="http://schemas.android.com/
    apk/res/android"
3.    xmlns:tools="http://schemas.android.com/
    tools"
4.    android:layout_width="match_parent"
5.    android:layout_height="match_parent"
6.    tools:context=
    "com.empresalogistica.icms.MainActivity">
7.
8.    <LinearLayout
9.      android:layout_width="match_parent"
10.     android:layout_height="wrap_content"
11.     android:layout_margin="8dp"
12.     android:orientation="vertical">
13.
14.   <ImageView
15.     android:layout_width="match_parent"
16.     android:layout_height="200dp"
17.     android:contentDescription="Imagem Brasil"
18.     android:src="@drawable/brasil" />
19.
20.   <TextView
21.     android:layout_width="match_parent"
22.     android:layout_height="wrap_content"
```

```
23.       android:text="Digite o Estado:" />
24.
25.     <EditText
26.       android:id="@+id/editTextEstado"
27.       android:layout_width="match_parent"
28.       android:layout_height="wrap_content"
29.       android:hint="Digite o Estado aqui"
30.       android:inputType="textCapCharacters"
31.       android:maxLength="2" />
32.
33.     <TextView
34.       android:layout_width="match_parent"
35.       android:layout_height="wrap_content"
36.       android:text="Digite o Valor da Mercadoria:"
    />
37.
38.     <EditText
39.       android:id="@+id/editTextValor"
40.       android:layout_width="match_parent"
41.       android:layout_height="wrap_content"
42.       android:hint="Digite o Valor aqui"
43.       android:inputType="numberDecimal" />
44.
45.     <!--Este LinearLayout é incluído para que
    possamos definir uma cor de fundo para apenas
    os seus elementos-filhos-->
46.     <LinearLayout
47.       android:layout_width="match_parent"
48.       android:layout_height="wrap_content"
49.       android:background="#e5efff"
50.       android:orientation="vertical"
51.       android:padding="8dp">
```

```
52.
53.     <TextView
54.       android:layout_width="match_parent"
55.       android:layout_height="wrap_content"
56.       android:text="Porcentagem do ICMS:"
57.       android:textAlignment="center" />
58.
59.     <TextView
60.       android:id="@+id/textViewPorcentagem"
61.       android:layout_width="match_parent"
62.       android:layout_height="wrap_content"
63.       android:text="%"
64.       android:textAlignment="center" />
65.
66.     <TextView
67.       android:layout_width="match_parent"
68.       android:layout_height="wrap_content"
69.       android:layout_marginTop="16dp"
70.       android:text="Valor Total da Mercadoria:"
71.       android:textAlignment="center" />
72.
73.     <TextView
74.       android:id="@+id/textViewTotal"
75.       android:layout_width="match_parent"
76.       android:layout_height="wrap_content"
77.       android:text="R$ 0,00"
78.       android:textAlignment="center" />
79.     </LinearLayout>
80.
81.     <Button
82.       android:layout_width="match_parent"
83.       android:layout_height="wrap_content"
```

```
84.       android:text="Calcular" />
85.     </LinearLayout>
86.   </ScrollView>
```

Você poderá utilizar os elementos e atributos que julgar necessários para a construção da interface, bem como modificá-los para apresentar uma interface mais intuitiva.

Avançando na prática

Interface para simulação de investimentos

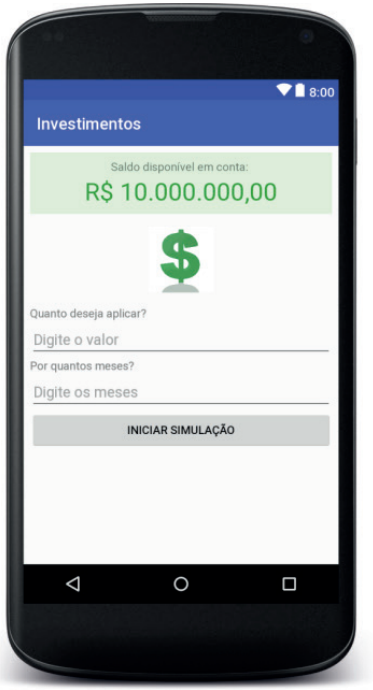
Descrição da situação-problema

Caro programador, você foi contratado por um banco privado para elaborar a interface de uma *Activity* que simule os investimentos dos seus clientes. O banco informou que os clientes poderão digitar o valor que desejam aplicar e também por quantos meses pretendem manter o investimento. Após os clientes digitarem essas informações, deverá ser disponibilizado um botão para simularem os investimentos disponíveis para os seus perfis. Crie a interface conforme solicitada pelo banco e sinta-se à vontade para explorar os atributos de cada elemento ao criar o *layout*.

Resolução da situação-problema

Crie um projeto no Android Studio com uma *Empty Activity* e altere o arquivo "activity_main.xml" para construir a interface solicitada pelo banco. A Figura 1.12 apresenta uma sugestão para a criação da interface da *Activity*, que irá simular os investimentos dos clientes.

Figura 1.12 | Interface para simular investimentos



Fonte: adaptado de <https://pixabay.com/pt/d%C3%B3lar-dinheiro-neg%C3%B3cios-moeda-311345/>.

Algumas dicas para a criação:

(i) Você poderá utilizar *ScrollView* como elemento raiz para adicionar barra de rolagem na tela do seu aplicativo.

(ii) *ScrollView* aceita apenas um elemento-filho, então adicione um *LinearLayout* como elemento-filho de *ScrollView*. Assim você poderá adicionar quantos elementos-filhos desejar para *LinearLayout*.

(iii) Adicione à interface os elementos que exibam informações para o usuário.

(iv) Acrescente também os elementos que permitirão aos usuários inserir os valores solicitados pelo banco antes de iniciarem a simulação de investimentos.

(v) No elemento *EditText* que receberá o valor do investimento, acrescente o atributo *android:inputType="numberDecimal"* para que apenas valores numéricos sejam inseridos.

(vi) Caso queira destacar alguma área na interface, adicione um *LinearLayout* a outro *LinearLayout*, para que você possa definir

atributos específicos para apenas um grupo de elementos-filhos.

(vii) É possível alterar o tamanho, cor e alinhamento do texto exibido no elemento *TextView* por meio dos atributos:

a. *Tamanho: android:textSize="30dp"*

b. *Cor: android:textColor="#298c17"*

c. *Alinhamento: android:textAlignment="center"*

As dicas estão listadas. Agora, adicione os elementos na ordem que julgar necessária, conciliando sempre os requisitos solicitados para a interface.

Faça valer a pena

1. Ao construir uma interface, é necessário incluir *Views* para que o usuário possa interagir com o aplicativo. Os recursos de *layouts* identificam as *Views* como elementos em XML, e cada elemento possui atributos que permitem personalizá-lo. Para definir a largura e a altura de um elemento utilizamos os atributos "android:width" e "android:height", respectivamente.

Qual valor deve ser atribuído para que uma *View* preencha todo o espaço disponível na largura e ocupe apenas o espaço do seu próprio conteúdo na altura?

a) android:width=" wrap_content" e android:height=" match_parent"
b) android:width="match_parent" e android:height="wrap_content"
c) android:width="match_parent" e android:height=" match_parent"
d) android:width=" wrap_content" e android:height="wrap_content"
e) android:width="match_parent" e android:height="100dp"

2. *Activities, Services, Content Provider* e *Broadcast Receivers* são componentes disponíveis no Android que permitem a construção de um aplicativo. Esses componentes apresentam diversas características, entre ela: (i) a possibilidade de criar uma tela para que o usuário possa interagir com o aplicativo; (ii) um componente que continua em execução, mesmo quando o usuário alterna para outro aplicativo; (iii) a criação do código-fonte em classes java que possuem a extensão ".class"; (iv) a criação de classes java que estendem um objeto do tipo *Activity* ou *AppCompatActivity*; (v) o armazenamento na pasta "java" e dentro do seu respectivo *package*; e (vi) a possibilidade de executar alguma funcionalidade do aplicativo quando o sistema é iniciado.

A respeito das *Activities*, selecione as características corretas.

a) I, III, IV e V estão corretas.
b) II, III, IV estão corretas.
c) I, IV e V estão corretas.
d) II, III e V estão corretas.
e) I, III e V estão corretas.

3. Para que o usuário possa interagir com o aplicativo, é necessário fornecer uma *Activity* com uma interface, e para que a interface seja criada, é necessário criar um recurso de *layout* que ficará armazenado na pasta "res/layout". Para implementar esse recurso de *layout*, são disponibilizados diversos *Views*. Podemos listar como exemplos o layout *ScrollView*, que permite adicionar uma barra de rolagem na tela do aplicativo, o *ImageView*, que permite inserir uma imagem na interface e, ainda, o *Button*, que permite adicionar um botão na interface para que o usuário possa pressioná-lo a fim de executar uma funcionalidade do aplicativo.

Hierarquicamente, quais *Views* são necessárias para construir um recurso de *layout* que forneça um texto informativo e uma caixa de texto.

a) LinearLayout, TextView e EditText.
b) TextView e EditText.
c) EditText e TextView.
d) LinearLayout, EditText e TextView.
e) ScrollView, TextView e EditText.

Seção 1.3

Trabalhando com *views* em *activities*

Diálogo aberto

Seja bem-vindo à última seção desta unidade. Na seção anterior, aprendemos como criar o *layout* de uma tela para que o usuário possa interagir com o aplicativo. Nesta seção, estudaremos como acessar os elementos que estão disponíveis no *layout* criado. A partir da interação entre o usuário e esses elementos, o aplicativo apresentará comportamentos programados por nós. Todos os aplicativos que utilizamos executam algum comportamento ao interagirmos com os elementos que dispõem o seu *layout*. Por exemplo, ao abrirmos um aplicativo de e-mails e selecionarmos um e-mail recebido, como resposta à interação será aberto o conteúdo deste e-mail selecionado.

Agora é hora de finalizar o aplicativo para a empresa de logística. A interface do aplicativo desenvolvida na etapa anterior foi aprovada. Agora você está na reta final e deverá programar toda a lógica para que o aplicativo funcione. Você construiu o recurso de *layout* em um arquivo separado do seu código fonte, então como vinculá-los? Como recuperar o estado e o valor da mercadoria digitados pelo colaborador da empresa. Como detectar uma ação de clique no botão "Calcular"? Ainda, como exibir o resultado do cálculo para o usuário?

Essa etapa é fundamental para o desenvolvimento de aplicativos. Para que você consiga cumprir sua missão, você irá aprender como criar referências aos elementos do *layout*, para que os valores digitados pelo usuário possam ser recuperados. Você também aprenderá a trabalhar com métodos nas *Activities*, que nos permitirão adicionar comportamentos específicos para o aplicativo.

Mantenha o foco, pois você está a um passo de finalizar o seu primeiro aplicativo. Com o conteúdo apresentado nessa seção, você poderá se desafiar a criar novas funcionalidades para ele. Mãos à obra!

Não pode faltar

Para que possamos compreender como acessar um elemento em um recurso de *layout*, vamos analisar a etapa seguinte do processo de compilação. Ao testar, implantar, assinar e distribuir o seu aplicativo, é necessário realizar o processo de compilação. Esse processo irá compilar o código fonte com os recursos disponíveis no aplicativo e os empacotará em um arquivo com extensão ".apk". Durante o processo de compilação, o Kit de Desenvolvimento de Software (SDK) executa a ferramenta *Android Asset Packaging Tool - aapt* ("Ferramenta de Empacotamento de Recursos do Android"). Essa ferramenta se encarrega de procurar por arquivos e referências de recursos e, como resultado, a Classe R é gerada automaticamente. Essa classe fornece referências para todos os recursos disponibilizados no diretório "res", e nunca deve ser alterada pelo programador, pois a própria ferramenta de compilação se encarrega de atualizá-la.

Exemplificando

Durante o desenvolvimento da lógica de algum componente Android, por exemplo, uma *Activity*, podemos acessar os recursos disponibilizados no diretório "res" por meio da Classe R. Utilizamos a sintaxe **`R.tipo_do_recurso.nome_do_recurso`**.

Por exemplo, para acessar o *layout* da *MainActivity*, utilizaremos então **`R.layout.activity_main`**.

Caso seja necessário acessar algum elemento dentro desse *layout*, devemos atribuir um ID para o elemento que queremos acessar, por meio do atributo **`android:id="@+id/nome_do_elemento`**". O símbolo "arroba" (@) no início do valor definido para esse atributo significa que o XML deverá identificar o restante do valor como um recurso de ID. O sinal de "mais" (+) indica que esse ID está sendo declarado para aquele elemento e que deverá ser gerada uma referência na Classe R para ser acessada mais tarde na classe "MainActivity.java". E, para acessá-la na classe "MainActivity.java", utilizaremos **`R.id.nome_do_elemento`**.

Lembre-se

Quando uma classe java estende outra classe, estamos trabalhando o conceito de Herança em Programação Orientada a Objetos. Caso haja dúvidas para relembrar esse conceito, recomendo que assista ao vídeo abaixo:

CURSO EM VÍDEO. **Curso POO Teoria #10ª** – Herança (Parte 1). (01min15s – 07min15s). Disponível em: <https://www.youtube.com/watch?v=_PZldwo0vVo>. Acesso em: 13 jun. 2018.

Ao sobrepor um método, utilizamos os conceitos de Polimorfismo. Para relembrar este conceito, assista ao vídeo a seguir:

CURSO EM VÍDEO. **Curso POO Teoria #12ª** – Conceito Polimorfismo (Parte 1). (01min30s – 13min40s). Disponível em: <https://www.youtube.com/watch?v=9-3-RMEMcq4>. Acesso em: 13 jun. 2018.

Agora vamos analisar a nossa Activity "MainActivity.java".

```
package com.exemplo.primeiroapp;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends
AppCompatActivity {

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

Linha 1: define o pacote a que esta classe pertence.
Linhas 3 a 4: importam as classes dos objetos que são utilizados

na *Activity*. Durante o desenvolvimento do seu aplicativo, você poderá utilizar outros objetos. As classes pertencentes a esses objetos devem ser importadas nessa seção. Normalmente o Android Studio importa essas classes automaticamente.

Linha 6: define a classe **MainActivity** como pública e estende um objeto do tipo **AppCompatActivity**. Ao estender esse objeto, a nossa classe *MainActivity* irá trabalhar como uma atividade.

Linha 8: **@Override** é conhecido como uma Anotação que indica que o método na linha seguinte será sobreposto.

Linha 9: declara o método **onCreate()**. A construção da assinatura deste método deve ser a mesma da classe pai, ou seja, AppCompatActivity possui um método "**protected void onCreate(Bundle savedInstanceState)**" e a nossa classe *MainActivity* está sobrepondo este método para que possamos acrescentar o nosso código ao inicializar a atividade.

Linha 10: executa os procedimentos definidos no método **onCreate()** da classe pai **AppCompatActivity**. É necessário chamar a execução da classe pai, pois estamos acrescentando lógica à inicialização da nossa Activity e não substituindo a lógica existente na classe pai.

Linha 11: **setContentView(int layout)** é outro método disponível em **AppCompatActivity**, responsável por definir a interface para o usuário. Esse método recebe como parâmetro um valor referente ao *layout* que será inicializado. Observe que, neste momento, o arquivo "**activity_main.xml**" está sendo acessado por meio da classe R.

Assimile

Quando o usuário inicia o aplicativo, uma *Activity* é exibida no dispositivo móvel. Para que esse processo aconteça, essa atividade executa diversos métodos. O primeiro método executado é o **onCreate()**, que:

(i) Inicialmente é declarado na classe **AppCompatActivity**, porém, devemos sobrepô-lo em cada *Activity* do nosso aplicativo para que possamos configurá-lo de acordo com a necessidade dele.

(ii) É responsável por apresentar a lógica de inicialização de uma *Activity*, por exemplo, utilizamos esse método para definir a interface para o usuário e inicializar os objetos que serão utilizados nessa *Activity*.

(iii) É chamado sempre que uma atividade é criada pela primeira vez.

Agora que nossa MainActivity é inicializada com uma interface definida por nós, devemos acessar os elementos dessa interface. Primeiramente definimos qual elemento queremos acessar e criamos uma instância desse objeto no método **onCreate()** da **MainActivity.**

```
1.    package com.exemplo.primeiroapp;
2.
3.    import android.os.Bundle;
4.    import android.support.v7.app.
      AppCompatActivity;
5.    import android.widget.EditText;
6.
7.    public class MainActivity extends
      AppCompatActivity {
8.
9.      private EditText mEditText;
10.
11.     @Override
12.     protected void onCreate(Bundle
      savedInstanceState) {
13.       super.onCreate(savedInstanceState);
14.       setContentView(R.layout.activity_main);
15.
16.       mEditText = (EditText) findViewById(R.
      id.editText);
17.     }
18.   }
```

Linha 9: declara o objeto que queremos acessar.

Linha 16: por meio do método **`findViewById(int id)`**, é possível inicializar o objeto que desejamos acessar na interface. Observe que também estamos utilizando a Classe R para fazer referência a um elemento dentro do recurso de *layout*.

Assimile

Toda classe *Activity* fornece o método **"`findViewById(int id)`"**, que é responsável por localizar um elemento de acordo com o parâmetro fornecido. Caso não seja encontrado nenhum elemento com o *id* fornecido no parâmetro, então o método retorna *null*.

O método *findViewById(int id)* localiza o elemento na interface inicializada.

Segundo o Developer (2018), houve uma mudança de assinatura no método **`findViewById(int id)`** a partir do SDK 26. Como resultado, se você definiu a configuração **`compileSdkVersion`** no arquivo "build.gradle (Modulo: app)" para o nível 26 ou superior, não será mais necessário realizar um *cast*, conforme exibido na linha 16 do código acima. A utilização do método `findViewById(int id)` poderá ser simplificada conforme linha abaixo.

```
16. mEditText = findViewById(R.id.editText);
```

Pesquise mais

Observe que:

(i) O objeto **`EditText`** na classe **`MainActivity`** recebe o nome "mEditText".

Por questões de convenção de nomeação, utilizamos nas classes:

(i) Campos não-públicos e não-estáticos começam com "m".

(ii) ampos estáticos começam com "s".

(iii) Outros campos começam com a letra minúscula.

(iv) Campos estáticos, públicos e definidos como "final", conhecidos como "Constantes", são declarados em letra maiúscula.

Verifique mais regras em:

ANDROID. **Follow field naming conventions**. In: AOSP Java Code Style for Contributors. [S.d.; s.p.]. Disponível em: <https://source.android.com/setup/code-style#follow-field-naming-conventions>. Acesso em: 13 jun. 2018.

Existem diversas maneiras para definir o comportamento do elemento *Button* ao ser pressionado pelo usuário. Nessa unidade seguiremos um passo-a-passo a fim de identificar a ação de toque no botão:

(i) Acesse o arquivo "activity_main.xml" e acrescente o atributo **`android:onClick="confirmar"`** no elemento `Button`.

```
38.     <Button
39.       android:id="@+id/button"
40.       android:layout_width="match_parent"
41.       android:layout_height="wrap_content"
42.       android:onClick="confirmar"
43.       android:text="Confirmar curso" />
```

(ii) Acesse a classe `MainActivity.class` e crie um método seguindo as características abaixo:

a. Deve ser público.

b. Não pode possuir retorno.

c. Deve ser criado com o mesmo nome definido no atributo **`onClick`** do elemento `Button`.

d. Deve receber como parâmetro um objeto do tipo `View`.

```
1.   package com.exemplo.primeiroapp;
2.
3.   import android.os.Bundle;
4.   import android.support.v7.app.
     AppCompatActivity;
5.   import android.view.View;
6.   import android.widget.EditText;
7.
8.     public class MainActivity extends
     AppCompatActivity {
9.
10.    private EditText mEditText;
11.
12.    @Override
13.    protected void onCreate(Bundle
     savedInstanceState) {
14.  super.onCreate(savedInstanceState);
15.  setContentView(R.layout.activity_main);
16.
17.  mEditText = findViewById(R.id.editText);
18.    }
19.
20.  public void confirmar(View view) {
21.  // Acrescente a lógica aqui
22.  }
23.  }
```

Toda a lógica desenvolvida dentro do método **confirmar(View view)** será executada quando o usuário pressionar o botão disponível na interface da **MainActivity**. O objeto "**mEditText**" faz referência ao elemento EditText na interface, portanto, agora, podemos recuperar o texto digitado pelo usuário por meio desse objeto.

```
20.  public void confirmar(View view) {
21.  // Acrescente a lógica aqui
22.  String textoDigitado = mEditText.getText().
     toString();
23.  }
```

Na linha 22, declaramos uma variável local **`String`** que recebe o valor digitado no **`EditText`** do *layout*. Vamos analisar o método **`getText()`**, conforme a Figura 1.13.

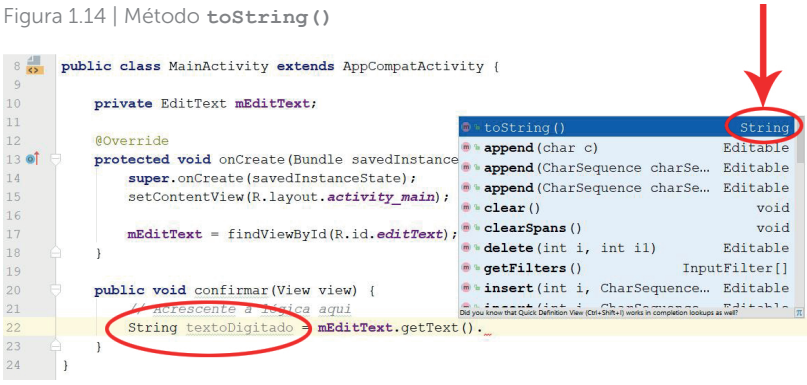
Figura 1.13 | Método `getText()`

Fonte: elaborada pelo autor.

Observe que, ao digitar o nome do objeto **`mEditText`** e adicionar o símbolo "." (ponto) para acessar os seus métodos, é exibida uma caixa com todos os métodos disponíveis. Na coluna à direita, conforme destacado na Figura 1.13, é exibido o tipo de retorno de cada método.

O objeto `EditText` possui o método **`getText()`**, que permite recuperar o texto digitado e retorna um objeto do tipo *Editable*. Como desejamos trabalhar com uma *String*, devemos chamar, ainda, o método **`toString()`** para recuperar o texto digitado, conforme exibido na Figura 1.14.

Figura 1.14 | Método `toString()`



Fonte: elaborada pelo autor.

Reflita

Sempre que recuperarmos um texto de um **EditText**, receberemos um objeto do tipo **String**. Imagine que nosso aplicativo exija que seja realizado um cálculo com valores digitados pelo usuário. Não será possível realizar um cálculo com objetos *String*. Desta forma, como devemos proceder?

Também é possível realizar o processo de exibir um texto na interface. Observe os códigos apresentados abaixo.

```
1.    package com.exemplo.primeiroapp;
2.
3.    import android.os.Bundle;
4.    import android.support.v7.app.
      AppCompatActivity;
5.    import android.view.View;
6.    import android.widget.EditText;
7.    import android.widget.TextView;
8.
9.    public class MainActivity extends
      AppCompatActivity {
10.
11.     private EditText mEditText;
12.     private TextView mTextView;
13.
14.     @Override
15.     protected void onCreate
      Bundle savedInstanceState) {
16.       super.onCreate(savedInstanceState);
17.       setContentView(R.layout.activity_main);
18.
19.       mEditText = findViewById(R.id.editText);
20.       mTextView = findViewById(R.id.textView);
21.     }
22.
23.     public void confirmar(View view) {
24.       // Acrescente a lógica aqui
```

```
25.         String textoDigitado = mEditText.
      getText().toString();
26.         mTextView.setText("O curso digitado é: "
      + textoDigitado);
27.       }
28.    }
```

Linha 12: declaramos o objeto a que desejamos ter acesso na interface.

Linha 20: criamos a instância para este objeto no método **onCreate()** por meio do método **findViewById()** e também a Classe R.

Linha 26: acessamos o método **setText()** e passamos, como parâmetro, o texto a ser exibido.

Com o código apresentado acima, sempre que o usuário pressionar o botão "Confirmar", o texto digitado no **EditText** será recuperado e exibido no **TextView**.

Pesquise mais

Existem duas formas para você instalar o seu aplicativo: por meio de um dispositivo físico com Android ou de um dispositivo virtual.

- Para que o aplicativo seja testado em um dispositivo físico, siga os seguintes passos:

I. Conecte o dispositivo no computador, com um cabo USB.

II. Em seguida, acesse, no dispositivo móvel, o menu de Configurações > Sobre > e clique várias vezes sobre o menu Número da Versão.

III. Após clicar várias vezes, será liberado o menu de Programador. Para acessar o menu recém-liberado, vá novamente em Configurações e acesse o menu Programador.

IV. Dentro do menu Programador, deverá ser ativado o menu Depuração USB. A opção Depuração USB serve para fins de desenvolvimento. Por exemplo, para instalar *apps* no dispositivo.

V. Finalmente, no Android Studio, selecione o menu Run > Run App. O seu dispositivo móvel deverá aparecer na listagem de dispositivos disponíveis para instalar o aplicativo.

- Para utilizar o emulador, recomendamos a seguinte leitura:

ANDROID. **Executar um aplicativo no Android Emulator.** In: Android Developers. [S.d.; s.p.]. Disponível em: <https://developer.android.com/studio/run/emulator.html?hl=pt-br#runningapp>. Acesso em: 13 jun. 2018.

Para executar o Android Emulator, será necessário criar um Dispositivo Virtual Android (AVD). Caso não tenha criado um para definir as características de um *smartphone*, siga os passos do artigo a seguir:

ANDROID. **Criar um AVD**. In: Android Developers. [S.d.; s.p.]. Disponível em: <https://developer.android.com/studio/run/managing-avds.html?hl=pt-br#createavd>. Acesso em: 13 jun. 2018.

Finalizamos essa seção compreendendo que:

(i) Para acessar algum recurso disponível no diretório "res", é necessário utilizar a Classe R.

(ii) Deve ser utilizado o método **`onCreate()`**, disponível em uma *Activity* para definir um *layout* e inicializar os objetos que serão utilizados naquela *Activity*.

(iii) Por meio do método **`setContentView(int layout)`** disponível em uma *Activity*, é possível definir um layout para a *Activity*.

(iv) Por meio do método **`findViewById(int id),`** também disponível em uma *Activity*, é possível instanciar os elementos de um *layout*.

(v) Para detectar uma ação de *click* do usuário em um elemento *Button*, pode ser utilizado o atributo **`android:onClick="método_da_activity"`** em tal elemento.

Sem medo de errar

Caro desenvolvedor, chegou a hora de finalizar o aplicativo para a empresa de logística. Na etapa anterior, você criou o recurso de *layout* para que o usuário possa interagir com o aplicativo. Nesta última etapa, você deverá acessar os elementos desse *layout* para recuperar os valores digitados pelo usuário. Também será necessário adicionar a lógica que realizará o cálculo da porcentagem do ICMS e exibirá o resultado para o usuário.

Confira os passos e as dicas a seguir para finalizar o aplicativo:

(i) Acesse a classe **`MainActivity.java`** e declare os objetos do *layout* que deseja acessar.

(ii) Instancie esses objetos no método **`onCreate()`** de sua *Activity*, utilizando o método **`findViewById(int id)`** e a Classe R.

(iii) Adicione o atributo **`android:onClick="calcular"`** no elemento `Button` que deverá realizar o cálculo.

(iv) Crie o método **`"public void calcular(View view){}"`** na *Activity* para que o botão possa responder à interação do usuário.

(v) entro do método **`"calcular()"`**, você deverá:

a) Recuperar os valores digitados por meio dos objetos declarados.

b) Converter o valor recuperado de *String* para *float*.

c) Verificar qual estado foi digitado para definir o respectivo valor da porcentagem. Consulte em algum site de pesquisa quais são as alíquotas de ICMS correspondentes para cada estado.

d) Realizar o cálculo e armazenar em uma nova variável *float*.

e) Utilizar o objeto **`TextView`** declarado para exibir o valor total. Nesse momento é importante fazer a conversão inversa de *float* para *String*.

Antes de iniciarmos a conversão dos tipos, observe que há diferenças entre *float* e *Float*:

(i) *float* é um tipo primitivo de valor que aceita casas decimais.

(ii) *Float* é uma classe java que contém um valor do tipo *float*, e essa classe nos permite acessar diversos métodos, por exemplo, métodos de conversão de *String* para *float*.

Para realizar a conversão de *String* para *float* e de *float* para *String*, será necessário utilizar as classes *String* e *Float*.

(i) Para converter de uma *String* para um tipo *float*, utilize o método **`Float.parseFloat(String textoDigitado)`**. Esse método retornará um valor *float*.

(ii) Para converter de um *float* para uma *String*, utilize o método **`String.valueOf(valor_float)`**. Esse método retornará uma *String* com o valor passado no parâmetro.

(iii) Há a opção de converter de um *float* para *String* utilizando a formatação de moeda. Utilize o método **`String.format("R$ %.2f", valor_float)`**. Esse método retornará uma *String* formatada no padrão "R$ 0,00".

Agora, analise o código e os comentários disponíveis na classe **`MainActivity`**.

```
9.    public class MainActivity extends
      AppCompatActivity {
10.
11.     private EditText mEditTextEstado;
12.     private EditText mEditTextValor;
13.     private TextView mTextViewPorcentagem;
14.     private TextView mTextViewTotal;
15.
16.     @Override
17.     protected void onCreate(Bundle
      savedInstanceState) {
18.
19.       super.onCreate(savedInstanceState);
20.       setContentView(R.layout.activity_main);
21.
22.       mEditTextEstado = findViewById(R.
```

```
id.editTextEstado);
23.      mEditTextValor = findViewById(R.
id.editTextValor);
24.      mTextViewPorcentagem = findViewById(R.
id.textViewPorcentagem);
25.      mTextViewTotal = findViewById(R.
id.textViewTotal);
26.    }
27.
28.    public void calcular(View view) {
29.      //Recuperamos o estado digitado
30.      String estado = mEditTextEstado.
getText().toString();
31.      //Recuperamos o valor digitado
32.      String valorString = mEditTextValor.
getText().toString();
33.      // Convertemos o valorString para tipo
float
34.      float valor = Float.
parseFloat(valorString);
35.      // Inicializamos uma variável float com o
valor da porcentagem
36.      float porcentagem = 0;
37.      // Verifica qual estado foi digitado para
definir o valor da porcentagem
38.      switch (estado) {
39.      // Neste momento você deverá criar case
para todos os estados
40.       case "RO":
41.         porcentagem = 17.5f;
42.         break;
43.       case "SP":
44.         porcentagem = 18;
45.         break;
46.    }
47.
48.    // Declara uma nova variável float com o
valor do cálculo
49.    float total = valor + (valor * porcentagem
```

```
      / 100);
50.
51.       // Exibi o resultado no TextView para o
      usuário
52.       mTextViewPorcentagem.setText(String.
      valueOf(porcentagem) + "%");
53.       mTextViewTotal.setText(String.format("R$
      %.2f", total));
54.     }
55.   }
```

Com a estrutura da classe conforme o código acima, sempre que o usuário clicar no botão "Calcular", será realizado e exibido o cálculo do ICMS correspondente ao estado e ao valor digitados pelo usuário.

Finalizamos aqui o desenvolvimento da lógica principal do aplicativo. Nada o impede de apresentar uma nova versão do aplicativo com funcionalidades mais precisas. Por exemplo, é possível adicionar, antes de realizar as conversões, uma estrutura condicional para verificar se o valor digitado da mercadoria é válido.

A sua colaboração com a equipe de desenvolvimento foi essencial para concluirmos a entrega deste projeto. O aplicativo para a Empresa de Logística já está disponível em sua primeira versão. Bom trabalho!

Avançando na prática

Cálculo para pagamento de conta

Descrição da situação-problema

Uma empresa que atua no ramo alimentício deseja disponibilizar um aplicativo ao seus cliente para agilizar o pagamento da conta. Você foi contratado por essa empresa para desenvolver um aplicativo Android que calcule o valor da taxa de serviço e o troco para o cliente. O aplicativo deve receber, como entrada, o valor da conta e o valor em papel moeda que o cliente usará para pagar tal conta. Quando o usuário pressionar o botão de calcular, deverá

ser aplicado 15% sobre o valor da conta, que corresponderá à taxa de serviço. O aplicativo também deverá calcular o novo valor total, correspondente ao valor da conta acrescido do valor da taxa de serviço. Por fim, deverá calcular o troco, que corresponderá ao valor em dinheiro subtraído do valor total e exibir todos os valores calculados para o usuário.

Resolução da situação-problema

Para iniciar o desenvolvimento do aplicativo solicitado pela empresa, você deverá criar um projeto no Android Studio com uma *Empty Activity*. Siga os passos a seguir para criar a estrutura necessária para o aplicativo solicitado:

(i) Defina a interface da **`MainActivity`** para que os clientes possam interagir com o aplicativo. Para realizar esse passo, acesse o arquivo *"activity_main.xml"* e acrescente os elementos que julgar necessários para criar uma interface intuitiva. Observe a sugestão de criação na Figura 1.15.

(ii) Para utilizar dois botões na horizontal, conforme exibido na Figura 1.15, você precisará utilizar a técnica de inserir um **`LinearLayout`** dentro de outro **`LinearLayout`**. Um elemento deste tipo deve conter o atributo **`android:orientation="vertical"`** e o outro elemento deve conter o atributo **`android:orientation="horizontal"`**.

(iii) Na interface sugerida, foram disponibilizados dois elementos Button dentro de um **`LinearLayout`**, com a orientação definida para horizontal. Adicione os atributos **`"android:onClick="limparDados"`** e **`"android:onClick="calcular"`** em cada elemento Button, respectivamente.

(iv) Após adicionar os atributos do passo anterior, será necessário criar dois métodos na classe **`MainActivity`** para acrescentar o comportamento de cada botão da interface.

(v) Também na classe **`MainActivity`**, é necessário declarar e instanciar os **`EditText e os TextView`** disponíveis na interface para que você possa interagir com esses elementos. Atenção! Para

que você possa acessá-los por meio do método **`findViewById(int id)`** e a Classe R, é necessário definir em cada elemento XML o atributo **`android:id="@+id/nome_do_elemento"`**.

(vi) Agora é hora de adicionar a lógica em cada método criado. Para o método **`calcular()`**: a) recupere os valores dos **`EditText`** em uma **`String`**; b) verifique se os valores não estão vazios; c) converta para um tipo float; d) realize os cálculos; d) converta novamente para String e exiba o resultado para o usuário. Para o método **`limparDados()`**: a) limpe os valores de cada **`EditText`**.

Figura 1.15 | Interface para cálculo de pagamento de conta

Fonte: elaborada pelo autor.

Código para MainActivity.java

Você poderá verificar uma opção de código-fonte para o problema acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u1/s3/codigo.pdf> ou por meio do QR Code.

Você poderá disponibilizar uma nova versão do aplicativo fornecendo novas funcionalidades, por exemplo, permitir que o usuário digite o valor da porcentagem da taxa de serviço. Ao finalizar os passos listados acima, o aplicativo estará disponível para a empresa que o contratou.

Faça valer a pena

1. Ao inicializar uma **`Activity`**, o sistema Android executa alguns métodos específicos correspondentes a essa **`Activity`**. O primeiro método a ser executado é o método **`onCreate()`**.
É recomendado, ao utilizar o método **`onCreate()`**:
(i) Definir uma interface para o usuário.
(ii) Inicializar os objetos que serão utilizados na **`Activity`**.
(iii) Inserir diretamente a lógica de algum botão disponível na interface.
(iv) Executar a lógica de **`onCreate()`** implementada na classe pai por meio do comando **`super.onCreate(savedInstanceState)`**.
(v) Recuperar os valores digitados pelo usuário no *layout*.

Selecione a alternativa correta sobre as recomendações ao utilizar o método **`onCreate()`** disponível em uma *Activity*.

a) Apenas as afirmações III e V estão corretas.
b) Apenas as afirmações I, II e III estão corretas.
c) Apenas as afirmações I, II, III e V estão corretas.
d) Apenas as afirmações I, II e IV estão corretas.
e) Apenas as afirmações II e IV estão corretas.

2. Para acessar um elemento **`EditText`** disponível em um recurso de *layout*, é necessário seguir três passos: (i) atribuir um ID para esse elemento no recurso de *layout*, por meio do atributo **`android:id="@+id/editText"`**; (ii) declarar um objeto **`EditText`** na **`Activity`**; (iii) instanciar este objeto no método **`onCreate()`** utilizando o método **`findViewById(R.id.editText)`**.
Assinale a alternativa que permite recuperar um número digitado pelo usuário em um **`EditText`**.

a) float numeroDigitado = Float.parseFloat(mEditText.getText().toString())
b) float numeroDigitado = String.valueOf(mEditText.getText().toString())
c) String numeroDigitado = Float.parseFloat(mEditText.getText().toString())

d) String numeroDigitado = mEditText.getText()
e) float numeroDigitado = mEditText.getText().toString()

3. Ao construir um recurso de *layout*, é necessário utilizar elementos em XML. Para que esses elementos atendam aos requisitos do aplicativo, deve-se utilizar atributos específicos para cada elemento. É possível detectar um toque de interação em um elemento **`Button`** disponível na interface por meio do atributo **`android:onClick="clicar_no_botao"`**. Contudo, não basta definir apenas esse atributo, pois é necessário criar um método na **`Activity`** que irá implementar o comportamento do aplicativo após o botão ser pressionado.

Assinale a alternativa que apresenta o método na **`Activity`** que responderá ao atributo **`android:onClick="clicar_no_botao"`**.

a) void clicar_no_botao () { }
b) void onClick (Button button) { }
c) public void clicar_no_botao (View view) { }
d) public void onClick(View view) { }
e) public void clicar_no_botao { }

Referências

ANDROID. **Android Studio:** O IDE oficial do Android. Disponível em: <https://developer.android.com/studio/index.html>. Acesso em: 2 mar. 2018.

ANDROID. **Atividades**. Disponível em: <https://developer.android.com/guide/components/activities.html?hl=pt-br>. Acesso em: 6 abr. 2018.

ANDROID. **Buttons**. Disponível em: <https://developer.android.com/guide/topics/ui/controls/button.html>. Acesso em: 26 mar. 2018.

ANDROID. **Configure sua compilação**. Disponível em: <https://developer.android.com/studio/build/index.html>. Acesso em: 3 mar. 2018.

ANDROID. **Criar e gerenciar dispositivos virtuais**. Disponível em: <https://developer.android.com/studio/run/managing-avds.html>. Acesso em: 2 mar. 2018

ANDROID. **Follow field naming conventions**. Disponível em: <https://source.android.com/setup/contribute/code-style#follow-field-naming-conventions>. Acesso em: 25 mar. 2018.

ANDROID. **Fundamentos de aplicativos:** componentes de aplicativo. Disponível em: <https://developer.android.com/guide/components/fundamentals.html?hl=pt-br>. Acesso em: 19 mar. 2018.

ANDROID. **Layout linear.** Disponível em: <https://developer.android.com/guide/topics/ui/layout/linear.html?hl=pt-br>. Acesso em: 19 mar. 2018.

ANDROID. **Recursos e APIs do Android 8.0:** Mudança de assinatura findViewById(). Disponível em: <https://developer.android.com/about/versions/oreo/android-8.0.html#fvbi-signature>. Acesso em: 2 abr. 2018.

ANDROID. **ScrollView**. Disponível em: <https://developer.android.com/reference/android/widget/ScrollView.html>. Acesso em: 19 mar. 2018.

CORPORATION, International Data. **Smartphone OS Market Share**. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>. Acesso em: 28 fev. 2018.

CORPORATION, Oracle. **Java SE Downloads**. Disponível em: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Acesso em: 1 mar. 2018.

DEITEL, Paul; DEITEL, Harvey; WALD, Alexander. **Android 6 para programadores:** uma abordagem baseada em aplicativos. 3. ed. Porto Alegre: Bookman, 2016.

TANENBAUM, Andrew S.; BOS, Herbert. **Sistemas Operacionais Modernos.** 4. ed. São Paulo: Pearson Education do Brasil, 2016.

W3SCHOOLS. XML **Regras de sintaxe**. Disponível em: <https://www.w3schools.com/xml/xml_syntax.asp>. Acesso em: 19 mar. 2018.

Armazenamento Key-Value e aplicações com Android™

Convite ao estudo

Caro programador, seja bem-vindo à segunda unidade de estudo da disciplina de Desenvolvimento para Dispositivos Móveis. Aqui vamos conhecer funcionalidades mais avançadas para desenvolvermos os nossos aplicativos. Os conteúdos nos permitirão desenvolver aplicativos com funcionalidades mais complexas e com recursos de armazenamento de dados.

O Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) é uma entidade privada sem fins lucrativos e o seu objetivo é apoiar os pequenos negócios em todo o território nacional. O SEBRAE sempre inova e oferece novos serviços para todos os microempreendedores.

O Microempreendedor Individual (MEI) possui diversos benefícios, por exemplo, a legalidade de emitir notas fiscais. Além dos benefícios, o MEI também possui suas obrigações, entre elas podemos listar a obrigatoriedade do envio da Declaração Anual de Faturamento do Simples Nacional (DASN), em que o microempreendedor deve informar o valor total de faturamento do ano anterior ao envio.

Cabe ao microempreendedor individual manter anotado em seu caderno de organização todo o seu faturamento organizado por ano, a fim de preencher e enviar a DASN. Vamos exemplificar da seguinte maneira: durante o ano de 2018, ele deve fazer anotações com os valores de seu faturamento para que, em 2019, some todos os lançamentos do ano anterior,

chegue a um valor final e, então, preencha a DASN com este valor e a envie.

Caro aluno, diante do disposto, o SEBRAE publicou um edital para realizar um processo de licitação, que é um procedimento para contratar empresas com as propostas mais vantajosas. O objeto do edital publicado é a contratação de empresa para desenvolver um aplicativo Android™ para auxiliar a obrigatoriedade imposta a todos os microempreendedores individuais.

Você, programador, também sócio de uma empresa de desenvolvimento de aplicativos móveis, apresentou a seguinte proposta ao SEBRAE: desenvolver um aplicativo Android para armazenar os lançamentos do MEI, que agrupará em um único local todos os lançamentos de faturamento para agilizar o preenchimento da DASN. Portanto, o microempreendedor não precisará mais consultar o seu caderno de organização para realizar a soma de todos os lançamentos, pois o aplicativo exibirá o valor total organizado por ano.

Após análise das propostas pelo SEBRAE, a de sua empresa foi declarada vencedora e agora é hora de iniciar os trabalhos.

Como programador, você definiu três fases de desenvolvimento do aplicativo: para iniciar, você construirá o layout que os usuários vão utilizar para interagir com o aplicativo. Em seguida, deverá adicionar a lógica para que as informações fornecidas pelos usuários sejam armazenadas no aplicativo e, por fim, você disponibilizará uma nova versão do aplicativo para que o microempreendedor individual possa personalizá-lo.

Nesta unidade você será desafiado a utilizar técnicas avançadas para a criação do segundo aplicativo. Não perca o foco dos estudos, pois este momento fará a diferença no futuro.

Seção 2.1

Desenvolvendo UI com *ConstraintLayout*

Diálogo aberto

Caro programador, seja bem-vindo à primeira seção da unidade Desenvolvendo UI com *ConstraintLayout*. Você já se perguntou como um aplicativo Android reconhece quando o usuário toca na tela? Além disso, já pensou em como construir layouts responsivos para que os aplicativos possam se adaptar em diferentes tamanhos de telas?

Nesta seção serão apresentados os conceitos de *Listeners*, que são interfaces capazes de detectar uma interação entre o usuário e o layout. Para que seja possível construir um layout responsivo, também será apresentado nesta seção o elemento *ConstraintLayout*.

Caro aluno, você está à frente de desenvolver um projeto que ajudará milhões de microempreendedores. Nesta primeira etapa, será criado o layout do aplicativo, respeitando o fluxo de processamento que determinará a melhor forma de interagir com o usuário.

O objetivo do aplicativo é armazenar o valor do faturamento anual do microempreendedor, portanto, será necessário desenvolver um layout que permita adicionar ou excluir valores e que também exiba o valor total do faturamento organizado por ano.

O layout deve conter um elemento para que o usuário possa selecionar um ano e outro elemento para exibir o valor total de faturamento correspondente ao ano selecionado. Além disso, deve disponibilizar para o usuário outros elementos com as opções de incluir ou excluir valores referentes ao ano selecionado.

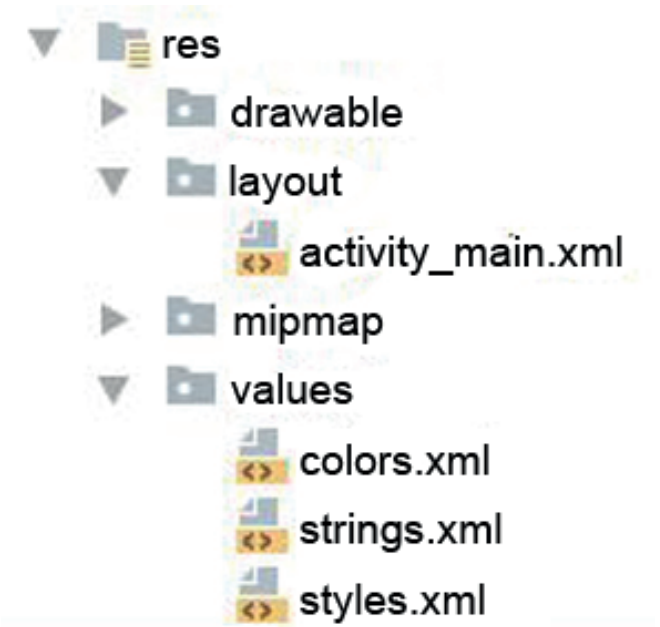
Qual elemento você escolherá para o microempreendedor selecionar o ano que deseja lançar um valor? Como será verificado se o usuário deseja adicionar ou excluir um valor? A soma total do faturamento referente ao ano selecionado ficará em destaque na *Activity*? Existem diversos elementos disponíveis para a elaboração de um aplicativo Android, contudo você deverá definir quais utilizar.

Cada detalhe neste momento fará a diferença. Dedique um tempo especial e explore cada conteúdo apresentado nesta seção para desenvolver aplicativos com mais qualidade.

Não pode faltar

O Android organiza os recursos que não são código-fonte java na pasta "res", abreviatura para *resources* (recursos). Os layouts utilizados pelos usuários para interagir com o aplicativo são conhecidos como recursos de layouts e estão armazenados na subpasta layout, conforme ilustrado na Figura 2.1.

Figura 2.1 | Pasta de recursos



Fonte: captura de tela do Android Studio.

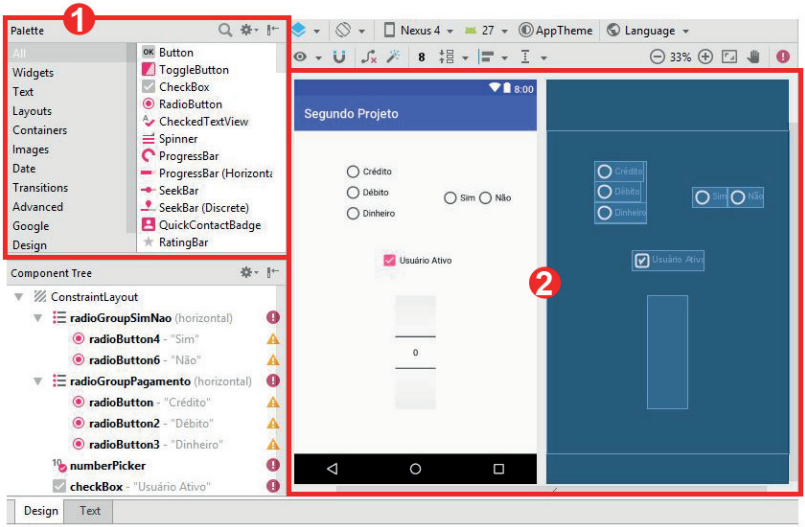
As demais subpastas apresentam outros tipos de recursos. Analise as definições abaixo:

- A subpasta *drawable* armazena todas as imagens utilizadas no aplicativo.
- A subpasta *mipmap* armazena ícones de inicialização do aplicativo.
- A subpasta *values* armazena diferentes tipos de recursos, entre eles valores de cores (*colors.xml*), valores de *strings* (*strings.xml)* e estilos (*styles.xml*).

Até o momento, construímos os layouts utilizando os elementos *ScrollView* e *LinearLayout*. Além deles, o Android nos oferece outros recursos para construirmos os layouts. Nesta seção vamos explorar o *ConstraintLayout*. Segundo Android (2018), através deste elemento é possível criar um recurso de layout com uma interface complexa, sem a necessidade de inserir os elementos de forma hierárquica, conforme utilizado no *LinearLayout*. Toda a construção da interface poderá ser modelada através do *LayoutEditor*. Ao abrirmos um arquivo disponível na pasta de recursos de layout, o *LayoutEditor* será carregado, conforme a Figura 2.2. O *LayoutEditor* é uma ferramenta disponibilizada pela IDE Android Studio que permite editar e visualizar os layouts em tempo de construção, sem a necessidade de compilar e instalar o aplicativo em um dispositivo físico ou emulador. Sua interface também permite criar layouts simplesmente arrastando e soltando os elementos, sem editar códigos em XML.

Vamos criar um projeto no Android Studio com uma *Empty Activity* (atividade vazia), para estudarmos este modo de criação de layout. Após criar o projeto, abra o arquivo "activity_main.xml" para que possamos modificá-lo. Na Figura 2.2 é iniciada a criação de um recurso de layout utilizando o elemento *ConstraintLayout*.

Figura 2.2 | *LayoutEditor*



Fonte: captura de tela do Android Studio, elaborada pelo autor.

1: Conforme estudamos na Unidade 1, a janela *Palette* dispõe os elementos para a construção do layout. É possível arrastar os elementos disponíveis nesta janela para a área de *Design Editor*.

2: Esta é a área de *Design Editor* que possui dois modos de visualizações: a) modo *Design* (lado direito), que exibe uma visão colorida do layout e b) modo *Blueprint* (lado esquerdo), que exibe apenas o contorno de cada elemento.

Antes de compreendermos como o *ConstraintLayout* organiza a exibição dos seus elementos, vamos analisá-los conforme dispostos na Figura 2.2.

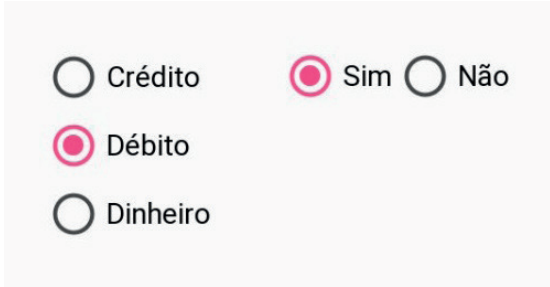
Na parte superior do layout, inserimos *RadioButton* e *RadioGroup*. RadioButton é um tipo de elemento que indica uma opção para que o usuário possa selecionar. Por padrão, ele poderá escolher apenas um *RadioButton* disponível em cada lista de opções. *RadioGroup* é um tipo de elemento que permite agrupar vários *RadioButtons* em uma lista de opções. Ao utilizar um *RadioGroup* para criar grupos de *RadioButtons*, será permitido selecionar apenas um *RadioButton* para cada grupo criado.

Reflita

Imagine que você esteja desenvolvendo um layout que exibirá para o usuário uma lista com diversas opções de configurações de um aplicativo, porém o usuário poderá selecionar apenas uma opção desta lista. Será possível utilizar o elemento *RadioButton* para representar cada opção desta lista. Qual será o comportamento se criarmos este layout com vários *RadioButtons*, porém não os inserirmos no elemento *RadioGroup*?

Observe na Figura 2.3 os dois elementos *RadioGroups*. Cada um possui os seus elementos-filhos *RadioButtons*. Durante a execução do aplicativo, será permitido selecionar apenas um *RadioButton* em cada *RadioGroup* criado.

Figura 2.3 | *RadioGroup* e *RadioButton*



Fonte: elaborada pelo autor.

O próximo elemento inserido é um *CheckBox*, conforme a Figura 2.4.

Figura 2.4 | Elemento Checkbox

Fonte: elaborada pelo autor.

Diferentemente do elemento anterior, o usuário poderá selecionar vários *CheckBox* disponíveis em um layout, sem a necessidade de criar grupos. O atributo *android:checked="true"* permite inicializar o elemento já selecionado. Se alterar o valor do atributo para *"false"*, inicializará o elemento desmarcado. Existem duas formas para modificar o atributo: a) através do modo XML, conforme estudamos na unidade anterior ou b) através da janela *Attributes*.

A Figura 2.5 a seguir ilustra o elemento *NumberPicker*.

Figura 2.5 | Elemento *NumberPicker*

Fonte: elaborada pelo autor.

Este elemento permite ao usuário selecionar um número dentro de um intervalo pré-definido. Para que o *NumberPicker* funcione de maneira adequada, será necessário instanciar este objeto na MainActivity.java e definir os métodos *setMinValue(valor)* e *setMaxValue(valor)* para este objeto. Os métodos citados definem o intervalo de números disponibilizados para o usuário selecionar.

```
10.  @Override
11.  protected void onCreate(Bundle
     savedInstanceState) {
12.     super.onCreate(savedInstanceState);
13.     setContentView(R.layout.activity_main);
14.
15.    // Declaramos um objeto local, ou seja, poderá
     ser utilizado apenas dentro do método onCreate()
16.     NumberPicker mNumberPicker = findViewById(R.
     id.numberPicker);
17.
18.     // Definimos o intervalo de números entre 0
     e 10
19.     mNumberPicker.setMinValue(0);
20.     mNumberPicker.setMaxValue(10);
21.  }
```

Atenção

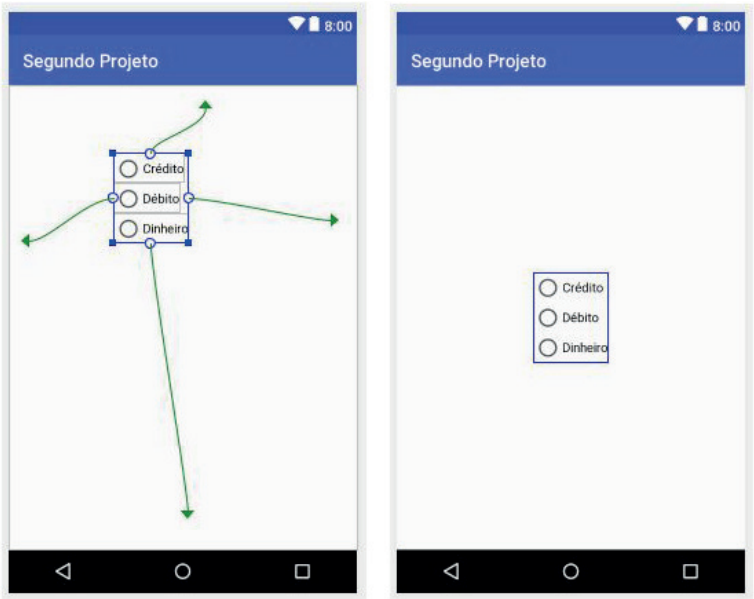
Ao construir um layout, não se esqueça de definir para cada elemento o atributo *android:id="@+id/nome_do_elemento"*. Através deste ID será possível instanciar o elemento na classe *MainActivity.java*.

Agora vamos compreender como o *ConstraintLayout* organiza os elementos *Views*. Diferentemente do *LinearLayout*, conforme estudado na Unidade 1, não há a necessidade de inserirmos os elementos de forma hierárquica. O *ConstraintLayout* realiza uma conexão entre todos os elementos disponíveis na interface.

Exemplificando

Vamos realizar a conexão do primeiro elemento no layout. Selecione o *RadioGroup* que exibe as opções de pagamento e arraste os círculos exibidos em sua borda para as margens do layout. Observe na Figura 2.6 como esta conexão é realizada e qual será o novo posicionamento do *RadioGroup*. Os demais elementos estão ocultos na Figura 2.6 apenas para que seja possível obter uma melhor visualização.

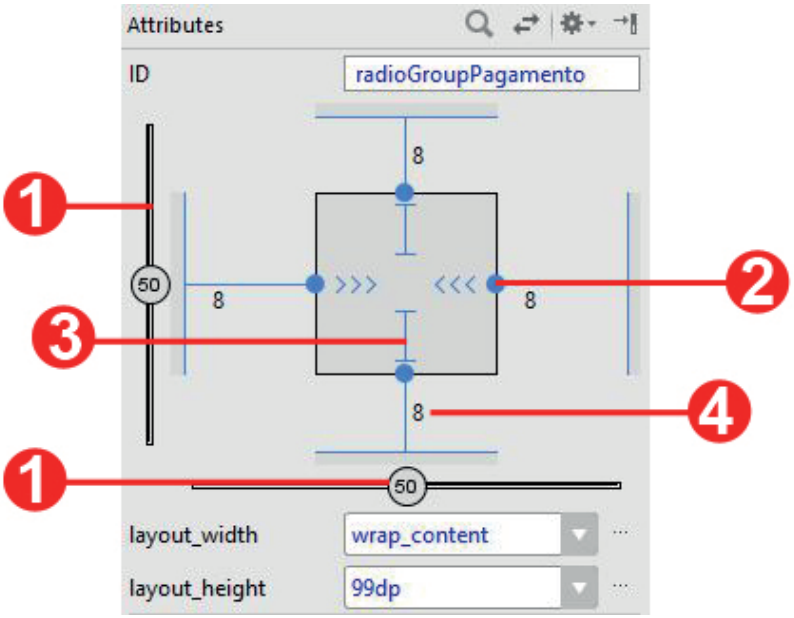
Figura 2.6 | Posicionamento do *RadioGroup*



Fonte: captura de tela do Android Studio, elaborada pelo autor

Observe na Figura 2.7 abaixo as opções disponíveis na janela *Attributes*, para que novos ajustes possam ser realizados no posicionamento de um elemento.

Figura 2.7 | Janela *Attributes*



Fonte: captura de tela do Android Studio, elaborada pelo autor.

1: *Constraint Bias* são barras que permitem realizar ajustes nos elementos vertical ou horizontalmente. Essas barras são ativadas sempre que você realizar a conexão dos dois lados do elemento, ou seja, os lados direito e esquerdo para posicionar horizontalmente ou os lados superior e inferior para posicionar verticalmente. Se você conectar apenas um lado na posição vertical ou horizontal, a *Constraint Bias* não ficará disponível para ajustes.

2: Exclusão de Conexão permite remover uma conexão inserida ao elemento.

3: Modo de altura/comprimento permite alterar o tamanho do elemento. Ao utilizar o *ConstraintLayout*, são disponibilizados três valores: i) *fixed* que permite determinar um tamanho fixo para o elemento; ii) *wrap_content* que define o tamanho do elemento de acordo com o tamanho de seu conteúdo; iii) *match_constraint* que expande o tamanho do elemento o quanto for preciso para encaixar no respectivo tamanho de cada lado da conexão.

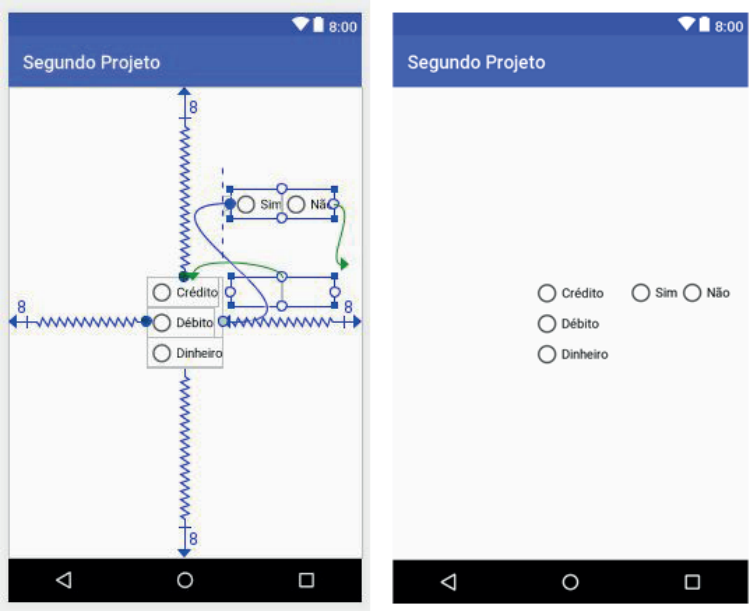
4: Margens permite definir a distância entre os elementos conectados.

Exemplificando

Vamos exemplificar a conexão entre dois elementos do layout. Selecione o *RadioGroup* com as opções "Sim/Não" e arraste os círculos exibidos em sua borda para os círculos exibidos na borda do outro *RadioGroup*.

Observe na Figura 2.8 como as conexões são realizadas e o novo posicionamento dos elementos.

Figura 2.8 | Posicionamento entre elementos



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Experimente utilizar na janela *Attributes* a barra *Constraint Bias* para posicionar o *RadioGroup* com as opções de pagamento na parte superior do layout. Observe que o outro *RadioGroup* também será reposicionado automaticamente, pois estão conectados.

Com as conexões apresentadas e os ajustes disponíveis na janela *Attributes*, é possível conectar e posicionar todos os elementos em um *ConstraintLayout*.

Pesquise mais

Consulte mais detalhes sobre as ferramentas disponíveis no *LayoutEditor* no site oficial de desenvolvimento Android, disponível em: <https://developer.android.com/studio/write/layout-editor.html?hl=pt-br>. Acesso em: 3 jul. 2018.

No Android também há diversos caminhos para detectarmos a interação entre o usuário e os elementos *Views* de um layout. Todos os elementos *Views* possuem "ouvintes" capazes de detectar a interação entre o usuário e o elemento. Neste momento vamos trabalhar o conceito de *Listener* e interface. O *Listener* é uma interface que age como uma camada que detecta a ação do usuário e executa um comportamento dentro do aplicativo. Por exemplo, é possível "ouvir" quando um usuário clica em um botão através da interface *OnClickListener*. Também é possível ouvir quando um valor de um *NumberPicker* é alterado através da interface *OnValueChangeListener*.

Assimile

Sempre que um *Listener* é criado, um método correspondente a ele deverá ser sobreposto, pois este método é executado quando uma ação é "ouvida" pelo *Listener*. Dentro do método sobreposto deverá ser adicionado o comportamento do aplicativo. Observe no Quadro 2.1 algumas interfaces de *Listener* e os métodos correspondentes que deverão ser sobrepostos.

Quadro 2.1 | *Listeners*

Interface	Método	Descrição
OnClickListener	onClick()	Detecta interação de toque
OnLongClickListener	onLongClick()	Detecta interação de toque prolongado

Interface	Método	Descrição
OnValueChangeListener	onValueChange()	Detecta quando um valor é alterado
OnCheckedChangeListener	onCheckedChanged()	Detecta quando uma opção é selecionada em um RadioGroup

Fonte: elaborado pelo autor.

Para "ouvir" uma interação, deve-se criar uma instância do *Listener* que se deseja ouvir na classe *MainActivity.java*. Observe o código abaixo para que possamos analisá-lo.

```
1. public class MainActivity extends AppCompatActivity {
2.     // Criando uma classe anônima para o Listener
3.     private NumberPicker.OnValueChangeListener valorAlteradoListener = new NumberPicker.OnValueChangeListener() {
4.         @Override
5.         public void onValueChange(NumberPicker numberPicker, int oldvalue, int newValue) {
6.     // Adicione a lógica aqui
7.     }
8. };
9.     @Override
10.     protected void onCreate(Bundle savedInstanceState) {
11. super.onCreate(savedInstanceState);
12. setContentView(R.layout.activity_main);
13.     // Instaciamos o objeto NumberPicker
```

```
14. NumberPicker mNumberPicker = findViewById(R.
    id.numberPicker);
15.  // Definimos o intervalo de números entre 0 e 10
16.  mNumberPicker.setMinValue(0);
17.  mNumberPicker.setMaxValue(10);
18. // Associamos o "Ouvinte" com o NumberPicker
19. mNumberPicker.setOnValueChangedListener(valorAl
    teradoListener);
20.  }
21. }
```

Linha 3: criamos uma classe anônima para a interface "ouvinte" através dos passos:

i) Defina o modificador de acesso do objeto para *private*.

ii) Declare o tipo do *Listener*. Utilizamos *OnValueChangeListener*.

iii) Defina o nome do objeto para *valorAlteradoListener*.

iv) Utilize a palavra-chave new para indicar uma nova instância.

v) Utilize as teclas de atalho CTRL + SHIT + ESPAÇO para abrir a caixa de sugestão do Android Studio. Selecione a opção *OnValueChangeListener* para que o Android Studio preencha o restante do código.

Linha 4: observe que existe a anotação *@Override* para indicar que o método *onValueChange()* está sendo sobreposto.

Linha 5: este método recebe três parâmetros. O primeiro parâmetro é um objeto *NumberPicker* que corresponde ao *NumberPicker* que está sendo interagido. O segundo parâmetro é um valor int que corresponde ao valor antigo do *NumberPicker*. O terceiro parâmetro também é um valor int que corresponde ao novo valor selecionado.

Linha 19: associe o *Listener* criado com o *NumberPicker* através do método *setOnValueChangedListener()*.

Com o código apresentado acima, é possível desenvolver um comportamento específico para o aplicativo sempre que um valor for alterado no elemento *NumberPicker*.

Pesquise mais

Você pode conhecer mais detalhes sobre as Escutas de Evento no site oficial de desenvolvimento do Android, disponível em: <https://developer.android.com/guide/topics/ui/ui-events.html?hl=pt-br>. Acesso em: 3 jul. 2018. Nesta página são apresentados mais exemplos sobre como definir um *Listener*. Compare-os para ampliar as possibilidades ao desenvolver um aplicativo Android.

Nesta seção trabalhamos com o *ConstraintLayout*, um elemento do tipo layout que o permite criar um recurso através do modo *Design* no *LayoutEditor*. Também aplicamos uma nova técnica para detectarmos uma interação entre o usuário e o aplicativo através das interfaces *Listeners*.

Continue focado nos estudos, pois serão apresentadas novas funcionalidades para integrarem os nossos aplicativos Android.

Sem medo de errar

Caro programador, a sua empresa foi contratada pelo SEBRAE para desenvolver um aplicativo Android, a fim de registrar o faturamento anual dos microempreendedores individuais. O seu aplicativo deverá disponibilizar um layout que permitirá ao usuário:

i) Selecionar um ano para lançamento e consulta de faturamento.

ii) Disponibilizar opções para adicionar ou excluir um valor.

iii) Registrar um valor de faturamento referente ao ano selecionado.

iv) Consultar o valor total de faturamento referente ao ano selecionado.

Para iniciarmos a primeira etapa de desenvolvimento do aplicativo MEI, crie um projeto no Android Studio com uma *Empty Activity*. Acesse o recurso de layout "activity_main.xml" para modificarmos o layout de interação entre o usuário e o aplicativo.

Para que o aplicativo atenda aos requisitos propostos, observe na Figura 2.9 quais elementos são utilizados e como estão dispostos no layout.

Figura 2.9 | Layout para o aplicativo MEI

Fonte: captura de tela do Android Studio, elaborada pelo autor.

Analise os passos abaixo para a construção do layout conforme Figura 2.9:

i) Utilize um *ConstraintLayout*.

ii) Arraste os seguintes elementos para a área de layout: a) *NumberPicker* para selecionar o ano; b) *RadioGroup* com dois elementos-filhos *RadioButtons* para as opções "Adicionar" ou "Excluir"; c) *EditText* para digitar o valor; d) *TextView* para exibir o total do faturamento; e) *Button* para o armazenamento do valor.

iii) Ao utilizar o *ConstraintLayout*, será necessário conectar todos os elementos, observe o exemplo de conexão na figura 2.9 e repita este passo para todos os elementos.

Após finalizar a construção do layout, acesse a classe *MainActivity*.java e faça as seguintes inclusões:

```
1. public class MainActivity extends AppCompatActivity {
2.
3. private NumberPicker numberPicker;
4. private NumberPicker.OnValueChangeListener valorAlteradoListener = new NumberPicker.OnValueChangeListener() {
5.
6. @Override
7. public void onValueChange(NumberPicker numberPicker, int valorAntigo, int valorAtual) {
8. // Adicione a lógica para exibir o valor do ano selecionado
9. }
10. };
11.
12. @Override
13. protectedvoid onCreate(Bundle savedInstanceState) {
14. super.onCreate(savedInstanceState);
15. setContentView(R.layout.activity_main);
16. numberPicker = findViewById(R.id.numberPicker);
17.
18. // Define o intervalo dos anos entre 2015 a 2030
19. numberPicker.setMinValue(2015);
20. numberPicker.setMaxValue(2030);
21.
22. // Registra o Listener para alteração de valores no NumberPicker
23. numberPicker.setOnValueChangedListener(valorAlteradoListener);
24. }
25. }
```

Linhas 3 e 16: declare e instancie o objeto *NumberPicker*.

Linhas 19 e 20: defina o intervalo de anos que será permitido selecionar no *NumberPicker*.

Linha 4: crie um objeto que "ouça" a interação entre o usuário e o *NumberPicker*. É necessário "ouvir" esta interação para que seja exibido o valor do faturamento correspondente ao ano selecionado. Neste momento não será necessário desenvolver a lógica para exibir o valor do ano selecionado, pois estamos apenas preparando a estrutura da classe *MainActivity.java* antes de iniciarmos a próxima etapa do projeto.

Linha 23: defina o Listener para o NumberPicker.

Finalizamos aqui a primeira etapa do desenvolvimento para o aplicativo Android MEI. Utilizamos novos elementos de desenvolvimento de layout e também novas técnicas para detectarmos as interações entre o usuário e os elementos de um recurso de layout.

Avançando na prática

Detalhes de Produto

Descrição da situação-problema

Uma empresa de moda especializada em atacado e varejo deseja contratar uma equipe de programadores para desenvolver um aplicativo para os seus clientes. O aplicativo disponibilizará um catálogo com todos os seus produtos. A empresa exige a exibição da foto do produto, título, descrição, tamanhos disponíveis e preço.

Você é integrante da equipe contratada e lhe foi delegada a tarefa de construir o layout que mostrará os detalhes de cada produto.

Resolução da situação-problema

Para executar a tarefa que lhe foi delegada, crie um projeto no Android Studio com uma *EmptyActivity*. Acesse o recurso "activity_main.xml" e distribua os elementos no layout, de modo que sejam apresentados intuitivamente os detalhes sobre o produto. Observe na Figura 2.10 a sugestão de layout criada para a empresa que o contratou.

Figura 2.10 | Layout para detalhes de produto

Fonte: captura de tela do Android Studio, elaborada pelo autor.

Para a criação do layout sugerido, foram utilizados um *ConstraintLayout*, um *ImageView* e quatro *TextView*. Arraste-os da janela *Palette* para a área de *Design* e faça as conexões entre eles, para que possam ser posicionados. Também utilize a janela *Attributes* para personalizar tamanho, cor e margens de cada elemento *View*. Se desejar, faça modificações para apresentar uma interface mais clara para o usuário. Bom trabalho!

Faça valer a pena

1. Segundo Android (2018), disponível em: <https://developer.android.com/training/constraint-layout/index.html?hl=pt-br>, acesso em: 3 jul. 2018, a API do *ConstraintLayout* e o *LayoutEditor* disponível no Android Studio foram construídos especialmente um para o outro. Como resultado, é possível usufruir do poder do elemento *ConstraintLayout* diretamente

no modo *Design* disponível no *LayoutEditor*, ou seja, é possível criar um recurso de layout apenas arrastando e soltando os *Views*, sem necessidade de editar códigos em XML.

Quais os passos necessários para ajustar vertical ou horizontalmente um elemento-filho de *ConstraintLayout*?

a) i) Selecione o elemento; ii) Arraste-o vertical ou horizontalmente.
b) i) Selecione dois elementos; ii) Conecte-os na borda superior; iii) Arraste-os vertical ou horizontalmente.
c) i) Selecione o elemento; ii) Utilize a opção de match_constraint disponível na janela *Attributes*.
d) i) Selecione o elemento; ii) Utilize a opção *margin* disponível na janela *Attributes*.
e) i) Selecione o elemento; ii) Conecte os lados superior e inferior para ajustar verticalmente ou os lados direito e esquerdo para ajustar horizontalmente; iii) Utilize a opção *ConstraintBias* disponível na janela *Attributes*.

2. Um *Listener* é uma interface responsável por detectar uma interação entre o usuário e um elemento *View*. O *Listener* contém um método que deve ser sobreposto e é responsável pelo comportamento que o aplicativo executará após a interação do usuário. O *Listener* também deverá ser associado ao elemento *View* através do método correspondente *set...Listener()*.

Qual é o método responsável por receber como parâmetro uma instância de um *Listener* para detectar uma interação de toque entre o usuário e um objeto *Button*?

a) setOnCheckedChangeListener(buttonListener)
b) setOnLongClickListener(buttonListener)
c) setOnValueChangeListener(buttonListener)
d) setOnClickListener(buttonListener)
e) setOnClick(buttonListener)

3. Um *ConstraintLayout* permite criar um recurso de layout responsivo, ou seja, todos os elementos *Views* são adaptados de acordo com o tamanho da tela em que o aplicativo está sendo executado. Para definir a posição de um elemento em um *ConstraintLayout*, você deve adicionar pelo menos uma conexão horizontal e uma vertical. Cada conexão representa uma posição e um alinhamento referente ao elemento em que está conectado. Como

resultado, ao movimentar um elemento, os demais elementos conectados a ele também serão reposicionados.

Quando o aplicativo for executado, qual será o posicionamento dos elementos *Views*, caso não sejam conectados entre si em um *ConstraintLayout*?

a) Não serão exibidos no layout.
b) Serão reposicionados para a borda superior do layout.
c) Serão reposicionados para a borda esquerda do layout.
d) Serão reposicionados para as bordas superior e esquerda do layout.
e) Serão reposicionados nas bordas inferior e esquerda do layout.

Seção 2.2

Armazenamento *Key-Value*

Diálogo aberto

Olá programador, seja bem-vindo à segunda seção da unidade. Na seção anterior nós desenvolvemos um layout utilizando *ConstraintLayout* e adicionamos os *Listeners* para detectar as interações entre o usuário e os elementos do layout. Vamos iniciar esta seção com a seguinte cena: imagine que você esteja utilizando um aplicativo de lembretes para armazenar as tarefas que realizará durante o dia. Após adicionar as tarefas ao aplicativo, você o fechou. Alguns minutos depois, lembrou-se de adicionar mais algumas tarefas, então retornou ao aplicativo. Contudo, percebeu que os seus dados não estão armazenados. Com certeza, isso é uma situação frustrante com o aplicativo. Para corrigir a questão apresentada, o Android oferece soluções para trabalhar com armazenamento de dados.

Você chegou à segunda fase de desenvolvimento do aplicativo para o SEBRAE e deseja agilizar a vida do microempreendedor para que ele não precise realizar todos os lançamentos todas as vezes que quiser saber o faturamento anual. Você deve garantir que a soma de todos os lançamentos fique armazenada no dispositivo móvel. Assim, quando o microempreendedor retornar ao aplicativo para fazer um novo lançamento, os valores já lançados estarão disponíveis para serem acrescidos com os novos lançamentos daquele ano.

Dando continuidade no aplicativo para o MEI, é hora de definir a estrutura em que os dados serão armazenados. Acesse o projeto criado na seção anterior e acrescente na classe *MainActivity.java* os métodos necessários para armazenar, excluir e consultar os valores de faturamento do MEI. Defina como serão recuperados os valores já lançados e como serão acrescidos os novos valores e, ao iniciar o aplicativo, exiba automaticamente o saldo atual do faturamento para o usuário.

Para concluirmos a proposta apresentada para o segundo aplicativo, trabalharemos com armazenamento de dados nesta

seção. Serão apresentados conceitos de armazenamento de dados com estrutura *key-value* no Android e técnicas para salvar e recuperar os valores armazenados. Continue focado nos estudos e aproveite o conteúdo sobre armazenamento de dados para desenvolver aplicativos que ofereçam mais funcionalidades para o usuário.

Não pode faltar

Nesta seção, vamos trabalhar com os conceitos e as técnicas de armazenamentos de dados. Segundo Android (2018), o Android oferece uma lista de opções para que os dados dos aplicativos fiquem salvos de maneira persistente, ou seja, é possível salvar os dados de um aplicativo Android e recuperá-los mesmo após o aplicativo ser encerrado.

Preferências Compartilhadas é a primeira opção disponível para armazenamento de dados no Android, através das quais vamos armazenar dados primitivos em uma estrutura *key-value* (chave-valor).

Exemplificando

Para entendermos como uma estrutura *key-value* funciona, vamos analisar o exemplo a seguir:

```
{
  "perfil" : "administrador",
  "valor" : 75.4,
  "quantidade" : 28,
  "privilegio" : true
}
```

Temos quatro registros com chaves do tipo *String* e valores primitivos associados a cada chave.

Neste exemplo, as chaves são "perfil", "valor", "quantidade" e "privilegio". Atenção, as chaves obrigatoriamente devem ser do tipo *String*.

Os valores associados a essas chaves podem ser do tipo *String*, *float*, *int boolean* ou *Set<String>*. Devemos utilizar o nome da chave para acessar o valor correspondente, por exemplo, a chave "perfil" possui o valor "administrador".

Ao armazenarmos dados com as Preferências Compartilhadas no Android, devemos utilizar a interface *SharedPreferences*. Segundo Android (2018), um objeto *SharedPreferences* organiza os dados da seguinte maneira:

i) *SharedPreferences* armazena os dados em um arquivo com estrutura *key-value*.

ii) Para utilizarmos o *SharedPreferences* devemos fornecer um nome para o arquivo que terá os dados armazenados.

iii) É possível possuir mais de um arquivo com dados armazenados pelo *SharedPreferences*, basta apenas fornecer nomes diferentes para os arquivos.

iv) Um objeto *SharedPreferences* nos fornece métodos para escrevermos e lermos os valores armazenados.

Para criar ou acessar um arquivo de *SharedPreferences*, você deverá chamar um desses métodos:

i) getSharedPreferences(String nomeDoArquivo, int modoDeAcesso)

a) Recebe dois parâmetros: o primeiro refere-se ao nome do arquivo que será utilizado para gravar os dados; o segundo refere-se ao modo de operação para gravar e/ou acessar o arquivo.

b) Uma característica importante deste método é que outra *Activity* do aplicativo terá acesso aos dados gravados neste arquivo.

ii) getPreferences(int modoDeAcesso)

a) Recebe apenas o parâmetro de modo de operação para gravar e/ou acessar o arquivo.

b) Neste método não é necessário fornecer o nome do arquivo para gravação, pois será fornecido o nome da *Activity* em que o método está sendo chamado. Este método é utilizado caso precise apenas de um arquivo de preferência para a *Activity*.

Assimile

O parâmetro Modo de Operação para gravar e/ou acessar o arquivo define como o *SharedPreferences* irá tratar o arquivo. Nesta seção conheceremos dois modos:

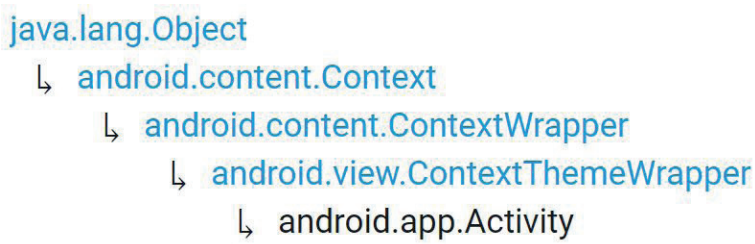
i) MODE_PRIVATE é o modo mais utilizado. Ao usá-lo, o arquivo de preferências criado estará disponível apenas para o aplicativo que o criou.

ii) MODE_APPEND verifica se o arquivo já existe e, se existir, adiciona novos dados no final do arquivo.

Mas afinal, onde estão localizados esses modos para que possamos utilizar? Vamos aprofundar nossos estudos para entendermos como uma classe *Activity* é construída.

Em Java, a classe *Object* é a classe-pai para todas as demais classes que trabalhamos. Observe na Figura 2.11 a hierarquia da classe *Activity*.

Figura 2.11 | Hierarquia da classe Activity



Fonte: <https://developer.android.com/reference/android/app/Activity>. Acesso em: 4 jul. 2018.

Observe que a classe *Activity* herda a classe *Context*, que herda *Object*. Context possui informações sobre o ambiente do aplicativo. Também são disponibilizadas constantes na classe Context para que os aplicativos possam utilizá-las e entre elas estão as constantes MODE_PRIVATE e MODE_APPEND.

Concluímos que uma *Activity* é filha da classe *Context* que possui as constantes *MODE_PRIVATE e MODE_APPEND*.

Ao chamar os métodos citados, será retornado um objeto *SharedPreferences*. Analise o código abaixo:

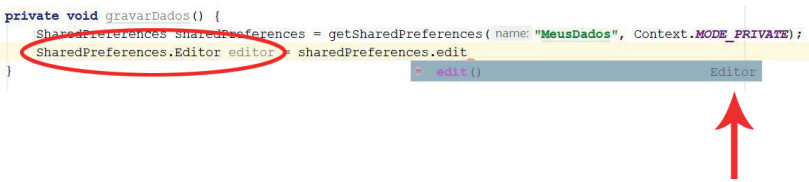
```
18. private void gravarDados(){
19. SharedPreferences sharedPreferences = getShared
    Preferences("MeusDados", Context.MODE_PRIVATE);
20. }
```

Linha 18: criamos um método na *Activity* para gravar os dados em um arquivo de preferência.

Linha 19: criamos um objeto *SharedPreferences* com o nome de arquivo definido como "MeusDados" e no modo privado.

Para que possamos salvar dados, devemos chamar o método *edit()* que nos retornará um *Editor*. Observe na Figura 2.12 o objeto de retorno do método *edit()*.

Figura 2.12 | Criação do objeto SharedPreferences.Editor



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Através do objeto *editor* criado é possível salvar os dados. Para cada tipo de dado que se deseja salvar, há um método correspondente. Observe no Quadro 2.2 os métodos disponíveis.

Quadro 2.2 | Métodos para armazenar valores no *SharedPreferences*

Método	Descrição
putString(*String* chave, *String* valor)	Permite salvar um valor do tipo *String*
putFloat(*String* chave, *float* valor)	Permite salvar um valor do tipo *float*
putLong(*String* chave, *long* valor)	Permite salvar um valor do tipo *long*
putInt(*String* chave, *int* valor)	Permite salvar um valor do tipo *int*
putBoolean(*String* chave, *boolean* valor)	Permite salvar um valor do tipo *boolean*

Método	Descrição
putStringSet(*String* chave, *Set<String>* valor)	Permite salvar um valor do tipo *Set<String>*

Fonte: elaborado pelo autor.

Reflita

Um objeto *SharedPreferences*.Editor nos oferece métodos específicos para armazenarmos cada tipo de valor. Durante a execução de um aplicativo, podemos chamar por estes métodos e fornecer como parâmetro a mesma chave. Qual será o comportamento do objeto *SharedPreferences.Editor* caso tente armazenar um valor referente a uma chave existente?

Após os dados serem direcionados para o objeto *editor*, deve-se então realmente efetuar a gravação dos valores no arquivo através do método *apply()*.

Exemplificando

Dando continuidade no exemplo apresentado no início desta seção, observe o código abaixo para salvar os dados em um arquivo de preferência.

```
15. private void gravarDados(){
16.     // Recebemos uma instância do arquivo de preferências
17.     SharedPreferences sharedPreferences = getSharedPreferences("MeusDados", MODE_PRIVATE);
18.
19.     // Recebemos uma instância para iniciarmos no modo de edição
20.     SharedPreferences.Editor editor = sharedPreferences.edit();
21.
22.     // Adicionamos os dados que desejamos salvar
23.     editor.putString("perfil","administrador");
```

```
24.   editor.putFloat("valor", 75.4f);
25.   editor.putInt("quantidade", 28);
26.   editor.putBoolean("privilegio", true);
27.
28.   // Realizamos a gravação do arquivo
29.   editor.apply();
30. }
```

Um objeto *SharedPreferences* fornece métodos que nos permitem recuperar os dados. Atenção: para recuperar dados em um arquivo de preferência, não há necessidade de chamar o método *edit()*.

Para cada tipo de dado que se deseja recuperar, também há um método correspondente. Os métodos para recuperar os dados salvos recebem dois parâmetros: i) o primeiro refere-se à chave que se deseja buscar; ii) o segundo refere-se a um valor padrão que será retornado caso a chave não exista. Observe no Quadro 2.3 os métodos disponíveis para recuperar os valores utilizando *SharedPreferences*.

Quadro 2.3 | Métodos para recuperar valores do *SharedPreferences*

Método	Descrição
getString(*String* chave, *String* valorPadrao)	Retorna o valor do tipo *String*.
getFloat(*String* chave, *float* valorPadrao)	Retorna o valor do tipo *float*
getLong(*String* chave, *long* valorPadrao)	Retorna o valor do tipo *long*
getInt(*String* chave, *int* valorPadrao)	Retorna o valor do tipo *int*
getBoolean(*String* chave, *boolean* valorPadrao)	Retorna o valor do tipo *boolean*
getStringSet(*String* chave, *Set<String>* valorPadrao)	Retorna o valor do tipo *Set<String>*

Fonte: elaborado pelo autor.

Exemplificando

Analise as linhas e os comentários abaixo para recuperarmos os valores de um arquivo de preferência.

```
32. private void localizarDados(){
33.     // Recebemos uma instância do arquivo de preferências
34.     SharedPreferences sharedPreferences = getSharedPreferences("MeusDados", MODE_PRIVATE);
35.
36.     // Recuperamos os valores através das chaves
37.     String perfil = sharedPreferences.getString("perfil", "usuário comum");
38.     float valor = sharedPreferences.getFloat("valor", 0);
39.     int quantidade = sharedPreferences.getInt("quantidade", 0);
40.     boolean privilegio = sharedPreferences.getBoolean("privilegio", false);
41.
42.     // Esta chave não existe, portanto será retornado o segundo parâmetro fornecido ao chamar o método
43.     float multa = sharedPreferences.getFloat("multa", 100);
44. }
```

Linha 32: criamos um método para recuperar os valores.

Linhas 37 até 40: recuperamos os valores com que desejamos trabalhar e os armazenamos em variáveis locais.

Linha 43: tentamos recuperar um valor com uma chave inexistente. O valor retornado será 100 (cem), pois foi fornecido como valor padrão ao chamar o método *getFloat()*.

Finalizamos a construção dos métodos gravarDados() e localizarDados(). Nosso próximo passo é analisarmos técnicas para que possamos realizar melhorias no código. Vamos focar em duas técnicas para que no final delas seja possível apresentar o código completo da classe *MainActivity*.

1) Para iniciarmos o estudo da primeira técnica, observe que em ambos os métodos criados recebemos uma instância de *SharedPreferences* conforme linha abaixo.

```
SharedPreferences sharedPreferences = getSharedPr
eferences("MeusDados", Context.MODE_PRIVATE);
```

Como já estudamos, o primeiro parâmetro fornecido é o nome do arquivo em que os dados serão armazenados. É muito comum, porém por equívoco, programadores fornecerem nomes de arquivos inválidos ou que ainda não existam. Ao tentar recuperar um valor deste arquivo, o retorno referente a uma chave não será o valor desejado.

Para solucionar esta questão, recomenda-se criar uma constante com o nome do arquivo e, sempre que formos utilizar o *SharedPreferences*, usaremos a constante criada.

2) A segunda técnica em que focaremos nos permite realizar gravações em um arquivo de preferência de forma mais simplificada. Em alguns casos, é preciso armazenar apenas um valor ou outro, portanto, não há necessidade de criarmos explicitamente um objeto *Editor*.

Para solucionar esta segunda questão, chame os métodos de forma encadeada, ou seja, um após o outro.

Analise o código e os comentários abaixo. No código são apresentadas as soluções para as duas técnicas destacadas anteriormente. Todo o código poderá ser modificado para que atenda às necessidades do seu aplicativo.

```
1.  package com.segundoprojeto
2.
3.  import android.content.Context;
4.  import android.content.SharedPreferences;
5.  import android.os.Bundle;
6.  import android.support.v7.app.AppCompatActivity;
7.
8.  public class MainActivity extends AppCompatActivity {
9.
10.   // Criamos uma constante com o nome do arquivo
11.   public static final String ARQUIVO_MEUS_DADOS =
"MeusDados";
12.
13.   @Override
14.  protected void onCreate(Bundle savedInstanceState)
{
15.   super.onCreate(savedInstanceState);
16.   setContentView(R.layout.activity_main);
17.  }
18.
19.  private void gravarDados() {
20.   // Observe que fazemos referência à constante
criada na linha 11
      SharedPreferences sharedPreferences =
21.  getSharedPreferences(ARQUIVO_MEUS_DADOS, Context.
MODE_PRIVATE);
22.
      // De forma encadeada, acessamos o objeto
23.  Editor, informamos os valores a serem armazenados
e gravamos o arquivo
24.   sharedPreferences.edit()
25.       .putString("perfil", "administrador")
26.       .putFloat("valor", 75.4f)
27.       .putInt("quantidade", 28)
28.       .putBoolean("privilegio", true)
29.       .apply();
30.  }
31.
32.  private void localizarDados() {
33.   // Observe que fazemos referência à constante
criada na linha 11
```

```
        SharedPreferences sharedPreferences =
34.  getSharedPreferences(ARQUIVO_MEUS_DADOS, Context.
     MODE_PRIVATE);
35.
36.     // Recuperamos os valores através das chaves
        String perfil = sharedPreferences.
37.  getString("perfil", "usuário comum");
        float valor = sharedPreferences.getFloat("valor",
38.  0);
        int quantidade = sharedPreferences.getInt("quantidade", 0);
39.

        boolean privilegio =
40.  sharedPreferences.getBoolean("privilegio", false);
41.     }
42.  }
```

Pesquise mais

Você poderá encontrar mais informações sobre armazenamento de dados no site oficial de desenvolvimento do Android, disponível em: <https://developer.android.com/guide/topics/data/data-storage#pref>. Acesso em: 4 jul. 2018.

Finalizamos nossos estudos sobre armazenamento de dados com estrutura *key-value* e aprendemos a gravar e recuperar dados através do *SharedPreferences*. Continue seus estudos e pratique os conteúdos apresentados nesta seção para aplicá-los em seus aplicativos.

Sem medo de errar

Nesta segunda fase de desenvolvimento do aplicativo para o SEBRAE, você deverá desenvolver a lógica para que os lançamentos dos valores anuais sejam armazenados no aplicativo.

Os dados deverão ser armazenados utilizando *SharedPreferences* com a estrutura *key-value*, ou seja, uma chave do tipo *String* correspondente ao ano e um valor atribuído à chave correspondente ao saldo de faturamento. Veja o exemplo abaixo:

{ "2016" : 9800;

"2017" : 15000;

"2018" : 42000 }

Dando continuidade ao projeto do Android Studio da seção anterior, acesse a classe *MainActivity* e siga os passos abaixo:

i) Declare todos os objetos utilizados no layout e faça as referências correspondentes através do método *findViewById(int)* e a Classe R no método *onCreate()*.

ii) Crie uma constante com o nome do arquivo que será utilizado pelo *SharedPreferences*.

iii) Crie 3 métodos que serão responsáveis por:

a) Adicionar um valor correspondente ao ano selecionado.

b) Excluir um valor correspondente ao ano selecionado.

c) Exibir o saldo correspondente ao ano selecionado.

Adicione parâmetros para cada método criado conforme os códigos abaixo, desta forma facilitará o acesso ao *SharedPreferences*. O parâmetro *"ano"* indica a chave que deverá ser acessada no *SharedPreferences*. O parâmetro *"valor"* indica o valor que deverá ser adicionado ou subtraído do saldo atual armazenado.

```
private void adicionarValor(int ano, float valor) {
        SharedPreferences sharedPreferences = 
getSharedPreferences(ARQUIVO_MEUS_DADOS, Context.
MODE_PRIVATE);
 float valorAtual = sharedPreferences.getFloat(String.
valueOf(ano), 0);
  float novoValor = valorAtual + valor;
  sharedPreferences.edit()
      .putFloat(String.valueOf(ano), novoValor)
      .apply();
}

private void excluirValor(int ano, float valor) {
        SharedPreferences sharedPreferences = 
getSharedPreferences(ARQUIVO_MEUS_DADOS, Context.
```

```
MODE_PRIVATE);
 float valorAtual = sharedPreferences.getFloat(String.
valueOf(ano), 0);
  float novoValor = valorAtual - valor;
  if (novoValor < 0) {
    novoValor = 0;
  }
  sharedPreferences.edit()
      .putFloat(String.valueOf(ano), novoValor)
      .apply();
}

private void exibirSaldo(int ano) {
        SharedPreferences sharedPreferences =
getSharedPreferences(ARQUIVO_MEUS_DADOS, Context.
MODE_PRIVATE);
   float saldo = sharedPreferences.getFloat(String.
valueOf(ano), 0);
    mTextViewSaldo.setText(String.format("R$ %.2f",
saldo));
}
```

iv) Na seção anterior, nós criamos uma interface *OnValueChangeListener*. Dentro do método *onValueChange()* você deverá chamar o método criado anteriormente *exibirSaldo(ano)* e passar como parâmetro o novo ano selecionado.

```
    private NumberPicker.OnValueChangeListener
24. valorAlteradoListener = new NumberPicker.
    OnValueChangeListener() {
25.   @Override
26.   public void onValueChange(NumberPicker
    numberPicker, int oldValue, int newValue) {
27.       // Ao selecionar um ano diferente no
    NumberPicker, exibe o saldo para o usuário
28.       exibirSaldo(newValue);
29.   }
30. };
```

v) Também chame o método *exibirSaldo(mNumberPicker.getValue())* no *onCreate()* para que seja exibido o valor do saldo assim que a *Activity* for iniciada.

vi) Crie um objeto *OnClickListener* e vincule-o ao botão "Confirmar" para detectar interações de toques entre o usuário.

vii) Adicione o código abaixo referente ao método *onClick()* disponível em *OnClickListener*. Será necessário:

a) Linha 36: recuperar o valor digitado pelo usuário e verificar se não está vazio.

b) Linha 38: converter o valor digitado pelo usuário para *float*.

c) Linha 41: recuperar o ano selecionado através do comando *mNumberPicker.getValue()*.

d) Linhas 43 até 55: verificar qual *RadioButton* está selecionado. Caso seja Adicionar, chame o método criado anteriormente *adicionarValor(ano, valor)*; se for Excluir, chame o método *excluirValor(ano, valor)*.

e) Linha 56: exibir o novo saldo para o usuário através do método criado anteriormente *exibirSaldo(ano)*.

```
    private                    View.OnClickListener
32. mButtonConfirmarClick      =      new      View.
    OnClickListener() {
33.   @Override
34.   public void onClick(View view) {
35.     // Verifica se o EditText Valor não está
    em branco
36.   if    (!mEditTextValor.getText().toString().
    isEmpty()) {
37.     // Recupera o valor digitado e o converte
    para float
38.     loat valor = Float.parseFloat(mEditTextValor.
    getText().toString());
39.
40.     // Recupera o ano selecionado
41.     int ano = mNumberPicker.getValue();
42
43.     // Verifica qual RadioButton está
    selecionado
```

```
        // Recuperamos o ID do RadioButton que
44. está selecionado e comparamos com o ID dos
    RadioButtons que criamos no layout
45.       switch (mRadioGroup.getCheckedRadioButtonId()) {
        // Caso o RadioButton Adicionar esteja
46. selecionado
47.       case R.id.radioAdicionar:
48.         adicionarValor(ano, valor);
49.         break;
50.
        // Caso o RadioButton Excluir esteja
51. selecionado
52.       case R.id.radioExcluir:
53.         xcluirValor(ano, valor);
54.         break;
55.     }
56.     exibirSaldo(ano);
57.   }
58.  }
69. };
```

Finalizamos a segunda etapa de desenvolvimento do aplicativo para o SEBRAE. Agora o aplicativo é capaz de armazenar dados através do *SharedPreferences* com estrutura *key-value*. Caso sinta-se mais confortável com o conteúdo, faça modificações nos métodos criados para apresentar uma lógica com a sua identidade. Bom trabalho!

Avançando na prática

Cadastro de informações

Descrição da situação-problema

Uma empresa de jornalismo o contratou para atuar com uma equipe de programadores para desenvolverem um aplicativo Android que exibirá notícias recentes de acordo com o perfil do

usuário. Você deverá criar uma *Activity* responsável por coletar informações pessoais sobre o usuário. Essas informações devem ser armazenadas no dispositivo móvel para que o aplicativo ofereça uma experiência de navegabilidade personalizada. Você deverá construir o layout e a lógica necessária para armazenar o nome e a faixa etária do usuário. Após a conclusão, apresente o resultado para que a equipe de programadores também tenha acesso a essas informações.

Resolução da situação-problema

Para desenvolver a tarefa que lhe foi delegada, crie um projeto no Android Studio com uma *Empty Activity* (atividade vazia). Em seguida, modifique o arquivo activity_main.xml para apresentar o layout solicitado pela empresa. Observe a sugestão de criação conforme Figura 2.13.

Figura 2.13 | Layout para cadastro de informações

Fonte: elaborada pelo autor.

No próximo passo você deverá acessar a classe *MainActivity* e desenvolver a lógica para:

i) Acessar os elementos *EditText*, *RadioGroup* e *Button* do layout.

ii) Detectar uma interação de toque no botão Confirmar utilizando um *Listener*.

iii) Recuperar o nome digitado pelo usuário e verificar se não está vazio.

iv) Verificar qual faixa etária está selecionada no *RadioGroup* através de um *switch*.

v) Criar um método responsável por gravar os dados em um arquivo de preferência utilizando *SharedPreferences*.

Após finalizar o trabalho, apresente-o para que a equipe de desenvolvedores dê continuidade no aplicativo.

Faça valer a pena

1. Os aplicativos Android podem oferecer uma experiência de navegabilidade mais sofisticada para os usuários com as soluções disponíveis de armazenamento de dados. *SharedPreferences* é uma solução de armazenamento de dados com a estrutura definida em *key-value*. Para acessá-la, basta chamar o método *getSharedPreferences()* a partir de uma *Activity*.
É possível armazenar os seguintes dados utilizando *SharedPreferences*:
I- "titulo" : "Aplicação com *SharedPreferences*"
II- 1990 : "ano de nascimento"
III- "conteúdo" : "Conteúdo completo"
IV- "objeto" : new Objeto()

Analise os dados apresentados e identifique quais são aceitos para serem armazenados através do *SharedPreferences*.

a) Apenas as alternativas I, III e IV estão corretas.
b) Apenas as alternativas II e IV estão corretas.
c) Apenas as alternativas II e III estão corretas.
d) Apenas as alternativas I e III estão corretas.
e) Apenas as alternativas I, II e III estão corretas.

2. Um objeto *SharedPreferences.Editor* é responsável por receber valores para serem armazenados utilizando *SharedPreferences*. Para cada tipo de valor que se deseja armazenar, há um método específico que deve ser chamado. Por exemplo, caso deseje atribuir à chave "atualizado" um valor *boolean*, é necessário chamar o método *putBoolean("atualizado", false)*.

Qual será o comportamento do aplicativo caso o método *putBoolean("atualizado", true)* seja chamado novamente?

a) Será lançado um erro informando que a chave já existe.
b) O valor já armazenado será substituído pelo novo valor.
c) O método será ignorado, pois já existe uma chave armazenada.
d) Será perguntado ao usuário se deseja substituir o valor existente.
e) O aplicativo será encerrado.

3. Um objeto *SharedPreferences* nos fornece métodos específicos para recuperar cada tipo de valor armazenado. Ao chamar esses métodos é necessário fornecer dois parâmetros. O primeiro corresponde à chave e o segundo corresponde ao valor padrão. Observe os códigos abaixo.
sharedPreferences.edit().putFloat("novo_total", 800).apply();
float total = *sharedPreferences.getFloat*("total", 0).

Analise o código apresentado e assinale a alternativa que contém o valor da variável total.

a) Será lançado um erro, pois não existe a chave "total".
b) Será lançado um erro, pois não foi chamado o método *edit()* para recuperar o valor da chave.
c) Será exibida uma mensagem para o usuário informando-o que não foi encontrado nenhum valor.
d) O valor da variável *total* será 800 (oitocentos).
e) O valor da variável *total* será 0 (zero).

Seção 2.3

Trabalhando com Novas *Activities*

Diálogo aberto

Caro aluno, bem-vindo à última seção desta unidade. Ao utilizarmos os aplicativos instalados em nosso dispositivo móvel, podemos perceber que muitos deles apresentam mais de uma tela de interação. Por exemplo, um aplicativo de rede social apresenta uma tela para visualizar o feed de notícias, uma para realizar postagens e outra para visualizar o perfil do usuário. Nesta seção vamos trabalhar com a estrutura de várias *Activities* em um aplicativo. A criação de várias telas nos permite organizar e oferecer mais funcionalidades em nossos aplicativos.

Dando continuidade no aplicativo entregue ao SEBRAE, é hora de explorar novas funcionalidades. Você deseja liberar uma versão do aplicativo para que os microempreendedores possam personalizá-lo com o nome fantasia da empresa. O nome será exibido na barra da tela inicial. É importante manter este dado salvo, assim o microempreendedor não precisará digitar toda vez que acessar o aplicativo.

Você deve criar uma *Activity* para ser chamada a partir da tela inicial, para que os microempreendedores possam digitar o nome de sua empresa. Construa o layout da nova *Activity* fornecendo ao usuário elementos que o permitam digitar e salvar o nome fantasia da empresa. Em seguida, adicione a lógica necessária na classe *MainActivity.java* para determinar o melhor momento para recuperar esta informação e exibi-la na barra da tela inicial.

Para que a nova versão do aplicativo que será disponibilizado pelo SEBRAE possa ser concluída, estudaremos nesta seção como criar mais de uma *Activity* em um aplicativo. Também estudaremos como realizar a comunicação entre as *Activities* criadas. Os conceitos apresentados nesta seção abrirão oportunidades para desenvolver mais funcionalidades em seu aplicativo.

Boa criação!

Não pode faltar

Até o momento, construímos aplicativos que iniciam uma *Activity* para que o usuário possa interagir, porém, os aplicativos podem apresentar mais de uma *Activity*. Para criarmos outras nos aplicativos Android, siga os passos apresentados abaixo correspondentes à Figura 2.14:

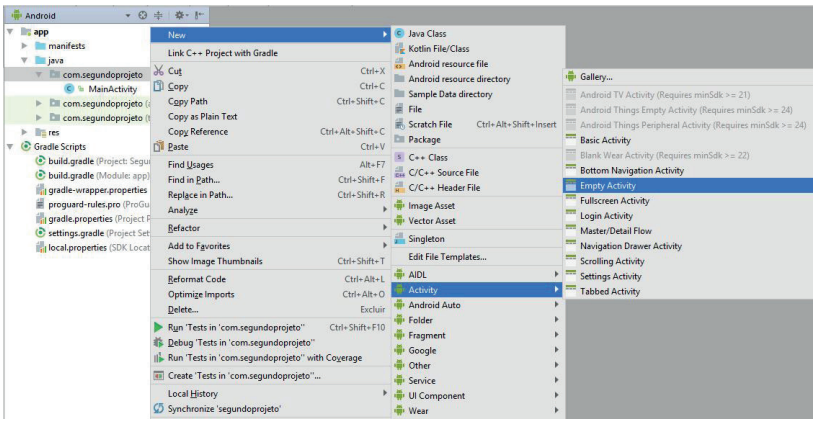
1) Na Janela *Project*, selecione o pacote ao qual deseja adicionar a *Activity*.

2) Clique com o botão direito no pacote selecionado.

3) Selecione o menu *New*, em seguida *Activity* e o tipo que se deseja criar.

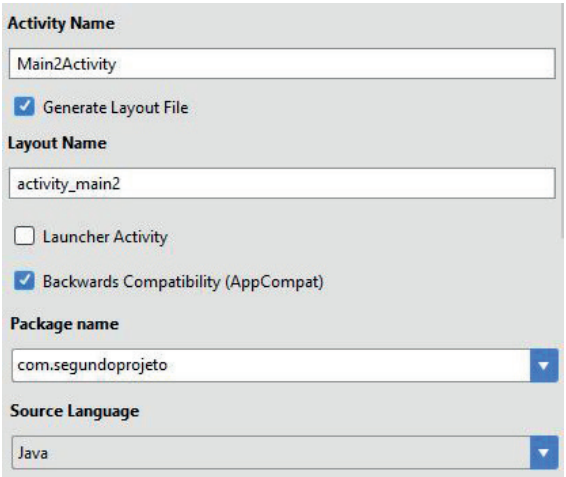
Figura 2.14 | Criação de uma *Empty Activity*

a) Menu para criar uma *Activity*



Fonte: captura de tela do Android Studio.

b) Opções para configurar uma nova Empty Activity



Fonte: captura de tela do Android Studio.

Opção 1: defina o nome da *Activity*. Este nome refere-se ao arquivo com o código-fonte java.

Opção 2: com a opção *Generate Layout File* ("Gerar arquivo de layout") selecionada, um recurso de layout será criado para a *Activity*.

Opção 3: defina o nome do recurso de layout que será gerado.

Opção 4: defina se a *Activity* será iniciada assim que o usuário executar o aplicativo.

Opção 5: ao manter *BackwardsCompatibility (AppCompat)* selecionado, a classe java referente à *Activity* estenderá um objeto *AppCompatActivity*. Caso seja desmarcada esta opção, a classe java estenderá um objeto *Activity*.

Opção 6: defina em qual pacote será criada a *Activity*.

Opção 7: defina a linguagem de desenvolvimento.

Atenção

Caso a opção *Launcher Activity* esteja selecionada, é necessário verificar no arquivo *AndroidManifest.xml* se apenas uma *Activity* possui o Intent-filter:

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER" />.

Observe o código abaixo do arquivo *AndroidManifest.xml*. Neste código são declaradas duas *Activities*, porém apenas uma delas possui o *Intent-filter* citado.

```
12. <activity android:name=".MainActivity">
13.   <intent-filter>
14.   <action android:name="android.intent.
action.MAIN" />
15.   <category android:name="android.intent.
category.LAUNCHER" />
16.   </intent-filter>
17. </activity>
18.
19. <activity android:name=".MenuActivity"></
activity>
```

Com o arquivo *AndroidManifest.xml* configurado desta forma, o sistema Android saberá que a *MainActivity* deverá ser iniciada ao executar o aplicativo.

Para que possamos trabalhar com mais de uma *Activity*, devemos conhecer o objeto *Intent*.

"*Intent* é um objeto de mensagem que pode ser usado para solicitar uma ação de outro componente de aplicativo." (Android, 2018, p. 1). Em outras palavras, conforme estudamos na Unidade 1, Seção 1.2, uma *Activity* é um componente de aplicativo Android. Uma *Activity* representa uma tela para o usuário interagir e, para que uma *Activity* possa trocar dados com outra e até mesmo iniciar outra, é necessário utilizar um objeto *Intent*.

Existem dois tipos de objetos *Intent*. Para que possamos compreendê-los, analise as definições e os cenários abaixo.

i) *Intent* Explícito: conforme o nome sugere, *Intent* Explícito é utilizado quando o nome do componente Android que desejamos carregar está mencionado no objeto *Intent*.

ii) *Intent* Implícito: declara uma ação que deverá ser executada pelo sistema Android, porém o nome do componente que executará esta ação não está mencionado.

Cenário 1: você está desenvolvendo um aplicativo que terá duas *Activities*. A primeira refere-se à tela de login e a segunda refere-se à tela principal do aplicativo. Ao iniciar o aplicativo, a tela de login será exibida para o usuário. Após ele preencher os dados de login e pressionar o botão "Entrar", os dados serão verificados pelo aplicativo. Se estiverem corretos, a segunda *Activity* será iniciada e deverá receber o usuário de login informado.

Neste cenário, vamos trabalhar com o *Intent* Explícito, pois nos permite informar explicitamente qual *Activity* será carregada. Ao declararmos um novo *Intent*, devemos informar dois parâmetros para o construtor:

i) O primeiro refere-se a um objeto *Context*.

ii) O segundo refere-se à *Class* do componente que será iniciado.

Observe o método abaixo que exemplifica o botão "Entrar" e apresenta os códigos para carregar a *Activity MenuActivity.java*.

```
31. private void realizarLogin(String usuario, String
    senha) {

32. if ((usuario.equals("Aluno")) && (senha.equals("123")) ){

33.   Intent intent = new Intent(this, MenuActivity.
    class);
34. intent.putExtra("login", usuario);
35. startActivity(intent);
36.   }
37. }
```

Linha 33: é criado um objeto *Intent* que recebe dois parâmetros: i) o primeiro refere-se a um *Context*. Conforme estudamos na seção anterior, uma *Activity* é filha da classe *Context*, desta forma, ao informarmos o valor *this* para o parâmetro, estamos fazendo referência à *Activity* e à classe *Context*. Porém, caso esteja trabalhando com uma *Inner Class* (por exemplo, ao utilizar

um *Listener*), chame o método *getBaseContext()* para recuperar a instância de *Context*. ii) o segundo parâmetro refere-se à *Class* do componente, ou seja, devemos informar o nome da *Activity* que desejamos iniciar com o sufixo *.class*.

Linha 34: o método *putExtra()* trabalha com a estrutura *key-value*. Este método nos permite adicionar informações extras para enviar com o *Intent*. No nosso exemplo enviaremos a chave "login" com o valor correspondente ao *usuário*.

Linha 35: utilizamos o método *startActivity(intent)* para iniciar uma *Activity* e passamos como parâmetro o *Intent* criado.

Exemplificando

O código abaixo exemplifica como recuperar as informações transmitidas por um objeto *Intent*. Para que a *MenuActivity* recupere a informação enviada com o *Intent*, complemente o método *onCreate()* da classe *MenuActivity.java* conforme o código abaixo:

```
8.  @Override
9.  protected void onCreate(Bundle
    savedInstanceState) {
10.   super.onCreate(savedInstanceState);
11.   setContentView(R.layout.activity_menu);
12.   Intent intent = getIntent();
13.   String usuarioRecebido = intent.
    getStringExtra("login");
14. }
```

Linha 12: o método *getIntent()* nos retorna o objeto *Intent* que iniciou a *Activity*.

Linha 13: declara uma variável local do tipo *String* chamada "usuarioRecebido" que receberá o valor enviado pelo objeto Intent.

Em seguida, é chamado o método *getStringExtra()* e, como parâmetro, é fornecida a chave referente ao valor que se deseja recuperar. Observe que a chave deverá ser a mesma utilizada no objeto *Intent* que iniciou a Activity, conforme exemplificado na linha 34 do código anterior.

Cenário 2: você está desenvolvendo um aplicativo e deseja compartilhar alguma informação dele por um e-mail, porém,

você não quer criar a funcionalidade de e-mail no seu aplicativo. É possível usar um *Intent* Implícito para atingir esta funcionalidade. Você utilizará uma ação para enviar dados, desta forma, o Android exibirá uma caixa de seleção com todos os aplicativos instalados no dispositivo móvel, capazes de enviar dados para que você possa escolher qual aplicativo utilizar.

Ao declararmos um novo *Intent*, diferentemente do cenário anterior, você não passará nenhum parâmetro para o construtor do objeto, mas deverá configurar o objeto *Intent* através de métodos.

Observe o código abaixo que exemplifica um *Intent* capaz de compartilhar uma mensagem de texto para outros aplicativos.

```
38. private void compartilharMensagem(String
    mensagem) {
39.   Intent sendIntent = new Intent();
40.   sendIntent.setAction(Intent.ACTION_SEND);
41.   sendIntent.putExtra(Intent.EXTRA_TEXT,
    mensagem);
42.   sendIntent.setType("text/plain");
43.   if (sendIntent.resolveActivity(getPackageMa
    nager()) != null) {
44.     startActivity(sendIntent);
45.   }
46. }
```

Linha 39: declaramos um novo objeto *Intent* com nenhum parâmetro no construtor.

Linha 40: utilizamos o método *setAction(Intent.ACTION_SEND)* para definir a ação "Enviar" que deverá ser executada no objeto *Intent*.

Linha 41: armazenamos no objeto *Intent*, através do método *putExtra()*, a chave *"Intent.EXTRA_TEXT"* e como valor definimos a mensagem que vamos compartilhar com outros aplicativos.

Linha 42: por meio do método *setType("text/plain")*, definimos o tipo de conteúdo que o *Intent* está enviando. No nosso exemplo, estamos enviando um "texto/simples".

Linha 43: caso não exista nenhum aplicativo capaz de receber o *Intent*, o aplicativo será encerrado com uma mensagem de erro. É possível contornar esta situação acrescentando uma estrutura condicional que verifica se há alguma *Activity* que pode receber o Intent criado. Se o resultado for diferente de nulo, significa que há no mínimo um aplicativo capaz de receber o *Intent* para ser processado. Desta forma, é seguro chamar o método *startActivity(sendIntent)*.

Linha 44: utilizamos o método *startActivity(intent)* para iniciar uma *Activity* de qualquer aplicativo que aceite a ação *Intent*. *ACTION_SEND* com o tipo *text/plain*.

Observe na Figura 2.15 o resultado do código acima.

Figura 2.15 | Exemplo de aplicação de *Intent* Implícito

Fonte: elaborada pelo autor.

Na caixa de seleção aberta, são exibidos todos os aplicativos instalados no dispositivo móvel capazes de receber um *Intent* com a ação *ACTION_SEND* e o tipo *text/plain*.

Um objeto *Intent* carrega informações que o sistema Android usa para determinar o componente a iniciar" (Android, 2018, p. 1). É possível adicionar essas informações ao objeto *Intent* através de métodos, e as principais informações que ele carrega são:

1) Nome do componente: sempre que um objeto *Intent* apresentar esta configuração, ele saberá exatamente qual componente executar.

2) Ação: especifica qual ação deverá ser executada pelo objeto *Intent*.

3) Dados: refere-se ao tipo de conteúdo ou ao conteúdo em si que o objeto *Intent* carrega.

4) Categoria: informa ao objeto *Intent* a categoria do componente que deverá ser executado.

5) Extras: fornecem ao objeto *Intent* informações adicionais necessárias para executá-lo, que são armazenas na estrutura *key-value*.

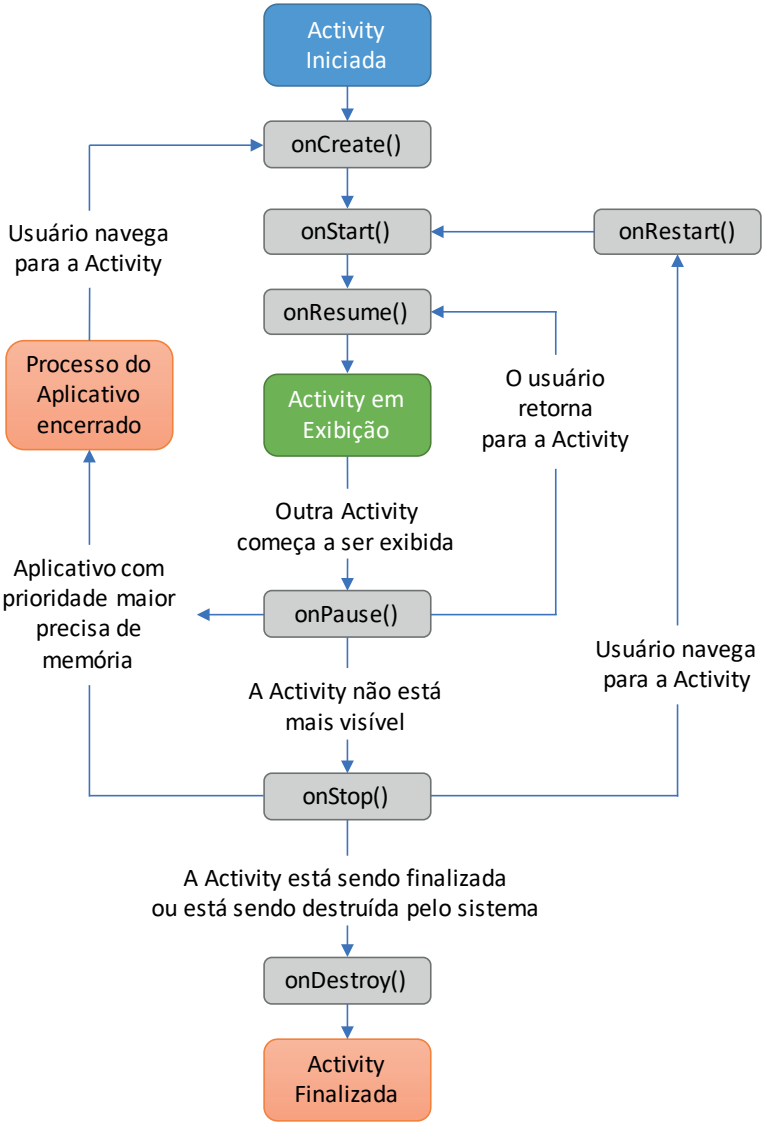
Pesquise mais

Você poderá conhecer outras ações disponíveis para trabalhar com o *Intent* através do guia de desenvolvimento para Android:

ANDROID Developers. Intents comuns. Android Developers, [s.l.], 2018. Disponível em: <https://developer.android.com/guide/components/intents-common>. Acesso em: 4 jul. 2018.

O nosso próximo passo nesta seção é aprofundar os estudos para compreendermos como uma *Activity* se comporta ao ser iniciada e encerrada. Toda *Activity* possui um ciclo de vida, ou seja, existe o momento em que ela é criada e o momento em que ela é destruída no sistema. Observe a Figura 2.16 e analise as fases do ciclo de vida de uma *Activity*.

Figura 2.16 | Ciclo de vida de uma *Activity*



Fonte: adaptada de: <https://developer.android.com/guide/components/activities/activity-lifecycle>. Acesso em: 4 jul. 2018.

A Figura 2.16 exibe as possibilidades do ciclo de vida de uma *Activity*. Vamos analisar o cenário 3 para compreender melhor.

Cenário 3: o usuário executa algum aplicativo de rede social instalado no dispositivo móvel, então os métodos on*Create(), onStart()* e *onResume()* da *Activity* inicial são chamados respectivamente. Durante a navegação pelo aplicativo, o usuário atende uma ligação. Então, automaticamente os métodos *onPause()* e *onStop()* são chamados, pois a *Activity* em que o usuário estava navegando não está mais visível. Após encerrar a ligação, o usuário retorna para o aplicativo que estava utilizando. Neste momento, a *Activity* poderá assumir dois caminhos: 1) caso o sistema precise de mais memória para processar a ligação que o usuário recebeu, a *Activity* executará novamente os métodos *onCreate(), onStart()* e *onResume()*; 2) caso o sistema mantenha o processo da *Activity*, os métodos onRestart(), onStart() e *onResume()* serão chamados.

Ao trabalharmos com uma *Activity*, é possível sobrepor cada método referente ao ciclo de vida. Segundo Android (2018), se você estiver desenvolvendo um aplicativo que reproduza vídeos, por exemplo, será possível pausar o vídeo e a conexão com a internet quando o usuário alternar para outro aplicativo, a fim de economizar recursos. Se o usuário retornar para o aplicativo, você poderá reconectar a internet e continuar o vídeo do mesmo lugar em que foi pausado.

Reflita

Cada método disponível no ciclo de vida de uma *Activity* nos permite desenvolver comportamentos específicos para o aplicativo. Quais métodos referentes ao ciclo de vida de uma *Activity* serão chamados, caso o usuário esteja com uma *Activity* visível, pressione o botão para exibir os aplicativos recentes e então decide voltar para a *Activity* em que estava trabalhando?

Analise os métodos de ciclo de vida de uma *Activity* e as recomendações de usos, conforme descritos no Quadro 2.4. Para sobrepor cada método em sua *Activity*, pressione CTRL+O e selecione o método que deseja sobrepor.

Quadro 2.4 | Métodos do ciclo de vida de uma *Activity*

Método	Descrição
onCreate()	O uso deste método é obrigatório. Ele deve ser sobreposto na *Activity* utilizando a anotação @ Override. Neste método é desenvolvida a lógica básica de inicialização da *Activity*. Por exemplo, deve-se definir um layout ou inicializar objetos que serão utilizados na *Activity*.
onStart()	Este método é chamado sempre que a *Activity* está visível para o usuário, porém é executado antes de a *Activity* se tornar interativa. Além disso, é realizado rapidamente pelo sistema.
onResume()	Quando a *Activity* assume a fase de seu ciclo de vida conhecida como *Resumed*, significa que ela está em primeiro plano para o usuário. É neste momento que o método *onResume()* é chamado.
onPause()	Segundo Android (2018), este método é chamado como uma primeira indicação de que o usuário está deixando o aplicativo. Esta afirmação não significa que o usuário realmente deixará o aplicativo, mas quer dizer que a *Activity* não está mais em primeiro plano. Caso algum recurso em uso na *Activity* seja bloqueado ao chamar este método, é necessário liberá-lo novamente no método *onResume()* ao voltar para o aplicativo.
onStop()	Quando a *Activity* não está mais visível para o usuário, significa que ela entrou na fase *Stopped*. Este estado pode acontecer quando, por exemplo, uma nova *Activity* é iniciada. Caso não haja um momento mais indicado para salvar dados em um bando de dados, você poderá salvá-los neste método.
onDestroy()	Este método é chamado sempre que uma *Activity* está na última fase. As duas possíveis causas para que uma *Activity* assuma este estado são: i) Quando o usuário rotacionar o dispositivo móvel do modo retrato para paisagem ou vice-versa. ii) Quando a *Activity* está sendo realmente finalizada pelo usuário.

Fonte: elaborado pelo autor.

Assimile

Analisando os três comportamentos do ciclo de vida de uma *Activity*, temos:

1) Quando o usuário pressiona o botão "Voltar", significa que ele tem a intenção de encerrar o aplicativo, então o método *onDestroy()* é chamado.

2) Quando o usuário pressiona o botão "Home", significa que ele não tem a intenção de encerrar o aplicativo e poderá voltar ao programa, então a *Activity* pode assumir a fase *Paused* ou *Stopped*.

3) Sempre que o usuário rotaciona o dispositivo móvel, a *Activity* que está visível para ele passará por todo o seu ciclo de vida, desde o método *onCreate()* até o *onDestroy()*.

Finalizamos o conteúdo da seção 2.3 e concluímos que:

1) É possível trabalhar com mais de uma *Activity* em um aplicativo.

2) Para que outra *Activity* seja aberta, é necessário utilizar um objeto *Intent*.

3) Para realizar a comunicação entre duas *Activities*, também é necessário utilizar um objeto *Intent*.

4) Um objeto *Intent* nos permite abrir um componente Android específico ou realizar alguma ação.

5) Toda *Activity* possui um ciclo de vida que nos permite declarar comportamentos específicos para o aplicativo.

Sem medo de errar

Você deseja desenvolver a segunda versão do aplicativo para o SEBRAE, que permitirá ao usuário gravar o nome da empresa, exibido no título da *Activity* inicial. Para elaborar essa versão, abra o projeto já criado no Android Studio. Crie uma *Empty Activity* conforme os passos abaixo:

1) Na janela *Project*, selecione o pacote dentro da pasta java e clique com o botão direito.

2) Acesse o menu *New*, *Activity* e selecione a opção *Empty Activity*.

3) Para a janela aberta, siga os passos abaixo:

a) Defina o nome para *PersonalizarActivity*.

b) Mantenha a opção *Generate Layout File* selecionada para que seja criado um recurso de layout.

c) Não selecione a opção *Launcher Activity*, pois esta não será a *Activity* a ser iniciada com o aplicativo.

d) Por fim, mantenha a opção *Backwards Compatibility (AppCompat)* selecionada para que a *Activity* estenda um objeto *AppCompatActivity*.

4) Acesse o arquivo *activity_personalizar.xml* para criar o layout que permitirá ao usuário digitar um título que será gravado. Acrescente os elementos *EditText* e *Button*.

5) Na classe *PersonalizarActivity.java*, acrescente a lógica necessária para armazenar o valor digitado pelo usuário assim que o botão for pressionado. Utilize os conceitos de *SharedPreferences*, conforme código abaixo.

```
17. // Recupera o valor digitado pelo usuário
18. String nomeFantasia = mEditTextNomeFantasia.
    getText().toString();
19.
20. // Verifica se o valor não está vazio
21. if(!nomeFantasia.isEmpty()) {
22. // Acessa o SharedPreferences e armazena o
    valor digitado pelo usuário associado à chave
    "nomeFantasia"
23. // Observe que a constante criada na
    MainActivity é fornecida no primeiro parâmetro,
    portanto é utilizado o mesmo arquivo para o
    SharedPreferences
24. getSharedPreferences(MainActivity.ARQUIVO_DE_
    GRAVACAO, Context.MODE_PRIVATE)
25. .edit()
26. .putString("nomeFantasia", nomeFantasia)
```

```
27.  .apply();
28.  }
```

6) Na classe *MainActivity.java*, sobreponha o método *onResume()* e recupere o valor armazenado no *SharedPreferences*. Atenção às dicas:

a) Para definir um título para a *Activity*, chame o método *setTitle()* e forneça como parâmetro uma *String* com o título.

b) Antes de chamar pelo método *setTitle()*, adicione uma estrutura condicional que verificará se existe algum valor armazenado no *SharedPreferences*.

```
154. @Override
155. Protected void onResume() {
156. super.onResume();
157.
158. // Recupera o SharedPreferences
159. SharedPreferences sharedPreferences =
     getSharedPreferences(ARQUIVO_DE_GRAVACAO,
     Context.MODE_PRIVATE);
160.
161. // Recupera o valor referente à chave
     "nomeFantasia".
162. // Caso não possua nenhuma chave gravada,
     retorna null
163. String nomeFantasia = sharedPreferences.
     getString("nomeFantasia", null);
164.
     // Caso nomeFantasia seja diferente de nulo,
165. há algum valor armazenado
166. if(nomeFantasia != null) {
167. // Define o título para a Activity
168. setTitle(nomeFantasia);
169. }
170. }
```

7) Acesse o *arquivo activity_main.xml* e adicione um botão para que a *PersonalizarActivity* possa ser aberta.

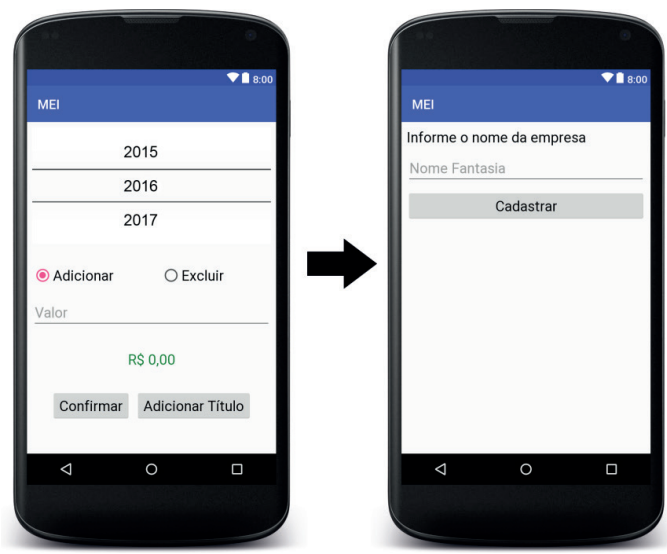
8) Novamente na classe *MainActivity.java*, adicione a lógica para iniciar a *PersonalizarActivity* assim que o botão for pressionado.

a) Caso você esteja utilizando um *Listener* para detectar uma interação de toque, ao criar o objeto *Intent*, não será possível fornecer o valor *this* para fazer referência ao *Context*, pois estamos trabalhando com uma Inner Class. Para recuperar o Context, chame pelo método *getBaseContext()*, conforme exemplificado na linha 67.

```
64.  private View.OnClickListener mButtonPersonali
     zarClickListener = new View.OnClickListener()
     {
65.    @Override
66.    public void onClick(View view) {
67.      Intent intent=new Intent(getBaseContext(),
     PersonalizarActivity.class);
68.      startActivity(intent);
69.      }
70.  };
```

Observe na Figura 2.17 a sugestão de criação dos layouts de ambas as *Activities*.

Figura 2.17 | Layouts para *MainActivity* e *PersonalizarActivity*



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Finalizamos a segunda versão do aplicativo para o SEBRAE. É possível personalizar a navegabilidade do aplicativo com o nome da empresa utilizando conceitos de armazenamento de dados e criação de novas *Activities*.

	Você poderá verificar uma opção de código-fonte para o problema proposto clicando aqui https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u2/s3/codigo.pdf ou por meio do QR Code.

Parabéns pelo trabalho!

Avançando na prática

Cálculo de IMC

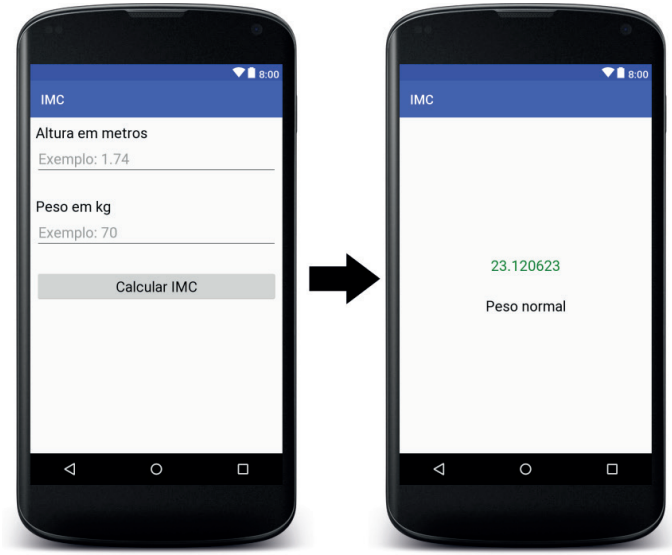
Descrição da situação-problema

Uma academia o contratou para desenvolver um aplicativo Android que será disponibilizado para os seus clientes. O objetivo do aplicativo é realizar o cálculo de Índice de Massa Corporal, conhecido como IMC. O aplicativo deverá conter duas telas de interações: na primeira, o usuário deverá fornecer a altura em metros e o peso em quilogramas. A segunda fornecerá o resultado do cálculo e a categoria em que o cliente se encontra.

Resolução da situação-problema

Para que o aplicativo solicitado possa ser desenvolvido, configure a sugestão de layout conforme a Figura 2.18 e siga os passos abaixo:

Figura 2.18 | Sugestão de layouts para aplicativo de cálculo de IMC



Fonte: captura de tela do Android Studio, elaborada pelo autor.

1) Crie um projeto no Android Studio com uma *Empty Activity*.

2) Acesse o arquivo activity_main.xml para construir o layout da *MainActivity*, de acordo com o solicitado pela empresa que o contratou.

3) Crie uma *Activity* e defina o nome para *ResultadoActivity*.

4) Acesse o arquivo activity_resultado.xml e construa o layout que será responsável por fornecer o resultado do cálculo do IMC e a categoria. Você poderá utilizar o elemento *TextView* para exibir o resultado.

Atenção para os passos 2 e 4: não se esqueça de fornecer o atributo android:id="@+id/nome_do_elemento" para todos os elementos que deverão ser acessados na classe java.

5) Acesse a classe *MainActivity*.java e adicione a lógica para:

a) Detectar uma interação de toque com o botão Calcular.

b) Recuperar os valores digitados.

c) Realizar o cálculo. Consulte em algum site de pesquisa os valores e as classificações do IMC.

d) Iniciar e enviar o resultado para a *Activity ResultadoActivity*.

6) Acesse a classe *ResultadoActivity.java* e adicione a lógica para:

a) Receber o *Intent* que iniciou a classe.

b) Recuperar os valores enviados junto com o objeto *Intent*.

c) Fazer referência ao *TextView* disposto no layout.

d) Exibir o resultado e a categoria nos elementos *TextView*.

Faça valer a pena

1. Ao desenvolvermos aplicativos que possuem mais de uma *Activity*, é necessário trabalharmos com o objeto *Intent* para iniciarmos cada *Activity* criada. Um *Intent* é classificado como *Intent* Explicíto quando sabemos exatamente qual *Activity* vamos iniciar; e *Intent* Implícito quando possuímos apenas uma ação que o *Intent* deverá executar.

Assinale a alternativa que apresenta a linha de comando que iniciará a *Activity DetalhesActivity.java*.

a) startActivity(this, DetalhesActivity.java).
b) startActivity(this, DetalhesActivity.class).
c) startActivity(new Intent(this, DetalhesActivity.java)).
d) startActivity(new Intent(this, DetalhesActivity.class)).
e) startActivity(this, new Intent(DetalhesActivity.class)).

2. Para que possamos trocar informações entre *Activities* durante a execução de um aplicativo, é necessário trabalhar com o objeto *Intent*. Inicialmente, devemos declarar um objeto *Intent* com o *Context* e a *Class* da *Activity* que será iniciada. Em seguida, com a ajuda de métodos, é possível armazenar valores na estrutura *key-value* para serem enviados à *Activity*. Observe o exemplo abaixo responsável por enviar à *Activity ProximaActivity.java* a chave *usuarioLogado* com o valor *nomeDoUsuario*.

```
Intent intent = new Intent(this, ProximaActivity.class);
intent.putExtra("usuarioLogado", nomeDoUsuario);
startActivity(intent);
```

Qual alternativa apresenta a linha de comando para que a *Activity ProximaActivity.java* recupere o valor do objeto *Intent*?

a) *String usuario = getIntent().*getStringExtra("usuarioLogado");
b) *String usuario = getIntent().getString*("usuarioLogado");
c) *String usuario = getIntent().getStringExtra*(this, "usuarioLogado");
d) *String usuario = newIntent*("usuarioLogado");
e) *String usuario = newIntent*(this, "usuarioLogado");

3. Toda *Activity* possui um estado referente ao seu ciclo de vida. Como resultado, uma *Activity* alterna o seu estado sempre que o usuário acessa ou sai do aplicativo. Ela também nos fornece métodos para cada estado que assume, para que possamos declarar comportamentos específicos no aplicativo.

Assinale a alternativa que apresenta a ordem de execução dos métodos associados ao ciclo de vida de uma *Activity* caso o usuário acesse algum aplicativo e pressiona diversas vezes o botão voltar até o aplicativo parar de ser exibido.

a) *onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy().*
b) *onCreate(), onStart(), onResume(), onStop(), onPause(), onDestroy().*
c) *onCreate(), onResume(), onStart(), onPause(), onStop(), onDestroy().*
d) *onCreate(), onStart(), onResume(), onPause(), onDestroy(), onStop().*
e) *onStart(), onCreate(), onResume(), onPause(), onStop(), onDestroy().*

Referências

ANDROID Studio. **Version 3.1.2**. [S.l.]: Google, 2018. Disponível em: <https://developer.android.com/studio/>. Acesso em: 4 jul. 2018.

ANDROID Developers. Atividades. Android Developers, [s.l.], 2018. Disponível em: <https://developer.android.com/guide/components/activities?hl=pt-BR#Lifecycle>. Acesso em: 4 jul. 2018.

_____. **Build a Responsive UI with ConstraintLayout.** [S.l.], 2018. Disponível em: <https://developer.android.com/training/constraint-layout/index.html?hl=pt-br>. Acesso em: 4 jul. 2018.

_____ . **Criar uma IU com o editor de layout.** [S.l.], 2018. Disponível em: <https://developer.android.com/studio/write/layout-editor.html?hl=pt-br>. Acesso em: 4 jul. 2018.

_____. **Eventos de entrada.** [S.l.], 2018. Disponível em: <https://developer.android.com/guide/topics/ui/ui-events.html?hl=pt-br>. Acesso em: 4 jul. 2018.

_____. **Inicie outra activity. Android Developers**, [s.l.], 2018. Disponível em: <https://developer.android.com/training/basics/firstapp/starting-activity>. Acesso em: 4 jul. 2018.

_____. **Intents e filtros de intents**. Android Developers, [s.l.], 2018. Disponível em: <https://developer.android.com/guide/components/intents-filters>. Acesso em: 4 jul. 2018.

_____. **Opções de armazenamento:** como usar preferências compartilhadas. Android Developers, [s.l.], 2018. Disponível em: <https://developer.android.com/guide/topics/data/data-storage#pref>. Acesso em: 4 jul. 2018.

_____. **Save key-value data. Android Developers**, [s.l.], 2018. Disponível em: <https://developer.android.com/training/data-storage/shared-preferences#java>. [S.l.], 2018. Acesso em: 4 jul. 2018.

_____. **SharedPreferences**. Android Developers, [s.l.], 2018. Disponível em: <https://developer.android.com/reference/android/content/SharedPreferences>. Acesso em: 4 jul. 2018.

_____. **Understand the Activity Lifecycle.** Android Developers, [s.l.], 2018. Disponível em: <https://developer.android.com/guide/components/activities/activity-lifecycle>. Acesso em: 4 jul. 2018.

DEITEL, Paul; DEITEL, Harvey; WALD, Alexander. **Android 6 para programadores:** uma abordagem baseada em aplicativos. 3. ed. Porto Alegre: Bookman, 2016.

ORACLE. **Class Object.** Oracle, [s.l.], [s.d.]. Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>. Acesso em: 4 jul. 2018.

Armazenamento local e aplicações com Android

Convite ao estudo

Caro aluno, seja bem-vindo à terceira unidade de Desenvolvimento para Dispositivos Móveis! Os nossos aplicativos estão ficando mais complexos e a cada unidade desenvolvemos mais funcionalidades para os usuários. Na segunda unidade nós estudamos como armazenar dados no aplicativo e como trabalhar com mais de uma *Activity*. Contudo, existem aplicativos que exigem a construção de diversas telas de interações e armazenam uma quantidade superior de dados. Vamos considerar um aplicativo de agenda. Existem vários contatos adicionados e diversas telas no aplicativo para que possamos adicionar, editar e consultar.

O nosso objetivo é desenvolver aplicativos que atendam às necessidades dos usuários, que muitas vezes são complexas e exigem dos nossos aplicativos uma organização para exibir as telas e uma estrutura de armazenamento mais sofisticada. Para atingirmos este objetivo, vamos conhecer e aplicar técnicas de *Fragments* e armazenamento local SQL para desenvolvermos o nosso terceiro aplicativo.

É hora de estudar brincando! Uma rede de ensino deseja disponibilizar um aplicativo Android que atuará como uma ferramenta de estudo para os seus alunos, e você foi contratado para atuar com a equipe de programadores no processo de criação desta ferramenta. Foi apresentada como proposta a criação de um jogo de perguntas e respostas para dispositivos móveis com Android.

No processo de criação deste jogo, é importante liberar duas telas para os alunos. A primeira deve permitir que eles cadastrem

as perguntas e as respostas. A segunda deve exibir uma pergunta por vez e a resposta deve permanecer oculta. Nesta segunda tela também devem ser disponibilizadas as opções de exibir a resposta e de pular a pergunta. Você deve garantir que todas as questões permaneçam armazenadas no dispositivo móvel, pois quando o aluno voltar ao aplicativo, não será necessário cadastrá-las novamente para jogar. O jogo não impõe limites, é apenas uma brincadeira. Vamos iniciar as atividades?

Para concluir o desafio proposto, você e a equipe de programadores dividiram o processo de desenvolvimento em três etapas. Inicialmente, deverão ser criados todos os layouts que os alunos acessarão no aplicativo. Gere o layout responsável por salvar as perguntas e as respostas e o layout responsável por exibi-las. O próximo passo é criar a estrutura do bando de dados. Este tipo de armazenamento é mais complexo, pois trabalharemos com dados estruturados. Para finalizar o aplicativo, vamos criar as funcionalidades para cadastrar e exibir as perguntas e as respostas.

Está lançado o desafio para esta unidade. Na primeira seção trabalharemos com *Fragments* para gerarmos as telas do aplicativo. Na segunda seção vão ser apresentadas a API SQLite e a biblioteca Room para trabalharmos com banco de dados local no Android. Para finalizar a unidade, criaremos um objeto de acesso a dados para manipularmos os dados no banco. Cada conteúdo oferecerá mais funcionalidades para os seus aplicativos. Portanto, explore-os e seja criativo!

Seção 3.1

Desenvolvendo UI com *Fragments*

Diálogo aberto

Caro aluno, é hora de iniciarmos a primeira seção desta unidade. Começaremos com uma tarefa: abra um aplicativo de notícias em um tablet e abra o mesmo aplicativo de notícias em um smartphone. Notou a diferença? No tablet, cuja tela é maior, o aplicativo de notícias disponibilizará no canto esquerdo a lista de notícias e no restante da tela será exiba a notícia. No smartphone, cuja tela é menor, a lista de notícias ocupará toda a tela e somente após selecionar uma delas, ela será carregada, ocupando toda a tela. Para desenvolvermos aplicativos que apresentam este comportamento, estudaremos os *Fragments* nesta seção.

Você foi contratado para atuar com a equipe de programadores para desenvolver um jogo para Android de perguntas e respostas para uma rede de ensino. Na primeira etapa de desenvolvimento será necessário criar os layouts projetados para smartphones.

O jogo deverá oferecer duas partes para os alunos. A primeira permitirá que eles cadastrem as perguntas e as respostas. A segunda exibirá uma pergunta por vez e a resposta deverá estar oculta. Nesta segunda parte, os usuários poderão exibir as respostas e ainda pular as perguntas cadastradas. Todos os desenvolvedores envolvidos no projeto devem apresentar os layouts de forma que atendam aos requisitos solicitados pela rede de ensino.

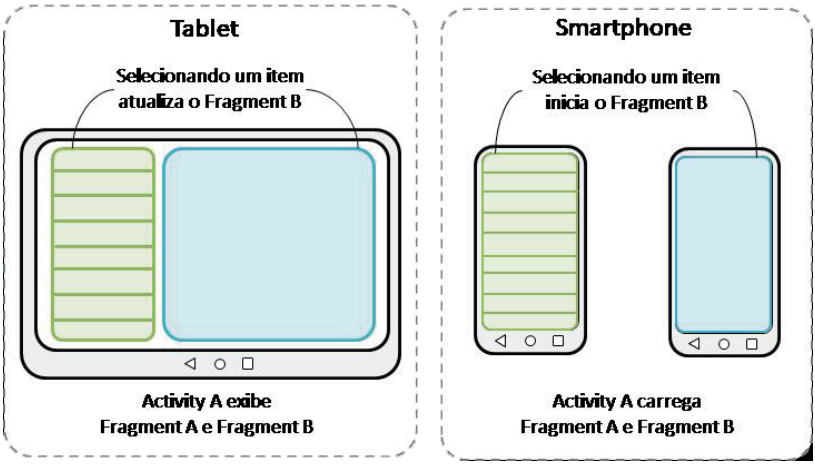
Para desenvolvermos a primeira fase do jogo, nós aprenderemos a trabalhar com *Fragments*, que são capazes de dividir uma *Activity* em partes para que possamos oferecer uma melhor experiência de navegabilidade para o usuário.

Atente-se a cada detalhe e seja criativo!

Não pode faltar

Para iniciarmos esta seção, vamos analisar a Figura 3.1, que exibe um aplicativo sendo executado em diferentes tamanhos de telas.

Figura 3.1 | Aplicativo em execução em diferentes dispositivos móveis



Fonte: adaptada de <https://bit.ly/2LKTfPd>. Acesso em: 25 jul. 2018.

A Figura 3.1 exibe um tablet e um smartphone, ambos com um aplicativo aberto.

- O tablet apresenta uma *Activity* dividida em duas partes: do lado direito uma parte que representa uma lista de itens e do lado esquerdo a outra parte que representa o conteúdo da lista.
- O smartphone nos apresenta as mesmas partes, porém não são exibidas ao mesmo tempo. É exibida a parte com a lista de itens e depois a parte com o conteúdo da lista.

O elemento que nos permite separar uma *Activity* em partes é conhecido como ***Fragment***.

Segundo Deitel, Deitel e Wald (2016), os *Fragments* foram criados a partir do Android 3.0 justamente para combinarem várias partes de um aplicativo em uma única *Activity*. A fim de aproveitar todo o espaço disponível em uma tela grande, vários *Fragments* poderão ser exibidos ao mesmo tempo.

Para criarmos *Fragments* em nosso aplicativo, seguiremos um passo a passo parecido com a criação de uma *Activity*.

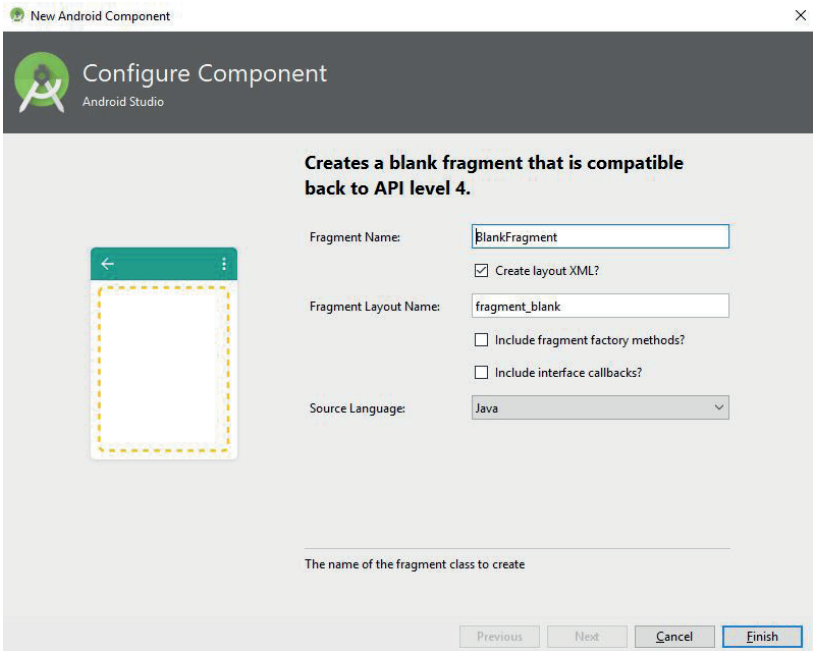
1. Clique com o botão direito do mouse em cima do pacote que vai armazenar o *Fragment*.

2. Acesse o menu *New*, em seguida *Fragment*.

3. Selecione a opção *Fragment* (*Blank*) para criarmos um *Fragmento* vazio.

A Figura 3.2 exibe a janela responsável por criar o *Fragment*.

Figura 3.2 | Janela de criação do *Fragment*



Fonte: captura de tela do Android Studio.

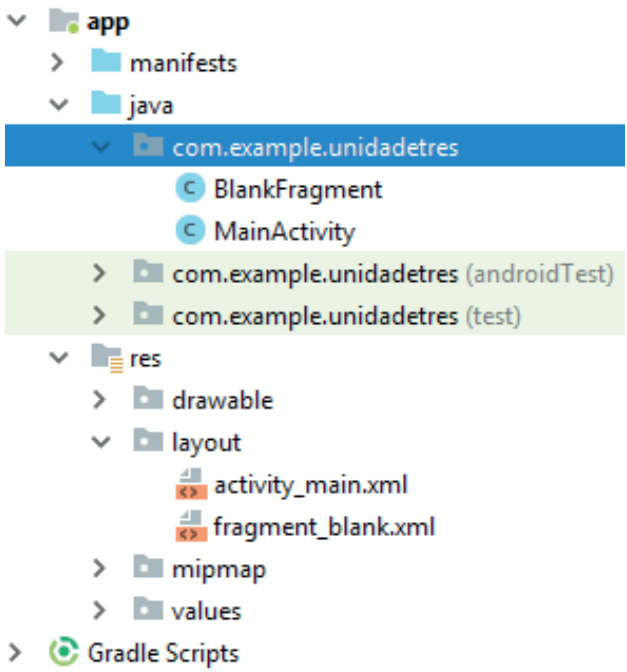
Analise a descrição de cada campo disponível na janela de criação do *Fragment*:

- O campo *Fragment Name* define o nome do *Fragment*.
- Mantenha a opção *Create layout XML* para criar um recurso de layout que será vinculado ao *Fragment*.
- *Fragment Layout Name* define o nome do recurso de layout que será salvo na pasta *res/layout*.
- As opções *Include Fragmente factory methods e Include interface callbacks* permitem adicionar uma estrutura no *Fragment* para realizar a comunicação com uma *Activity*. Neste momento não manteremos selecionadas essas opções.

- Para finalizar, defina a linguagem Java para trabalhar com o *Fragment*.

Após clicar em *Finish*, será criado um arquivo *BlankFragment.java* no pacote inicialmente selecionado e um arquivo referente ao layout na pasta *res/layout*. A Figura 3.3 exibe a atual estrutura do nosso projeto. Possuímos duas classes em nosso pacote Java: *MainActivity.java* e *BlankFragment.java*. Na pasta *res/layout*, há dois arquivos de recursos de layouts referentes a cada classe.

Figura 3.3 | Estrutura do projeto no Android Studio



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Vamos analisar o código criado para a classe referente ao *Fragment*:

```
1.  package com.example.unidadetres;
2.
3.  import android.os.Bundle;
4.  import android.support.v4.app.Fragment;
```

```
5.  import android.view.LayoutInflater;
6.  import android.view.View;
7.  import android.view.ViewGroup;
8.
9.  public class BlankFragment extends Fragment {
10.
11.   public BlankFragment() {
12.     // É obrigatório o uso de um construtor
    sem parâmetros
13.   }
14.
15.   @Override
16.   public View onCreateView(LayoutInflater
    inflater, ViewGroup container, Bundle
    savedInstanceState) {
17.
18.     // Infla o layout para o Fragment
19.     return inflater.inflate(R.layout.fragment_
    blank, container, false);
20.   }
21. }
```

Linha 9: a classe estende um objeto do tipo *Fragment*.

Linha 11: um *Fragment* deve possuir um construtor que não recebe parâmetros.

Linhas 15 e 16: o método *onCreateView()* é sobreposto. Este método é responsável por inicializar o layout do *Fragment*.

Você poderá acessar o arquivo *fragment_blank.xml* referente ao recurso de layout do *Fragment* para incluir os elementos que desejar. Eles seguem o mesmo padrão da construção de um layout para uma *Activity*.

Em seguida, é necessário preparar o layout da *Activity* para receber os *Fragments*. Acesse o recurso referente à *Activity* e adicione um elemento *FrameLayout*. Segundo Android (2018), o *FrameLayout* é projetado para reservar um espaço na tela para exibir um único elemento. Os *Fragments* serão inicializados dentro do elemento *FrameLayout* da *Activity*.

Exemplificando

Vamos exemplificar a criação de um recurso de layout referente a uma *Activity* capaz de receber um *Fragment*. Analise o código abaixo:

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <FrameLayout xmlns:android= "http://schemas.
    android.com/apk/res/android"
3.    xmlns:tools="http://schemas.android.com/
    tools"
4.    android:id="@+id/frameLayout"
5.    android:layout_width="match_parent"
6.    android:layout_height="match_parent"
7.    tools:context=".MainActivity">

8.  </ FrameLayout >
```

Linha 2: estamos utilizando como elemento raiz um *FrameLayout*.

Linha 4: atribuímos um ID para este elemento ser acessado na *Activity* ou no *Fragment*.

Linhas 5 e 6: este *FrameLayout* está configurado para preencher toda a tela da *Activity* através dos valores *match_parent*.

Neste momento, acessaremos a classe *MainActivity.java* para incluir o *Fragment* na *Activity*. Uma *Activity* nos fornece um Gerenciador de Fragmentos para que possamos trabalhar com os *Fragments*. Para acessarmos este Gerenciador de Fragmentos, devemos nos atentar ao seguinte detalhe:

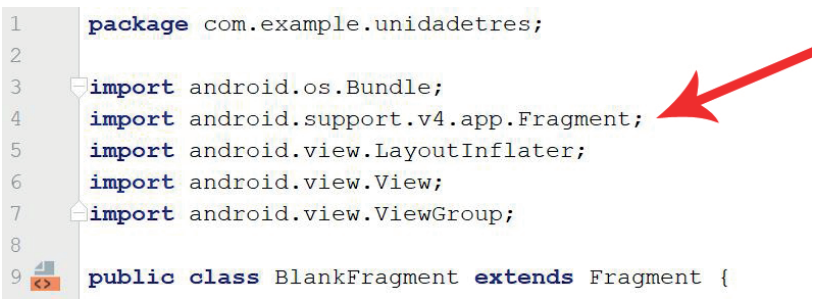
- Durante a criação do *Fragment*, é exibida a seguinte mensagem: *"Creates a blank fragment that is compatible back to API level 4"* ("Cria um fragmento em branco compatível com a API 4"), conforme exibida na Figura 3.2. Esta mensagem significa que o *Fragment* será criado utilizando a biblioteca de compatibilidade.

A biblioteca de compatibilidade definirá como acessar o Gerenciador de Fragmentos disponível na *Activity*.

Caso você já tenha criado o *Fragment* e não sabe se utiliza a biblioteca de compatibilidade, acesse a classe referente ao *Fragment*.

Na parte superior da classe, verifique quais *imports* são realizados, conforme a Figura 3.4.

Figura 3.4 | Classe de um *Fragment*



Fonte: captura de tela do Android Studio, elaborada pelo autor.

Se o *Fragment* utiliza a biblioteca de compatibilidade, constará o seguinte *import*:

```
import android.support.v4.app.Fragment;
```

Se o *Fragment* não utiliza a biblioteca de compatibilidade, constará:

```
import android.app.Fragment;
```

Verifique qual tipo de *Fragment* você está trabalhando para definir qual método chamar na *Activity* para acessar o Gerenciador de Fragmentos.

- Se a classe *Fragment* utilizar a biblioteca de compatibilidade, chame pelo método *getSupportFragmentManager()*.
- Caso contrário, chame pelo método *getFragmentManager()*.

Após chamar por algum destes métodos, será retornado um objeto *FragmentManager*.

Assimile

Segundo Android (2018), as bibliotecas de suportes oferecem funcionalidades e recursos presentes nas versões mais recentes do sistema operacional Android para que sejam aplicadas às versões mais antigas. O uso destas bibliotecas é recomendado para que o aplicativo atinja um público maior de usuários.

Ao recuperar um objeto *FragmentManager*, será necessário:

1. Iniciar uma transação.

2. Adicionar, substituir ou remover o *Fragment* da *Activity*.

3. Salvar a transação para que a ação seja realizada pelo Gerenciador de Fragmentos.

Exemplificando

Analise o exemplo abaixo referente à classe *MainActivity.java*. Este código é responsável por incluir um *Fragment* na *Activity*.

```
1.  package com.example.unidadetres;
2.
3.  import android.os.Bundle;
4.  import android.support.v4.app.
    FragmentTransaction;
5.  import android.support.v7.app.
    AppCompatActivity;
6.
7.  public class MainActivity extends
    AppCompatActivity {
8.
9.    @Override
10.   protected void onCreate(Bundle
    savedInstanceState) {
11.     super.onCreate(savedInstanceState);
12.     setContentView(R.layout.activity_main);
13.
14.     BlankFragment fragment = new
    BlankFragment();
15.
16.     FragmentManager FragmentManager =
    getSupportFragmentManager();
17.
18.     FragmentTransaction FragmentTransaction =
    FragmentManager.beginTransaction();
19.
20.     FragmentTransaction
```

```
21.         .replace(R.id.frameLayout, fragment)
22.         .commit();
23.     }
24. }
```

Linha 14: criamos uma instância do *Fragment*.

Linha 16: acessamos o objeto *FragmentManager*.

Linha 18: recuperamos o objeto *FragmentTransaction* e iniciamos uma transação chamando pelo método *beginTransaction()*.

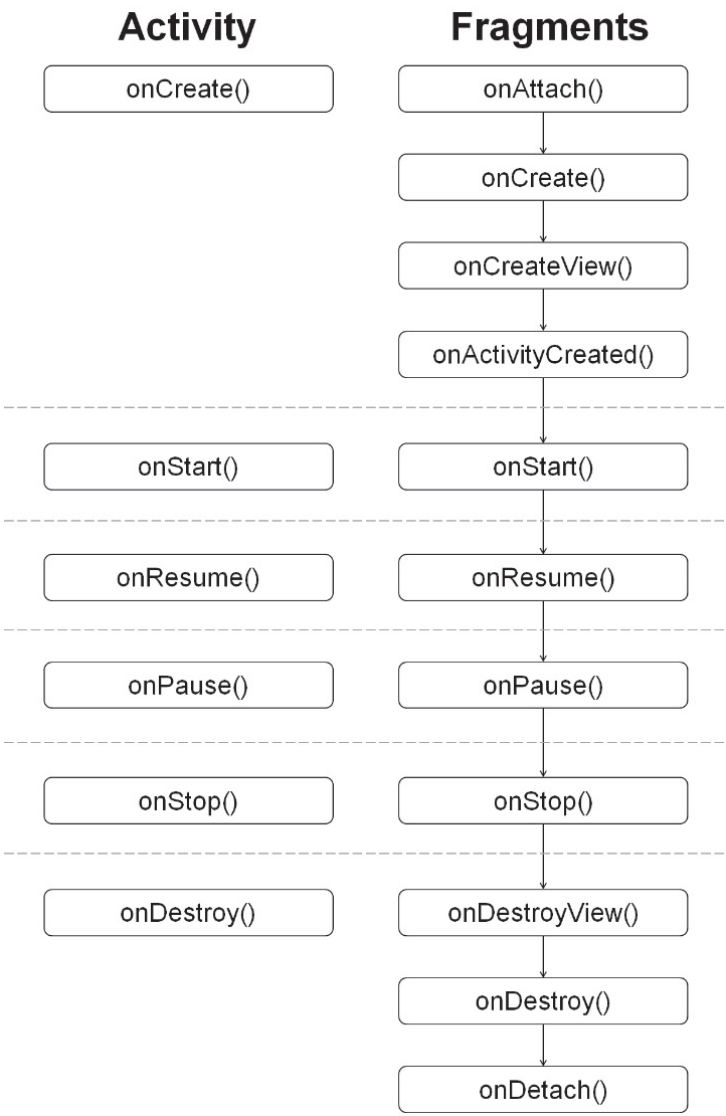
Linha 21: através do objeto *FragmentTransaction*, chamamos pelo método *replace()*, responsável por substituir o fragmento em exibição na *Activity*. Este método recebe dois parâmetros: i) o primeiro refere-se ao ID do elemento que receberá o fragmento; ii) o segundo refere-se ao *Fragment* que será incluído na *Activity*.

Linha 22: chamamos pelo método *commit()* para salvarmos a transação.

Com o código acima, o *BlankFragment* será adicionado na *Activity* ao iniciá-la.

Conforme estudamos na Unidade 2, Seção 2.3, uma *Activity* possui um ciclo de vida. Nesta seção, veremos que um *Fragment* também possui um ciclo de vida. Analise a Figura 3.5 que exibe o ciclo de vida de uma *Activity* e de um *Fragment*.

Figura 3.5 | Ciclo de vida de uma *Activity* e de um *Fragment*



Fonte: adaptado de <https://bit.ly/2LKTfPd>. Acesso em: 25 jul. 2018.

Ao analisarmos a Figura 3.5, podemos concluir que um *Fragment* possui alguns métodos extras referente ao ciclo de vida de uma *Activity*. O Quadro 3.1 apresenta a definição para alguns desses métodos.

Quadro 3.1 | Ciclo de vida de um *Fragment*

Método	Descrição
onAttach()	Chamado quando o *Fragment* é anexado a uma *Activity*.
onCreate()	Segundo Android (2018), este método é chamado ao criar o *Fragment*.
onCreateView()	Retorna um objeto *View* referente ao layout do *Fragment*. Portanto, inicialize o layout do *Fragment* neste método.
onActivityCreate()	Método chamado após o método *onCreate()* da *Activity* ser finalizado. Utilize-o para recuperar objetos que estão na *Activity* e instanciar os elementos do layout do *Fragment*.
onDestroyView()	Chamado quando os elementos do layout estiverem sendo destruídos.
onDetach()	Chamado quando o *Fragment* é desanexado de uma *Activity*.

Fonte: elaborado pelo autor.

Reflita

O ciclo de vida de um *Fragment* está vinculado ao de uma *Activity*. O *Fragment* chama por métodos correspondentes a cada estado do ciclo de vida de uma *Activity*. Por exemplo, se a *Activity* assume o estado *Resumed*, então o método *onResume()* da *Activity* e do *Fragment* serão chamados. A questão é: qual método será chamado primeiro?

Imagine que a classe *BlankFragment.java* possui um *EditText* e você deseja recuperar o valor digitado pelo usuário. Para criar a instância do *EditText* dentro de *BlankFragment.java*, será necessário seguir dois passos:

1. Sobreponha o método *onActivityCreated()*.
2. Chame por *getActivity().findViewById()*.

Observe o código abaixo:

```
@Override
public void onActivityCreated(@Nullable Bundle
savedInstanceState) {
    super.on[Activity]Created(savedInstanceState);

    mEditText = get[Activity]().findViewById(R.
id.[edit]Text);
    String textoDigitado = mEditText.getText().
toString;
}
```

Com este código, o *EditText* disponível no layout do *Fragment* foi instanciado e você poderá recuperar o valor digitado pelo usuário do mesmo modo utilizado em uma *Activity*.

Pesquise mais

Até o momento trabalhamos com uma *Activity* que exibe um *Fragment* por vez. Você deverá incluir algumas técnicas para trabalhar com mais de um *Fragment* na *Activity*. Para explorar novas possibilidades de trabalhar com *Fragments*, consulte o conteúdo do título "Exemplo" do seguinte documento:

ANDROID Developers. **Fragmentos**. Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2LKTfPd>. Acesso em: 25 jul. 2018.

Também é possível conhecer mais sobre *Fragments* através do vídeo:

ANDROID DEVELOPERS. **DevBytes:** apresentando os Fragments (Portuguese). [S.l.], 18 set. 2013. Disponível em: <https://bit.ly/2NL8Ejc>. Acesso em: 25 jul. 2018.

Até aqui, aprendemos a incluir *Fragments* em nossas *Activities*. Continue focado nos estudos para desenvolver mais funcionalidades nos aplicativos.

Sem medo de errar

Caro programador, você foi contratado para atuar com uma equipe de programadores para desenvolverem um aplicativo Android que atuará como um jogo de estudo de perguntas e respostas para uma rede de ensino.

Nesta primeira etapa de desenvolvimento, você deve estar focado nos layouts que o aplicativo apresentará para os alunos ao ser executado em um smartphone. O jogo deve apresentar dois layouts que serão desenvolvidos em dois *Fragments*:

1. O primeiro *Fragment* apresenta o layout em que os alunos poderão cadastrar as perguntas e as respostas do jogo.

2. O segundo *Fragment* apresenta o layout em que as perguntas são exibidas. Neste layout, o aluno poderá exibir a resposta da pergunta e poderá pular para a próxima pergunta cadastrada.

Para desenvolver esta etapa do aplicativo, siga os seguintes passos:

1. Crie um projeto no Android Studio com uma *Empty Activity*.

2. A *MainActivity* deverá ser preparada para receber os *Fragments* que serão criados a seguir. Acesse o arquivo *activity_main.xml* e insira um elemento FrameLayout que ocupe todo o espaço disponível do layout.

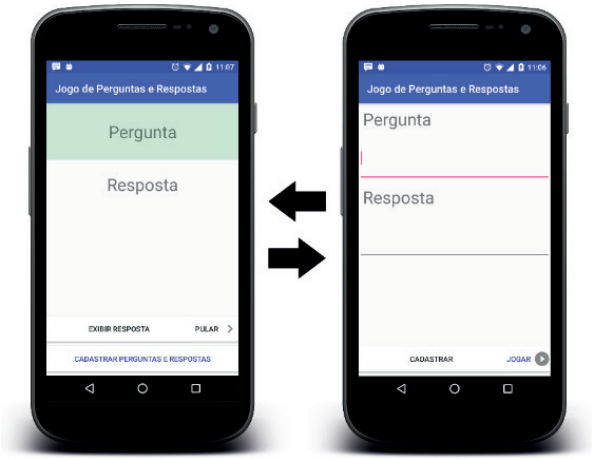
3. Em seguida, crie dois *Fragments (Blank)*. O primeiro será responsável por cadastrar as perguntas e as respostas. O segundo será responsável por exibi-las.

4. Acesse o recurso referente ao *CadastrarFragment*. Você poderá construir este layout utilizando um *ScrollView*, *LinearLayout* ou *ConstraintLayout*. Sinta-se à vontade para fazer conforme sua escolha.

5. Insira os elementos que julgar necessário para a criação do layout. A Figura 3.6 mostra uma sugestão de criação de ambos os fragmentos. Se desejar, utilize os atributos de cada elemento para adicionar cores, imagens, alterar as posições e tamanhos. Não se esqueça de adicionar os botões que permitem navegar de um *Fragment* para o outro. Se o usuário estiver no de cadastro, deverá ter um botão para o jogo iniciar. Se o usuário estiver jogando, deverá ter outro botão para iniciar o fragmento de cadastro.

6. Repita os passos 4 e 5 para criar o layout do *JogarFragment*.

Figura 3.6 | Sugestão de layout



Fonte: captura de tela do Android Studio, elaborada pelo autor.

7. Após construir os layouts dos *Fragments*, acesse a classe *CadastrarFragment.java* e crie um *Listener* para o botão Jogar.

8. Adicione o código abaixo responsável por carregar o *JogarFragment*. Observe que, para acessar o Gerenciador de Fragmentos, é necessário chamar primeiramente pelo método *getActivity()*, pois estamos trabalhando com o *Listener* dentro do *Fragment*.

```
private View.OnClickListener mJogarListener = new
View.OnClickListener() {
  @Override
  public void onClick(View view) {
    getActivity().getSupportFragmentManager().
beginTransaction()
        .replace(R.id.frameLayout, new
JogarFragment())
        .commit();
  }
};
```

9. Acesse a classe *JogarFragment.java* e crie um *Listener* para o botão Cadastrar.

10. Adicione o código abaixo responsável por carregar o *CadastrarFragment*.

```
  private View.OnClickListener mCadastrarListener
= new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        getActivity().getSupportFragmentManager().
beginTransaction()
                .replace(R.id.frameLayout, new
CadastrarFragment())
          .commit();
    }
  };
```

11. Vincule os *Listeners* aos botões. Observe o código abaixo referente ao *mJogarListener*. Repita este passo para o *mCadastrarListener* na classe *CadastrarFragment*.

```
  @Override
  public void onActivityCreated(@Nullable Bundle
savedInstanceState) {
      super.onActivityCreated(savedInstanceState);

    editTextPergunta = getActivity().findViewById(R.
id.editTextPergunta);
    editTextResposta = getActivity().findViewById(R.
id.editTextResposta);

          Button mButtonJogar = getActivity().
findViewById(R.id.buttonJogar);
       mButtonJogar.setOnClickListener(mJogarListe
ner);
  }
```

12. Para finalizar, acesse a classe *MainActivity.java* e adicione no método *onCreate()* o código responsável por inicializar o *JogarFragment* para que ele seja carregado assim que o aplicativo for executado.

Você poderá verificar uma opção de código-fonte para o problema proposto acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u3/s1/codigo_1_ddm.pdf> ou por meio do QR Code.

Até o momento, desenvolvemos todo o layout do aplicativo e acrescentamos os *Listeners* responsáveis por iniciar os *Fragments* criados. Na próxima seção você desenvolverá com a equipe a estrutura do banco de dados.

Avançando na prática

Aplicativo para estudos

Descrição da situação-problema

Uma empresa especializada em ensino de línguas estrangeiras o contratou para desenvolver um aplicativo Android, cujo objetivo é disponibilizar conteúdos de estudos para os alunos. O aplicativo deve apresentar as unidades de ensino disponíveis para os alunos e o conteúdo da unidade selecionada. Para facilitar a navegação do aplicativo em um smartphone, divida o layout em duas partes. Na primeira devem ser exibidas todas as unidades de ensino e a segunda deverá apresentar o conteúdo em si. Após finalizar, apresente os layouts para a empresa que o contratou para avaliação.

Resolução da situação-problema

Crie um projeto no Android Studio com uma *Empty Activity*. Em seguida, crie mais dois *BlankFragment*.

Observe a sugestão de criação conforme Figura 3.7 e atente-se às dicas abaixo para finalizar o aplicativo:

Figura 3.7 | Sugestão de layout para aplicativo de estudos

Fonte: captura de tela do Android Studio, elaborada pelo autor.

1. O layout da *Activity* foi construído utilizando um *ConstraintLayout* como elemento-pai.

2. O layout da *Activity* possui como elementos-filhos um *FrameLayout* e dois *Buttons*.

a) O *FrameLayout* é responsável por receber os *Fragments* criados.

b) Os *Buttons* são responsáveis por inicializar os *Fragments*.

3. Acesse a classe *MainActivity.java* e declare um *OnClickListener* para cada *Button*.

4. Adicione a lógica necessária no método *onCreate()* da *MainActivity.java* para carregar o primeiro *Fragment* ao iniciar o aplicativo.

Esta é a estrutura de um projeto que possui uma *Activity* com funcionalidade para alternar entre dois *Fragments*. Os layouts criados aqui são indicados para execução do aplicativo em um smartphone.

Você poderá verificar uma opção de código-fonte para o problema proposto acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u3/s1/codigo_2_ddm.pdf> ou por meio do QR Code.

Faça valer a pena

1. Segundo Deitel, Deitel e Wald (2016), os *Fragments* foram criados a partir do Android 3.0 e representam partes de um aplicativo Android que podem ser inseridos em uma *Activity*. O objetivo de um *Fragment* é oferecer um layout personalizado de acordo com o tamanho da tela do dispositivo móvel.

Qual elemento é recomendável incluir no layout de uma *Activity* para que ela possa receber um *Fragment*?
a) *LinearLayout*.
b) *FrameLayout*.
c) *ScrollView* e *LinearLayout*.
d) *ScrollView*.
e) *ConstraintLayout*.

2. As *Activities* e os *Fragments* em um aplicativo Android possuem um ciclo de vida. Para cada fase referente ao ciclo de vida, há um método correspondente que será chamado. É importante conhecer cada método do ciclo de vida para que seja possível implementar funcionalidades no método correto. Por exemplo, quando uma *Activity* é criada, o método onCreate() é chamado. E quando um *Fragment* é incluído em uma *Activity*, o método *onAttach()* é chamado.

Quais métodos são responsáveis por inflarem os layouts de uma *Activity* e de um *Fragment*, respectivamente?

a) *onCreate()* para a *Activity* e *onCreateView()* para o *Fragment*.
b) *onCreate()* para a *Activity* e *onCreate()* para o *Fragment*.
c) *onCreateView()* para a *Activity* e *onActivityCreated()* para o *Fragment*.
d) *onCreate()* para a *Activity* e *onAttach()* para o *Fragment*.
e) *onCreateView()* para a *Activity* e *onCreate()* para o *Fragment*.

3. Ao trabalharmos com *Fragments*, é possível criar vários layouts para separar as funcionalidades de um aplicativo. Esses fragmentos podem ser agrupados em uma *Activity*.
As *Activities* disponibilizam o método *findViewById()* para instanciar os elementos de um layout.
O código abaixo permite instanciar dentro da *Activity* um *EditText* localizado no recurso de layout.

```
mEditText = findViewById(R.id.editText);
```

Qual alternativa apresenta a linha de comando que permite instanciar dentro do *Fragment* um *EditText* localizado no recurso de layout?

a) *mEditText = new MainActivity().findViewById(R.id.editText);*
b) *mEditText = findViewById(R.id.editText);*
c) *mEditText = BlankFragment.findViewById(R.id.editText);*
d) *mEditText = new BlankFragment().findViewById(R.id.editText);*
e) *mEditText = getActivity().findViewById(R.id.editText);*

Seção 3.2

Introdução ao banco de dados local

Diálogo aberto

Caro aluno, seja bem-vindo à segunda seção desta unidade. Os aplicativos Android são capazes de armazenarem um conjunto de dados através de um banco de dados local. Podemos citar como exemplo o aplicativo de Contatos, que permite ao usuário salvar vários contatos com informações específicas para cada um. Para utilizarmos este modelo de armazenamento de dados em aplicativos Android, nesta seção nós vamos trabalhar com o Modelo Relacional de Dados.

Dando continuidade ao jogo de perguntas e repostas, todos os layouts criados na seção anterior foram aprovados pela rede de ensino, agora os programadores poderão trabalhar em cima da sua criação. Nesta segunda etapa do projeto será necessário criar a estrutura do banco de dados. Esta estrutura deve permitir que os alunos armazenem as perguntas e as respostas no dispositivo móvel.

Para desenvolvermos esta etapa, inicie o Android Studio e abra o projeto criado na seção anterior. Você deve preparar o seu projeto para trabalhar com armazenamento de dados local. Em seguida, crie as classes que representarão os dados armazenados e o banco de dados. Cada questão é composta por uma pergunta e uma resposta. Portanto, essas informações devem ser salvas no banco de dados. Utilize boas técnicas de programação, pois existem vários programadores envolvidos no projeto e, quanto maior a padronização, menor a chance de erros.

Para concluirmos a segunda etapa de desenvolvimento, vamos estudar nesta seção a API SQLite e a biblioteca Room, que nos permitirão armazenar dados estruturados no dispositivo móvel. Cada opção de armazenamento apresenta características distintas. Portanto, devemos conhecê-las, a fim de implementarmos no nosso aplicativo. Bons estudos!

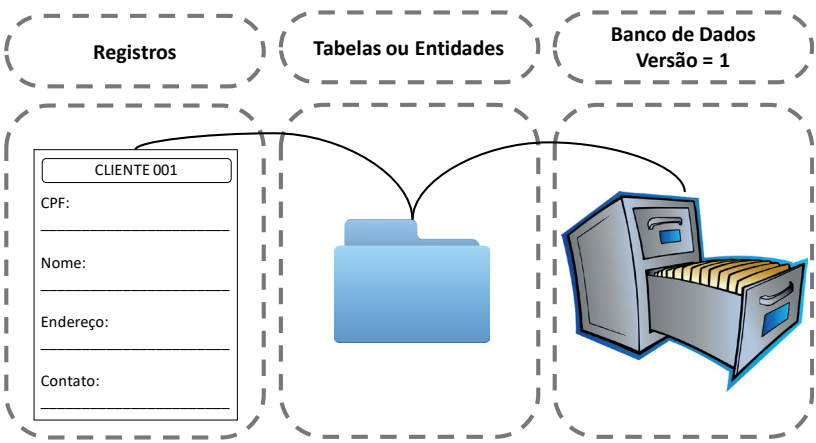
Não pode faltar

Para iniciarmos esta seção, vamos relembrar alguns conceitos de armazenamento de dados.

Segundo Alves (2014), o papel tem sido a forma mais comum para o registro de informações desde o século XV. A Figura 3.8 ilustra um processo de armazenamento de dados utilizando esta técnica. Contudo, o grande volume de informações geradas pelo homem tornou-se uma preocupação. Todas essas informações deveriam estar disponíveis para serem consultadas facilmente.

Com o avanço da tecnologia, surgiram os bancos de dados computadorizados. Segundo Cardoso e Cardoso (2012), em 1970, na IBM, Edgar Frank Codd definiu um modelo de armazenamento de dados conhecido atualmente como Modelo Relacional. Este modelo propõe que as informações que desejamos armazenar sejam gravadas em registros. Os registros devem ser agrupados em tabelas e o conjunto de tabelas representa o Banco de Dados. Através deste modelo, é possível relacionar os dados armazenados, ou seja, é possível relacionar, por exemplo, uma tabela de Cliente com uma de Compras, facilitando a consulta de quais compras foram realizadas por determinado cliente. Ainda segundo os mesmos autores, é possível implementar e manipular o modelo relacional utilizando a linguagem SQL (*Structure Query Language*).

Figura 3.8 | Representação de armazenamento de dados



Fonte: adaptado de <https://bit.ly/2LR9Lxk>. Acesso em: 26 jul. 2018.

O sistema Android nos oferece a API SQLite e a biblioteca Room para trabalharmos com armazenamento de dados relacionais.

A **API SQLite** é uma poderosa ferramenta para trabalhar com dados estruturados, porém exige do programador muito esforço para usá-la. Segundo Android (2018), o programador enfrenta o famoso *boilerplate code*, ou seja, o programador deve implementar muitos códigos para atingir determinada funcionalidade. Para solucionar o problema apresentado pelo SQLite, foi desenvolvida a biblioteca Room.

Room é a nova biblioteca para trabalhar com dados estruturados. Android (2018) recomenda o uso desta biblioteca e garante o aproveitamento do poder do SQLite.

Para trabalharmos com a biblioteca Room, é necessário incluí-la no nosso aplicativo.

1. No seu projeto, acesse o arquivo *build.gradle* (*Module: app*).
2. Insira a biblioteca Room, conforme as linhas 26, 27 e 28 abaixo.

Após inserir a biblioteca Room, sincronize o projeto.

```
1.  apply plugin: 'com.android.application'
2.
3.  android {
4.      compileSdkVersion 27
5.      defaultConfig {
6.          applicationId
    "com.rededeensino.jogodeperguntaserespostas"
7.          minSdkVersion 19
8.          targetSdkVersion 27
9.          versionCode 1
10.         versionName "1.0"
11.         testInstrumentationRunner
    "android.support.test.runner.
    AndroidJUnitRunner"
12.     }
13.     buildTypes {
14.         release {
15.             minifyEnabled false
```

```
16.             proguardFiles
    getDefaultProguardFile('proguard-android.
    txt'), 'proguard-rules.pro'
17.         }
18.     }
19. }
20.
21. dependencies {
22.     implementation fileTree(dir: 'libs',
    include: ['*.jar'])
23.     implementation 'com.android.
    support:appcompat-v7:27.1.1'
24.     implementation 'com.android.support.
    constraint:constraint-layout:1.1.0'
25.     implementation 'com.android.
    support:support-v4:27.1.1'
26.     def room_version = "1.1.0"
27.     implementation "android.arch.
    persistence.room:runtime:$room_version"
28.     annotationProcessor "android.arch.
    persistence.room:compiler:$room_version"

29.     testImplementation 'junit:junit:4.12'
30.     androidTestImplementation 'com.android.
    support.test:runner:1.0.2'
31.     androidTestImplementation 'com.android.
    support.test.espresso:espresso-core:3.0.2'
32. }
```

Pesquise mais

No código apresentado, utilizamos a atual versão 1.1.0 do Room. Você poderá alterar a versão utilizada pelo Room consultando o site das bibliotecas do Android, disponível em: <https://bit.ly/2NNE7RI>. Acesso em: 26 jul. 2018.

O segundo passo que devemos seguir é criar a representação do Registro e da Tabela que será armazenada no banco de dados. Em um aplicativo Android, é necessário criar uma classe Java para esta representação.

1. Clique com o botão direito do mouse em cima do pacote no qual deseja criar a classe.

2. Acesse o menu *New*, em seguida *Java Class*.

3. Insira o nome da classe e clique em OK.

4. Declare os atributos que deseja armazenar no banco de dados.

5. Crie os *Getters* e *Setters* para cada atributo.

Exemplificando

O exemplo a seguir nos apresenta a classe Cliente e seus atributos. Siga os passos abaixo para criar os *Getters* e *Setters* automaticamente no Android Studio.

1. Acesse a classe criada para representar a Tabela no banco de dados.

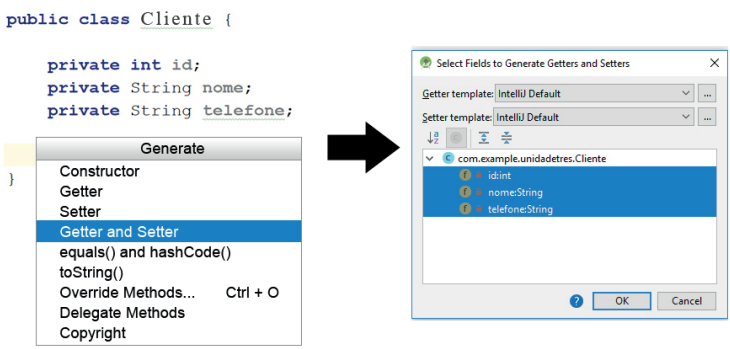
2. Posicione o cursor logo abaixo de definir os atributos.

3. Pressione ALT + INSERT.

4. A janela *Generate* será aberta, então selecione *Getter and Setter*.

5. Em seguida, selecione os atributos que deseja criar, conforme exibido na Figura 3.9.

Figura 3.9 | Janela para criar *Getters* e *Setters*



Fonte: captura de tela do Android Studio, elaborada pelo autor.

O Quadro 3.2 apresenta uma comparação entre o SQLite e o Room para criarmos a tabela no banco de dados.

Quadro 3.2 | Comparação entre SQLite e Room para criar uma tabela no banco de dados

SQLite	Para utilizar o SQLite, será necessário criar outra classe para trabalhar com o banco de dados. 1. Crie outra classe Java e estenda um objeto *SQLiteOpenHelper*. 2. Sobreponha o método *onCreate()*. 3. Insira o comando SQL para criar a tabela e os campos.
Room	Para utilizar o Room: 1. Acesse a classe criada com os *Getters* e *Setters*. 2. Insira a anotação *@Entity* ao declarar a classe.

Fonte: elaborado pelo autor.

Note que:

- O SQLite trabalha com duas classes. A classe Cliente exibida no exemplo acima representa os dados que serão armazenados no banco de dados. A segunda classe representa a criação da tabela no banco de dados através do comando SQL.
- Para utilizar o Room, basta criar uma classe e adicionar a anotação *@Entity*. Esta única classe representa os dados que serão armazenados e a própria criação da tabela no banco de dados. O Room se encarregará de processar o que for necessário para criar a tabela e os campos no banco de dados.

Analise o código abaixo que representa a classe Java responsável por armazenar o nome e o celular de um Cliente no banco de dados utilizando o Room. Note que nas linhas 9 e 10 da classe foi definido o atributo *id*, que representará a chave primária para a tabela através da anotação *@PrimaryKey(autoGenerate = true)*.

```
1.  package com.example.unidadetres;
2.
3.  import android.arch.persistence.room.Entity;
```

```
4.  import        android.arch.persistence.room.
    PrimaryKey;
5.
6.  @Entity
7.  public class Cliente {
8.
9.    @PrimaryKey(autoGenerate = true)
10.   private int id;
11.
12.   private String nome;
13.   private String telefone;
14.
15.   public int getId() {
16.     return id;
17.   }
18.   public void setId(int id) {
19.     this.id = id;
20.   }
21.   public String getNome() {
22.     return nome;
23.   }
24.   public void setNome(String nome) {
25.     this.nome = nome;
26.   }
27.   public String getTelefone() {
28.     return telefone;
29.   }
30.   public void setTelefone(String telefone) {
31.     this.telefone = telefone;
32.   }
33. }
```

O **banco de dados** é o local onde todas as tabelas estarão armazenadas. Devemos definir uma classe como ponto principal de entrada para o banco de dados. Analise o Quadro 3.3 com os passos para criar a classe com o banco de dados.

Quadro 3.3 | Criação da classe para o banco de dados

SQLite	1. Acesse a classe Java já criada que estende o objeto *SQLiteOpenHelper*. 2. Crie um construtor para a classe que será responsável por criar o banco de dados.
Room	Crie uma classe Java que atenda às seguintes especificações: 1. Seja abstrata. 2. Estenda um objeto *RoomDatabase*. 3. Possua a anotação *@Database* com os parâmetros referentes às entidades e à versão do banco de dados. Esta classe deve garantir que uma única instância do banco de dados seja criada durante a execução do aplicativo. Para atingir esta funcionalidade, utilize o padrão de projeto conhecido como *Singleton*.

Fonte: elaborado pelo autor.

Assimile

Segundo Gamma et al. (2007, p. 130), o padrão de projeto *Singleton* tem a intenção de "garantir que uma classe tenha somente uma instância". A classe que utiliza o padrão *Singleton* também deve fornecer um meio de acesso para a sua única instância.

O código abaixo representa a classe responsável pelo banco de dados utilizando Room. Analise o código e os comentários para compreendê-la.

```
1.  package com.example.unidadetres;
2.
3.  import android.arch.persistence.room.
    Database;
4.  import android.arch.persistence.room.Room;
5.  import android.arch.persistence.room.
    RoomDatabase;
```

```
6.  import android.content.Context;
7.
8.  @Database(entities = {Cliente.class},
    version = 1)
9.  public abstract class BancoDeDadosRoom
    extends RoomDatabase {
10.
11.   // Instância única para o Banco de Dados
12.   private static BancoDeDadosRoom INSTANCE;
13.
14.   // Modelo Singleton
15.   public static BancoDeDadosRoom
    getBancoDeDados(final Context context) {
16.     if (INSTANCE == null) {
17.       synchronized (BancoDeDadosRoom.class)
    {
18.         if (INSTANCE == null) {
19.           // Cria o banco de dados
20.           INSTANCE = Room.
    databaseBuilder(context.
    getApplicationContext(),
21.                   BancoDeDadosRoom.class,
    "cliente_database")
22.                   .allowMainThreadQueries()
23.                   .build();
24.         }
25.       }
26.     }
27.     return INSTANCE;
28.   }
29. }
```

Até o momento, apresentamos códigos referentes à utilização da biblioteca Room. O código abaixo representa a classe completa responsável por criar o banco de dados e a tabela utilizando o SQLite.

```
1.  package com.example.unidadetres;
2.
3.  import android.content.Context;
4.  import android.database.sqlite.
    SQLiteDatabase;
5.  import android.database.sqlite.
    SQLiteOpenHelper;
6.
7.  public class BancoDeDadosSQLite extends
    SQLiteOpenHelper {
8.
9.    // Construtor da classe responsável por
    criar o banco de dados
10.   // O nome e a versão do banco de dados são
    definidos neste construtor
11.   public BancoDeDadosSQLite(Context context)
    {
12.     super(context, "SQLiteBancoDeDados",
    null, 1);
13.   }
14.
15.   // Método responsável por criar a tabela e
    os campos no banco de dados
16.   @Override
17.   public void onCreate(SQLiteDatabase db) {
18.     String CRIAR_BD = "CREATE TABLE Cliente
    (id INTEGER PRIMARY KEY AUTOINCREMENT, nome
    TEXT, telefone TEXT)";
19.     db.execSQL(CRIAR_BD);
20.   }
21.
22.   // Método executado quando a versão do
    banco de dados é alterada
23.   // Atualmente a versão é 1, conforme valor
    explícito no construtor da classe
24.   @Override
```

```
25.    public void onUpgrade(SQLiteDatabase db,
     int oldVersion, int newVersion) {
26.    }
27.  }
```

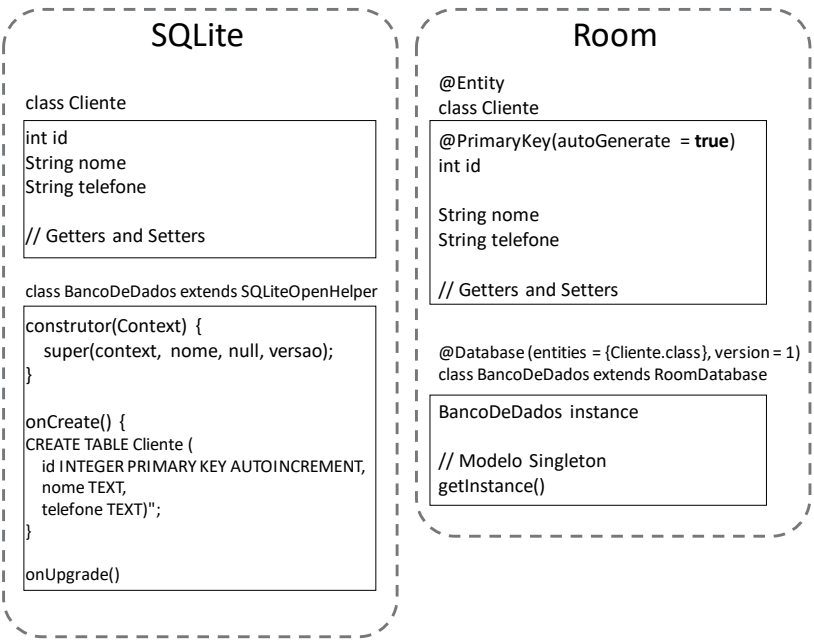
Reflita

Segundo Android (2018), a biblioteca Room fornece armazenamento local de dados através do mapeamento de objetos.

Durante o desenvolvimento de um aplicativo Android, quais benefícios a biblioteca Room fornece em relação ao SQLite?

Vamos finalizar esta seção analisando a Figura 3.10, que ilustra a estrutura de um projeto Android utilizando SQLite e Room.

Figura 3.10 | Estrutura de projeto Android com SQLite e Room



Fonte: elaborada pelo autor.

Finalizamos este item com duas estruturas de projeto Android para trabalharmos com o banco de dados relacional.

1. Ao utilizarmos SQLite, teremos:

a) Uma classe Java que representará os dados que serão armazenados.

b) Outra classe Java que estende um objeto *SQLiteOpenHelper*. Esta classe é responsável por criar e manter o controle de versão do banco de dados, criar a tabela e os campos através da linguagem SQL e realizar ajustes através do método *onUpgrade()*, caso a versão do banco de dados seja alterada.

2. Ao utilizarmos a API Room, teremos:

a) Uma classe Java que representará os dados e a tabela que serão armazenados.

b) Uma classe Java que estende um objeto *RoomDatabase*. Esta classe é responsável por fornecer uma instância de acesso ao banco de dados.

Na próxima seção estudaremos como incluir, excluir e pesquisar registros na tabela do banco de dados. Até lá!

Sem medo de errar

É hora de executarmos a segunda etapa de desenvolvimento do jogo para Android de perguntas e respostas para a rede de ensino. Nesta fase construiremos a estrutura em que as perguntas e as respostas serão armazenas e também o banco de dados. Você poderá optar pela API SQLite ou pela biblioteca Room para desenvolver esta etapa.

Caso opte por utilizar SQLite, consulte o código-fonte disponível para download. Os passos a seguir referem-se à biblioteca Room.

1. Abra o projeto no Android Studio criado na seção anterior.

2. Devemos incluir a biblioteca Room no projeto, portanto, acesse o arquivo *build.gradle (Module: app)* e insira-a conforme código abaixo. Após modificar o arquivo *build.gradle (Module: app)*, sincronize o projeto.

```
1.    apply plugin: 'com.android.application'
2.
3.    android {
4.      compileSdkVersion 27
5.      defaultConfig {
6.        applicationId "com.rededeensino.
jogodeperguntaserespostas"
7.        minSdkVersion 19
8.        targetSdkVersion 27
9.        versionCode 1
10.       versionName "1.0"
11.       testInstrumentationRunner "android.
support.test.runner.AndroidJUnitRunner"
12.     }
13.     buildTypes {
14.       release {
15.         minifyEnabled false
16.         proguardFiles
getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
17.       }
18.     }
19.   }
20.
21.   dependencies {
22.     implementation fileTree(dir: 'libs',
include: ['*.jar'])
23.     implementation 'com.android.
support:appcompat-v7:27.1.1'
24.     implementation 'com.android.support.
constraint:constraint-layout:1.1.0'
25.     implementation 'com.android.
support:support-v4:27.1.1'
26.
```

```
27.     def room_version = "1.1.0"
28.     implementation "android.arch.persistence.
room:runtime:$room_version"
29.     annotationProcessor "android.arch.
persistence.room:compiler:$room_version"
30.
31.     testImplementation 'junit:junit:4.12'
32.     androidTestImplementation 'com.android.
support.test:runner:1.0.2'
33.     androidTestImplementation 'com.android.
support.test.espresso:espresso-core:3.0.2'
34.   }
```

3. Para armazenarmos as perguntas e as respostas, vamos criar uma classe Java com o nome de `Questoes`.

4. Em seguida, insira três atributos privados nesta classe: i) o primeiro do tipo inteiro com o nome de `id`, que será referência para a chave primária; ii) o segundo atributo do tipo *String* com o nome de `pergunta`; iii) e, por fim, outro também do tipo *String* com o nome de `resposta`.

5. Após definir os três atributos, insira os *Getters* e *Setters* para cada atributo. Você poderá criá-los automaticamente no Android Studio através das teclas de atalho ALT + INSERT.

6. Para finalizar esta classe, insira a anotação *@Entity* ao declarar a classe e a anotação `@PrimaryKey(autoGenerate = true)` para o atributo `id`.

7. Agora é hora de criarmos outra classe Java responsável por fornecer acesso ao banco de dados. Crie uma classe Java abstrata que estenda um objeto *RoomDatabase*.

8. Adicione a anotação *@Database* e forneça como parâmetros as entidades que devem ser criadas e a versão do banco de dados.

9. Em seguida, defina um atributo privado e estático que faça referência a esta classe.

10. Crie um método com o padrão de projeto *Singleton* responsável por fornecer o único acesso ao banco de dados no aplicativo. O código abaixo apresenta a classe *BancoDeDados.java* completa.

```
1.  package com.rededeensino.
    jogodeperguntaserespostas.dados;
2.
3.  import android.arch.persistence.room.Database;
4.  import android.arch.persistence.room.Room;
5.  import android.arch.persistence.room.
    RoomDatabase;
6.  import android.content.Context;
7.
8.  @Database(entities = {Questoes.class},
    version = 1)
9.  public abstract class BancoDeDados extends
    RoomDatabase {
10.
11.   // Instância única para o Banco de Dados
12.   private static BancoDeDados INSTANCE;
13.
14.   // Modelo Singleton
15.   public static BancoDeDados
    getBancoDeDados(final Context context) {
16.     if (INSTANCE == null) {
17.       synchronized (BancoDeDados.class) {
18.         if (INSTANCE == null) {
19.           // Cria o banco de dados
20.           INSTANCE = Room.
    databaseBuilder(context.
    getApplicationContext(),
21.                   BancoDeDados.class, "bd_
    questoes")
22.                   .allowMainThreadQueries()
23.                   .build();
24.         }
25.       }
26.     }
27.     return INSTANCE;
```

```
28.     }
29.  }
```

Finalizamos aqui a segunda etapa do aplicativo Android para a rede de ensino.

Você poderá verificar uma opção de código-fonte para o problema proposto acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u3/s2/codigo_1_ddm.pdf> ou por meio do QR Code.

Na próxima seção focaremos em inserir as questões no banco de dados e exibi-las para os alunos. Até lá!

Avançando na prática

Aplicativo para contatos de clientes

Descrição da situação-problema

Uma empresa especializada em marketing digital deseja disponibilizar um aplicativo Android apenas para os seus colaboradores, com o objetivo de manter armazenada a ficha cadastral de cada cliente. O aplicativo deve armazenar o nome do cliente e as informações de contato, tais como e-mail, celular e endereços de redes sociais. Você foi contratado para atuar com a equipe de programadores que vai desenvolver o aplicativo Android e sua função é criar a estrutura do banco de dados. Ao finalizar, apresente o projeto para o restante da equipe.

Resolução da situação-problema

Após ser contratado para atuar em uma equipe de programadores, foi-lhe delegada a tarefa de criar a estrutura do banco de dados para um aplicativo Android, responsável por armazenar informações sobre os clientes de uma empresa de marketing digital. Para finalizar sua etapa, siga os passos a seguir:

1. Crie um projeto no Android Studio com uma *Empty Activity*.
2. Adicione a biblioteca Room no arquivo *build.gradle (Module: app)*.

3. Crie uma classe Java com o nome de Cliente que representará a tabela no banco de dados. Ao declarar a classe, insira a anotação *@Entity*.

4. Declare os atributos que deseja armazenar no banco de dados, por exemplo: nome, e-mail, WhatsApp, LinkedIn, Twitter, GitHub, YouTube, Snapchat, Facebook e Instagram. Não se esqueça de declarar um atributo id inteiro que representará a chave primária para cada cliente armazenado no banco de dados.

5. Crie os *Getters* e *Setters* para todos os atributos.

6. Em seguida, crie uma classe Java abstrata que estenda um objeto *RoomDatabase*.

7. Insira a anotação *@Database* com os parâmetros referentes às entidades e à versão do banco de dados.

8. Declare um objeto privado e estático referente a esta classe.

9. Crie um método no padrão de projeto *Singleton*, responsável por fornecer acesso ao banco de dados.

Após finalizar toda a estrutura da tabela e do banco de dados, apresente para a equipe de programadores.

Você poderá verificar uma opção de código-fonte para o problema proposto acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u3/s2/codigo_2_ddm.pdf> ou por meio do QR Code.

Faça valer a pena

1. A biblioteca Room permite que aplicativos Android armazenem dados estruturados no dispositivo móvel. Através dela é possível mapear os objetos para que sejam armazenados. Outra característica importante desta biblioteca é que deve ser criada uma classe como ponto de entrada para acessar o banco de dados.

Quais são as características presentes em uma classe que representa o ponto de entrada para o banco de dados utilizando a biblioteca Room?

a) A classe deve estender um objeto *RoomDatabase*.
b) A classe deve ser abstrata e estender um objeto *RoomDatabase*.

c) A classe deve estender um objeto *RoomDatabase* e possuir a anotação *@Database* com os parâmetros referentes às entidades e à versão do banco de dados.
d) A classe deve ser abstrata e possuir a anotação *@Database* com os parâmetros referentes às entidades e à versão do banco de dados.
e) A classe deve ser abstrata, estender um objeto *RoomDatabase* e possuir a anotação *@Database* com os parâmetros referentes às entidades e à versão do banco de dados.

2. Segundo Android (2018), a biblioteca Room fornece acesso ao banco de dados, garantindo todo o poder da API do SQLite. Para que informações possam ser armazenadas no banco de dados, é necessário mapear os objetos que estão representados em classes Java através de anotações. Uma classe pode representar uma tabela no modelo relacional de armazenamento de dados.

Qual é a anotação responsável para identificar um objeto Java como uma tabela no banco de dados relacional?

a) *@Entity*.
b) *@Table*.
c) *@Database*.
d) *@PrimaryKey*.
e) *@TableName*.

3. SQLite é uma poderosa API para trabalhar com dados estruturados no Android. Para utilizar esta API, é necessário criar uma classe Java que estende um objeto *SQLiteOpenHelper*. Os objetivos desta classe são:
I) Criar o banco de dados.
II) Manter o controle de versão do banco de dados.
III) Chamar pelo método *onCreate()* responsável por criar a tabela e os campos através de comandos SQL.
IV) Chamar pelo método *onUpgrade()* caso a versão do banco de dados seja alterada.

Qual alternativa apresenta os objetivos da classe que estende um objeto *SQLiteOpenHelper*?

a) Apenas as alternativas I, III e IV estão corretas.
b) Apenas as alternativas I, II e III estão corretas.
c) Apenas as alternativas II, III e IV estão corretas.
d) Apenas as alternativas I e III estão corretas.
e) Todas as alternativas estão corretas.

Seção 3.3

Trabalhando com banco de dados local

Diálogo aberto

Caro aluno, seja bem-vindo à última seção desta unidade. Hoje você já adicionou algum contato à sua agenda? Excluiu? Atualizou algum número? Ou então pesquisou por algum contato? Todas essas ações representam manipulações com o banco de dados. Para desenvolver um aplicativo que forneça estas opções para o usuário, estudaremos nesta seção como criar funcionalidades de inserção, exclusão, atualização e pesquisa no aplicativo Android através da API SQLite e da biblioteca Room.

Dando continuidade no desenvolvimento do jogo para a rede de ensino, é altamente recomendável determinar um padrão de acesso para manipular o banco de dados. Na seção anterior nós criamos a entidade e o banco de dados. A entidade representa as questões, que são compostas de perguntas e respostas. O banco de dados é responsável por armazenar todas as questões. Também definimos a conexão com o banco de dados.

Agora é hora de adicionarmos as funcionalidades para inserção e exibição das questões. Implemente as funcionalidades para cadastrar e exibir uma questão por vez para o usuário. Lembre-se de adicionar todas as que foram propostas no layout. No *JogarFragment.java*, você deve implementar o botão "Pular de Questão", responsável por exibir a próxima questão cadastrada no banco de dados. Como programador, você poderá definir se as questões serão exibidas aleatoriamente ou uma após a outra, conforme cadastrado.

Para finalizar o terceiro aplicativo, criaremos nesta seção um Objeto de Acesso a Dados, responsável por fornecer métodos para inserir, excluir, atualizar e pesquisar no banco de dados. As dicas estão dadas, então agora é hora de trabalhar. Desenvolva todas as funcionalidades solicitadas. Bons estudos!

Não pode faltar

Para trabalharmos com banco de dados local no Android, devemos criar três elementos em nossos aplicativos:

1. Entidade.
2. Banco de dados.
3. Objeto de Acesso a Dados.

Um Objeto de Acesso a Dados é representado pelo acrônimo DAO, com origem do inglês, *Data Access Object*. O objetivo do DAO é fornecer para o desenvolvedor métodos responsáveis por manipular os dados no banco.

Assimile

Sempre que o desenvolvedor desejar manipular os dados no banco de dados, é recomendável utilizar o padrão DAO. Este objeto deve oferecer métodos para inserir, excluir, atualizar e até mesmo realizar consultas no banco de dados.

O Quadro 3.4 apresenta os passos necessários para criar o DAO utilizando a API SQLite e a biblioteca Room. Ao longo do conteúdo vamos implementar esta classe, para que ela forneça métodos de acesso ao banco de dados.

Quadro 3.4 | Comparação entre SQLite e Room para criar o DAO

SQLite	1. Crie uma classe Java que representará o DAO.
Room	1. Crie uma classe Java abstrata. 2. Adicione a anotação *@Dao* ao declarar a classe.

Fonte: elaborado pelo autor.

Após criar a classe DAO, é necessário declarar os métodos para manipular o banco de dados. Nesta seção serão apresentados quatro cenários de manipulação de dados com o banco de dados. Cada cenário exemplifica a implementação da classe DAO utilizando a API SQLite e a biblioteca Room. Ao desenvolver um aplicativo Android, você poderá escolher qual implementação seguir.

Cenário 1: **inserir dados no banco de dados.** Vamos inserir um registro na tabela de Clientes do banco de dados. Acesse a classe referente ao DAO e siga os passos a seguir.

SQLite:

1. Declare um método com o nome *inserirCliente* e com as seguintes características:

a) Público.

b) Estático.

c) Que retorne um valor do tipo long, que representa o ID do Cliente após ser inserido no banco de dados.

d) Receba como parâmetro um objeto `Context` e um objeto `Cliente`.

Dentro do método `inserirCliente(Context, Cliente)` devemos seguir três passos:

- Recuperar uma instância do banco de dados.
- Criar um objeto do tipo *ContentValues* responsável por armazenar todos os dados que serão adicionados no banco de dados.
- Chamar pelo método responsável por inserir no banco de dados.

De acordo com Android (2018), um objeto *ContentValues* é usado para armazenar um conjunto de valores. Este objeto trabalha com a estrutura de *key-value* (chave-valor). Analise o código abaixo completo e os comentários correspondente ao método `inserirCliente()`.

```
  public static long inserirCliente(Context context,
Cliente cliente) {

    // Criamos uma instância da classe do banco de
dados
     BancoDeDadosSQLite bancoDeDadosSQLite = new
BancoDeDadosSQLite(context);
```

```
    // Recuperamos um objeto para escrever no banco de dados
            SQLiteDatabase sqLiteDatabase = bancoDeDadosSQLite.getWritableDatabase();

    // Este objeto é responsável por enviar os dados para o banco de dados
    ContentValues values = new ContentValues();
    values.put("nome", cliente.getNome());
    values.put("telefone", cliente.getTelefone());

    // Chamamos pelo método responsável por inserir no banco de dados
    // O primeiro parâmetro refere-se a tabela
     // O segundo parâmetro é nulo, pois estamos informando os dados através do objeto ContentValues
    // O terceiro parâmetro refere-se aos dados que serão adicionados
    return sqLiteDatabase.insert(
        "Cliente",
        null,
        values);
}
```

Room:

1. Declare um método com o nome *inserirCliente* e com as seguintes características:

a) Público.

b) Abstrato.

c) Que retorne um valor do tipo long, que representa o ID do Cliente após ser inserido no banco de dados.

d) Receba como parâmetro um objeto do tipo Cliente.

2. Adicione a anotação *@Insert* ao declarar o método.

A biblioteca Room executará o que for necessário para incluir o Cliente no banco de dados.

```
@Insert
public abstract long inserirCliente(Cliente cliente);
```

Cenário 2: **excluir dados do banco de dados.** Vamos excluir um registro da tabela de Clientes do banco de dados. Acesse a classe referente ao DAO e siga os passos a seguir.

SQLite:

1. Declare um método com o nome *excluirCliente* e com as seguintes características:

a) Público.

b) Estático.

c) Que retorne um valor do tipo int, que representa a quantidade de linhas excluídas.

d) Receba como parâmetro um objeto `Context` e um objeto `Cliente`.

Em SQL, para excluir um registro devemos executar o comando `DELETE FROM nome_da_tabela WHERE campo_id = id_do_cliente;`

Devemos implementar o método `excluirCliente()` para que apresente o mesmo comportamento do comando SQL. Dentro do método `excluirCliente(Context, Cliente)`, devemos seguir quatro passos:

- Recuperar uma instância do banco de dados.
- Definir a cláusula WHERE do comando SQL.
- Declarar um vetor de Strings com os argumentos da cláusula WHERE.
- Chamar pelo método responsável por excluir no banco de dados.

```
  public static int excluirCliente(Context context,
Cliente cliente) {

     // Criamos uma instância da classe do banco
de dados
      BancoDeDadosSQLite bancoDeDadosSQLite = new
BancoDeDadosSQLite(context);

    // Recuperamos um objeto para escrever no banco
de dados
            SQLiteDatabase     sqLiteDatabase     =
bancoDeDadosSQLite.getWritableDatabase();

    // Responsável por definir a cláusula WHERE
    String where = "id = ?";

    // Declara um vetor de Strings
    // Recebe como argumento o id do Cliente
    // Converte o id para String
    String[] argumentos = {String.valueOf(cliente.
getId())};

    // Chamamos pelo método responsável por excluir
no banco de dados
    // O primeiro parâmetro refere-se a tabela
     // O segundo parâmetro refere-se a cláusula
WHERE
    // O terceiro parâmetro refere-se aos argumentos
da cláusula WHERE
    return sqLiteDatabase.delete(
        "Cliente",
        where,
        argumentos);
  }
```

Room:

1. Declare um método com o nome *excluirCliente* e com as seguintes características:

a) Público.

b) Abstrato.

c) Que retorne um valor do tipo int, que representa a quantidade de linhas excluídas.

d) Receba como parâmetro um objeto do tipo Cliente.

2. Adicione a anotação *@Delete* ao declarar o método.

```
@Delete
public abstract int excluirCliente(Cliente cliente);
```

Cenário 3: **atualizar dados do banco de dados.** Vamos atualizar um registro na tabela de Clientes do banco de dados. Acesse a classe referente ao DAO e siga os passos a seguir.

SQLite:

1. Declare um método com o nome *atualizarCliente* e com as seguintes características:

a) Público.

b) Estático.

c) Que retorne um valor do tipo int, que representa a quantidade de linhas atualizadas.

d) Receba como parâmetro um objeto `Context` e um objeto `Cliente`.

Dentro do método `atualizarCliente(Context, Cliente)` devemos seguir cinco passos:

- Recuperar uma instância do banco de dados.
- Declarar um objeto *ContentValues* responsável por informar os novos valores que serão atualizados no banco de dados.

- Definir a cláusula WHERE do comando SQL.
- Declarar um vetor de Strings com os argumentos da cláusula WHERE.
- Chamar pelo método responsável por atualizar no banco de dados.

```
  public static int atualizarCliente(Context context, Cliente cliente) {

     // Criamos uma instância da classe do banco de dados
      BancoDeDadosSQLite bancoDeDadosSQLite = new BancoDeDadosSQLite(context);

    // Recuperamos um objeto para escrever no banco de dados
            SQLiteDatabase sqLiteDatabase = bancoDeDadosSQLite.getWritableDatabase();

    // Este objeto é responsável por enviar os dados para o banco de dados
     ContentValues values = new ContentValues();
     values.put("nome", cliente.getNome());
     values.put("telefone", cliente.getTelefone());

     // Responsável por definir a cláusula WHERE
     String where = "id = ?";

     // Declara um vetor de Strings
     // Recebe como argumento o id do Cliente
     // Converte o id para String
     String[] argumentos = {String.valueOf(cliente.getId())};
```

```
    // Chamamos pelo método responsável por atualizar
no banco de dados
     // O primeiro parâmetro refere-se a tabela
     // O segundo parâmetro refere-se aos dados do
cliente que será atualizado
    // O terceiro parâmetro refere-se a cláusula WHERE
     // O quarto parâmetro refere-se aos argumentos
da cláusula WHERE
     return sqLiteDatabase.update(
         "Cliente",
         values,
         where,
         argumentos);
   }
```

Room:

1. Declare um método com o nome *atualizarCliente* e com as seguintes características:

a) Público.

b) Abstrato.

c) Que retorne um valor do tipo int, que representa a quantidade de linhas atualizadas.

d) Receba como parâmetro um objeto do tipo Cliente.

2) Adicione a anotação *@Update* ao declarar o método.

```
   @Update
   public abstract int atualizarCliente(Cliente
cliente);
```

Cenário 4: **pesquisar todos os registros do banco de dados.** Vamos pesquisar todos os registros da tabela Clientes do banco de dados. Acesse a classe referente ao DAO e siga os passos a seguir.

SQLite:

1. Declare um método com o nome *pesquisarTodosCliente* e com as seguintes características:

a) Público.

b) Estático.

c) Que retorne um objeto do tipo Cursor.

d) Receba como parâmetro um objeto `Context`.

Dentro do método `pesquisarTodosCliente(Context)` devemos seguir dois passos:

- Recuperar uma instância do banco de dados.
- Chamar pelo método responsável por pesquisar no banco de dados.

Em SQL, para pesquisar um registro, devemos executar o comando `SELECT colunas FROM nome_da_databela WHERE condição GROUP BY colunas HAVING condição ORDER BY colunas;`

Caso forneça `null` como parâmetro, todos os registros serão pesquisados.

```
publicstatic Cursor pesquisarTodosClientes(Context context) {

    // Criamos uma instância da classe do banco de dados
     BancoDeDadosSQLite bancoDeDadosSQLite = new BancoDeDadosSQLite(context);

    // Recuperamos um objeto para ler do banco de dados
          SQLiteDatabase sqLiteDatabase = bancoDeDadosSQLite. getReadableDatabase();
```

```
    return sqLiteDatabase.query(
        table:   "Cliente",
        colunas:  null,
        where:    null,
        argumentos: null,
        groupBy:  null,
        having:   null,
        orderBy:  null);
}
```

Room:

1) Declare um método com o nome *pesquisarTodosClientes()* com as seguintes características:

a) Público.

b) Abstrato.

c) Que retorne um objeto `List<Cliente>`.

2) Adicione a anotação *@Query* ao declarar o método e receba como parâmetro o comando SQL com a consulta desejada.

```
@Query("SELECT * FROM Cliente")
public abstract List<Cliente> pesquisarTodos();
```

A seguir é exemplificado como pesquisar um Cliente pelo atributo `nome`. Observe que:

1. O método recebe como parâmetro uma String referente ao nome que se deseja pesquisar.

2. A cláusula WHERE foi adicionada no comando SQL.

3. O argumento da cláusula WHERE é identificado pelo sinal de dois pontos, seguido pelo mesmo nome do parâmetro recebido no método.

```
@Query("SELECT * FROM Cliente WHERE nome LIKE
:nome")
```

```
public abstract List<Cliente> pesquisarPorNome
(String nome);
```

Reflita

A biblioteca Room apresenta muitas vantagens em relação à API SQLite. Uma característica importante da biblioteca Room é que a linguagem SQL é verificada em tempo de compilação. Caso o programador forneça o nome de alguma tabela que não existe, o aplicativo não será compilado.

Por outro lado, a API SQLite não verifica a linguagem SQL em tempo de compilação. Qual será o comportamento de um aplicativo Android caso o programador forneça o nome de uma tabela que não existe no banco de dados?

Finalizamos aqui os quatro cenários que ilustram como criar os métodos responsáveis por inserir, excluir, atualizar e pesquisar no banco de dados. Ao utilizar a biblioteca Room, não é possível instanciar um objeto DAO, pois é uma classe abstrata. Para contornar esta situação, Android (2018) nos indica os seguintes passos:

1. Acesse a classe do banco de dados que estende o objeto *RoomDatabase*.
2. Declare um método abstrato que retorne um objeto DAO.

```
@Database(entities = {Cliente.class}, version =
1)
public abstract class BancoDeDadosRoom extends
RoomDatabase {

    // Através deste método é possível recuperar o
objeto DAO
    public abstract MyDAORoom getDAO();

    // Instância única para o Banco de Dados
```

```
    private static BancoDeDadosRoom INSTANCE;
    // Modelo Singleton
    public  static BancoDeDadosRoom getBancoDeDados
(final Context context) {
       if (INSTANCE == null) {
         synchronized (BancoDeDadosRoom.class) {
           if (INSTANCE == null) {
             // Cria o banco de dados
            INSTANCE = Room.databaseBuilder(context.
getApplicationContext(),
                    BancoDeDadosRoom.class, "cliente_
database")
                   .allowMainThreadQueries()
                   .build();
           }
         }
       }
       return INSTANCE;
    }
  }
```

Exemplificando

O código abaixo exemplifica como cadastrar um Cliente no banco de dados após o usuário digitar o nome e o telefone em um *Fragment*.

SQLite:

```
 String nome = mEditTextNome.getText().toString();

    String  telefone  =  mEditTextTelefone.getText().
toString();
```

```
  if ((!nome.isEmpty()) && (!telefone.isEmpty())) {
    Cliente cliente = new Cliente(nome, telefone);

    MyDAOSQLite.inserir(getActivity(), cliente);
  }
```

Room:

```
 String nome = mEditTextNome.getText().toString();
    String telefone = mEditTextTelefone.getText().
toString();

  if ((!nome.isEmpty()) && (!telefone.isEmpty())) {
    Cliente cliente = new Cliente(nome, telefone);

    BancoDeDadosRoom.getBancoDeDados(getActivity())
      .getDAO()
      .inserirCliente(cliente);
  }
```

Nesta seção estudamos como manipular dados que estão armazenados localmente em um aplicativo Android. Conhecemos os passos exigidos para a criação de um Objeto de Acesso a Dados utilizando a API SQLite e a biblioteca Room.

Pesquise mais

A biblioteca Room está em constante atualização. Para conhecer mais recursos disponíveis desta biblioteca, você poderá consultar a documentação oficial:

ANDROID Developers. **Save data in a local database using Room.** Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2Mrijvq>. Acesso em: 26 jul. 2018.

Finalizamos a estrutura de um aplicativo Android que oferece funcionalidades para armazenamento local de dados estruturados. Pratique o conteúdo apresentado para que você possa implementar aplicativos Android utilizando SQLite ou Room.

Sem medo de errar

É hora de finalizarmos o aplicativo para a rede de ensino. Você e a equipe de programadores devem adicionar as funcionalidades para inserção e exibição das questões. Nesta última etapa também será necessário implementar a funcionalidade para pular de questões. As dicas abaixo se referem à implementação do aplicativo utilizando a biblioteca Room.

1. Abra o projeto do jogo de perguntas e respostas no Android Studio.
2. Crie uma classe Java abstrata com o nome de DAO.
3. Adicione a anotação *@Dao* ao declarar a classe.
4. Nesta classe será necessário incluir dois métodos abstratos.
5. Declare um método público e abstrato, que retorne um valor do tipo long, com o nome de *inserirQuestoes* e receba como parâmetro um objeto `Questoes`. Adicione a anotação *@Insert* ao declarar o método.
6. Declare um método público e abstrato, que retorne uma `List<Questoes>`, com o nome de *pesquisarTodasQuestoes*. Adicione a anotação *@Query* e insira como parâmetro o comando SQL para pesquisar todas as perguntas.

Observe o código completo abaixo referente à classe DAO:

```
package com.rededeensino jogodeperguntaserespostas.dados;

import android.arch.persistence.room.Dao;
```

```
import android.arch.persistence.room.Insert;
import android.arch.persistence.room.Query;
import java.util.List;

@Dao
public abstract class DAO {

    @Insert
    public abstract long inserirQuestao(Questoes
questoes);

    @Query("SELECT * FROM Questoes")
            public abstract List<Questoes>
pesquisarTodasQuestoes();
}
```

7. Em seguida, vamos criar um método público e abstrato, que retorne um objeto DAO na classe *BancoDeDados.java*.

É hora de implementar os nossos *Fragments*.

8. Acesse a classe *CadastrarFragment.java*. Crie um *Listener* para o botão inserir, que será responsável por inserir as perguntas no banco de dados.

- Recupere a pergunta e a resposta digitada no *EditText*.
- Verifique se não estão vazias.
- Adicione no banco de dados.
- Após inserir, você poderá limpar os *EditText* e informar o usuário através de uma mensagem que a questão foi inserida.

```
// Recupera os valores digitados pelo usuário
String pergunta = mEditTextPergunta.getText().
toString();
String resposta = mEditTextResposta.getText().
toString();

// Verifica se não estão vazios
```

```
  if    ((!pergunta.isEmpty())    &&    (!resposta.
isEmpty())) {
        // Cria um objeto do tipo Questoes com os
valores digitados pelo usuário
        Questoes questoes = new Questoes(pergunta,
resposta);

      // Através da classe DAO, insere a Questão no
banco de dados
      BancoDeDados.getBancoDeDados(getActivity())
              .getDAO()
              .inserirQuestao(questoes);

      // Limpa os EditText após inserir a questao
      mEditTextPergunta.setText("");
      mEditTextResposta.setText("");

      // Exibe uma mensagem para o usuário
       Toast.makeText(getActivity(), "Inserido com
sucesso!", Toast.LENGTH_SHORT).show();
  }
```

9. Acesse a classe *JogarFragment.java*.

- Declare um objeto do tipo List<*Questoes*>.
- Declare os métodos *exibirResposta()* e *proximaQuestao()*.
- Crie um *Listener* para cada botão disponível no *Fragment* e chame pelo método correspondente. O botão Pular de Questão chamará pelo método *proximaQuestao()*. O botão Exibir Resposta chamará pelo método *exibirResposta()*.
- O método *exibirResposta()* é responsável por tornar visível para o usuário o *TextViewResposta*.
- No método *OnActivityCreated()*, recupere todas as questões armazenadas no banco de dados através da classe DAO e armazene no objeto *List<Questoes>*. Em seguida, chame pelo método

proximaQuestao() para apresentar uma questão para o usuário ao iniciar o aplicativo.

```
// Recupera todas as questões cadastradas no banco de dados
mListQuestoes=BancoDeDados getBancoDeDados (getActivity()).getDAO().pesquisarTodasQuestoes();

// Exibe uma pergunta ao iniciar o aplicativo
proximaQuestao();
```

- O método *proximaQuestao()* é responsável por recuperar uma questão do objeto *List* e exibi-la no *TextViewPergunta* e *TextViewResposta* do *Fragment*.

```
private void proximaQuestao(){
  // Verifica se a lista contém questões cadastradas
   if (!mListQuestoes.isEmpty()){
     // Recupera o total de questões cadastradas
     int totalDeQuestoes = mListQuestoes.size();

     // Cria um número aleatório dentre o total de questões cadastradas
          int indexAleatorio = new Random().nextInt(totalDeQuestoes);

      // Recupera um objeto Questoes da lista de forma aleatória
           Questoes questoes = mListQuestoes.get(indexAleatorio);

     // Exibe para o usuário a pergunta e a resposta
               mTextViewPergunta.setText(questoes.getPergunta());
               mTextViewResposta.setText(questoes.getResposta());
```

```
        // Mantém a resposta oculta
        mTextViewResposta.setVisibility(View.GONE);
    }
}
```

Finalizamos aqui o aplicativo de perguntas e respostas para a rede de ensino. Você poderá fazer alterações no aplicativo e sugerir novas versões para a equipe de programadores.

	Você poderá verificar uma opção de código-fonte para o problema proposto acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u3/s3/codigo_1.pdf> ou por meio do QR Code.

Explore as novas possibilidades através do armazenamento local no Android e seja criativo!

Avançando na prática

Aplicativo Android de anotações

Descrição da situação-problema

Você foi contratado para atuar com a equipe de programadores de uma empresa e deverá realizar a manutenção em um aplicativo Android que armazena localmente anotações do usuário. A empresa deseja atualizar a versão do aplicativo. Além de armazenar a anotação, a companhia deseja disponibilizar um recurso de consulta de anotações já armazenadas no dispositivo. Você deverá entregar um projeto no Android Studio com a funcionalidade solicitada para que a equipe de programadores possa dar continuidade na atualização da versão do aplicativo.

Resolução da situação-problema

Para entregar a nova funcionalidade para a empresa que o contratou, você deverá criar um projeto no Android Studio. Em seguida, siga as dicas abaixo para implementar a nova versão do aplicativo Android:

1. Insira no arquivo *build.gradle (Module: app)* a biblioteca Room.

2. Crie uma classe Java e insira a anotação *@Dao* ao declarar a classe.

3. Declare um método público e abstrato, que retorne um objeto `List<Anotacao>` e receba como parâmetro uma String referente à consulta que será realizada.

4. Insira a anotação *@Query* ao declarar este método e adicione o comando SQL para que a biblioteca Room consiga realizar a consulta.

```
@Query("SELECT * FROM Anotacao WHERE descricao LIKE :consulta")
public abstract List<Questoes> pesquisar(String consulta);
```

Após finalizar a classe DAO, apresente o projeto para que a equipe de programadores possa finalizar a atualização do aplicativo.

Faça valer a pena

1. Um aplicativo Android que trabalha com armazenamento de dados deve disponibilizar meios para que o usuário consiga inserir, excluir, atualizar e pesquisar dados no banco de dados. Ao trabalhar com a biblioteca Room para armazenar dados localmente, o programador deve disponibilizar métodos para que os dados possam ser manipulados através do Objeto de Acesso a Dados - DAO.

Assinale a alternativa que apresenta o método responsável por retornar um Objeto de Acesso a Dados utilizando a biblioteca Room.

a) *return DAO;*.
b) *public abstract DAO getDao();*.

c) *public DAO getDAO();*.
d) *public getDAO() { return DAO; }*.
e) *public static getDao();*.

2. O Objeto de Acesso a Dados, conhecido como DAO, é um padrão que fornece métodos responsáveis por manipular dados no banco de dados. No Android, a biblioteca Room exige a criação do componente DAO através da inserção de anotações em métodos abstratos. O código abaixo exemplifica a criação de um objeto de acesso a dados responsável por pesquisar todos os registros da tabela Colaboradores.

```
@Dao
public abstract class DAO {

    @Query("SELECT * FROM Colaboradores")
                public    abstract    List<Colaborador>
pesquisarTodosColaboradores();
}
```

Qual será o retorno do método *@Insert* caso receba apenas um parâmetro?

a) Retornará uma mensagem para que seja exibida para o usuário.
b) Retornará `true` caso o registro tenha sido incluído com sucesso.
c) Não haverá retorno, pois trata-se de um método de inserção.
d) Retornará o próprio objeto inserido para que o programador disponibilize para o usuário.
e) Retornará um valor do tipo long, que representa o ID do registro incluído.

3. Um aplicativo Android que utiliza a API SQLite para armazenar dados estruturados localmente deve possuir uma classe Java que estende um objeto *SQLiteOpenHelper*. O objetivo desta classe é criar a tabela e o banco de dados no dispositivo móvel. O programador também deve utilizar esta classe para obter referências de leitura e escrita do banco de dados. O código abaixo cria uma instância da classe do banco de dados.

```
BancoDeDados bd = new BancoDeDados(context);
```

Após declarar uma instância do banco de dados, quais os métodos responsáveis por obter referências de leitura e escrita, respectivamente, do banco de dados?

a) *db.getSQLiteDatabase();.*
b) *bd.getDatabase();.*
c) *bd.getWritableDatabase();.*
d) *bd.getReadableDatabase();.*
e) *bd.getReadableDatabase() e bd.getWritableDatabase();.*

Referências

ALVES, William Pereira. **Banco de Dados**. São Paulo: Érica, 2014.

ANDROID DEVELOPERS. **Android Jetpack:** Room. [S.l.], 4 jun. 2018. Disponível em: <https://bit.ly/2LEEhxp>. Acesso em: 27 jul. 2018.

_____. **DevBytes:** apresentando os Fragments (Portuguese). [S.l.], 18 set. 2013. Disponível em: <https://bit.ly/2NL8Ejc>. Acesso em: 26 jul. 2018.

ANDROID Developers. **Accessing data using Room DAOs**. Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2LUL1V1>. Acesso em: 27 jul. 2018.

_____. **Defining data using Room entities.** Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2NLEjB9>. Acesso em: 27 jul. 2018.

_____. **Fragmentos**. Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2LKTfPd>. Acesso em: 26 jul. 2018.

_____. **FrameLayout**. Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2NOLEQv>. Acesso em: 26 jul. 2018.

_____. **Save data in a local database using Room**. Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2Mrijvq>. Acesso em: 27 jul. 2018.

_____. **Save data using SQLite**. Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2NPvzdo>. Acesso em: 26 jul. 2018.

_____. **Visão geral dos recursos**. Android Developers, [s.l.], 2018. Disponível em: <https://bit.ly/2LOv965>. Acesso em: 26 jul. 2018.

ANDROID Studio. **Version 3.1.3.** [S.l.], 2018. Disponível em: <https://bit.ly/2HG8ZW1>. Acesso em: 26 jul. 2018.

CARDOSO, V.; CARDOSO, G. **Sistemas de banco de dados:** uma abordagem introdutória e aplicada. São Paulo: Saraiva, 2012.

DEITEL, P.; DEITEL, H.; WALD, A. **Android 6 para programadores:** uma abordagem baseada em aplicativos. 3. ed. Porto Alegre: Bookman, 2016.

GAMMA, Erich et al. **Padrões de Projeto:** soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2007.

Banco de dados na nuvem com Android

Convite ao estudo

Bem-vindo à última unidade do livro de Desenvolvimento para Dispositivos Móveis. Ao longo do livro, nós tivemos a oportunidade de estudarmos técnicas para o desenvolvimento de aplicativos Android. Todo o conhecimento adquirido o ajudará a desenvolver sua carreira profissional. Agora é hora de darmos mais um passo nos nossos estudos. É hora de nos conectarmos ao mundo!

Dicio (2018, [s. p.]) define "Informação" como uma reunião dos "dados sobre um assunto ou pessoa". Atualmente, aplicativos processam informações de seus usuários todo o momento. As informações estão armazenadas na nuvem e estão disponíveis sempre que o usuário precisar.

Nesta unidade, compreenderemos e aplicaremos técnicas para desenvolvermos um aplicativo Android com armazenamento de dados na nuvem.

Caro programador, você foi indicado por seus colegas de trabalho a participar de um projeto que desenvolverá um aplicativo Android para uma rede de academias. A rede possui unidades em diversas cidades no Brasil e os seus alunos têm direito de frequentar qualquer unidade como cortesia da assinatura do plano mensal. Para manter organização e controle do seu público, a rede deseja manter um cadastro de todos os seus alunos.

Você acompanhou o gerente do projeto até uma filial para entrevistar o responsável pela rede de academias e conseguiu anotar alguns detalhes que devem ser observados na hora

de desenvolver o aplicativo: qualquer unidade da rede terá autorização para cadastrar novos alunos. Os novos alunos devem aparecer no aplicativo de todas as unidades, ou seja, os dados devem estar atualizados em tempo real. Durante o diálogo com o responsável pela rede de academias, foi sugerido que algumas informações como nome, idade, altura, peso do aluno e unidade de cadastro devem, obrigatoriamente, ser armazenadas.

Após chegarem a um acordo entre o responsável pela rede de academias e a equipe do projeto, deve-se começar o desenvolvimento. Foram estabelecidas três etapas de desenvolvimento.

Para iniciar o projeto, o layout do aplicativo deverá ser construído. É importante que duas telas sejam disponibilizadas no aplicativo: a primeira tela deve exibir todos os alunos cadastrados e a segunda tela deve cadastrar alunos.

A segunda etapa de desenvolvimento deve ser focada na construção do banco de dados. Você deverá criar a estrutura necessária para que os alunos fiquem cadastrados no banco de dados em que todas as unidades da rede de academias terão acesso.

Para finalizar o aplicativo, implemente as funcionalidades necessárias para manipular os dados, como inclusão e atualização dos alunos.

As etapas de desenvolvimento estão definidas, contudo qual banco de dados será utilizado para que as informações dos alunos sejam compartilhadas em todas as unidades da rede de academias?

Para concluirmos esta unidade, na primeira seção, construiremos uma listagem de conteúdos utilizando o elemento RecyclerView. Na próxima seção, conheceremos a plataforma Firebase e o produto Realtime Database para criarmos um banco de dados na nuvem. Para finalizar, aprenderemos a manipular os dados no banco de dados na nuvem.

Seção 4.1

Desenvolvendo ui com Recyclerview

Diálogo aberto

Caro programador, seja bem-vindo à primeira seção da nossa última unidade! Após instalar um aplicativo de notícias em seu smartphone e abri-lo para conferir as últimas publicações, você observou que as publicações são exibidas uma após a outra. Existe uma listagem que você pode deslizar para cima e para baixo para encontrar novas publicações. Como desenvolvedor, você pensou: "como posso criar esse recurso de listagem no meu app?" A resposta dessa questão é o assunto central dessa seção.

Após ser contratado para desenvolver um aplicativo Android para uma rede de academias, foi delegada a você a primeira tarefa. Você programador deverá construir o layout com a listagem responsável por exibir todos os alunos cadastrados na rede de academias, bem como, o layout para cadastrar os alunos.

Para cada aluno exibido na listagem, devem ser exibidas as seguintes informações: nome, idade, altura, peso do aluno e unidade de cadastro. Certifique-se de implementar a listagem que exibirá todos os alunos utilizando os métodos corretos para que a listagem seja atualizada sempre que houver uma inclusão de aluno.

Após finalizar a primeira etapa de desenvolvimento, apresente a estrutura do projeto e o layout da listagem de conteúdo para o gerente do projeto.

Para concluirmos o desafio desta seção, conheceremos o elemento RecyclerView, responsável por exibir uma listagem de conteúdos dinamicamente, ou seja, cada item da listagem poderá ser modificado em tempo de execução. Conheça-o e explore os recursos disponíveis para construir aplicativos que forneçam uma listagem de conteúdos para os usuários. Bons estudos!

Não pode faltar

RecyclerView é um elemento *View* inserido no *layout* para exibir uma listagem de conteúdo. Por exemplo, você poderá exibir uma listagem com vários textos, uma listagem de músicas, fotos ou contatos.

Atenção

Você deve se atentar como o conteúdo da listagem será carregado. Por exemplo, você poderá carregar na sua *Activity* ou *Fragment* os dados que estão em um banco de dados –local ou na nuvem e exibi-los na listagem para o usuário.

O primeiro passo que devemos seguir para implementarmos um RecyclerView é definirmos quais dados serão exibidos na listagem. Portanto, você deverá declarar uma coleção de objetos e informar ao RecyclerView para exibir todos os objetos desta coleção. De acordo com Oracle (2018), o uso da Interface List permite ao programador criar uma coleção de objetos ordenados, ou seja, o programador tem controle sobre a ordem em que os itens são inseridos na coleção.

Nesta seção, para ilustrar a implementação do *RecylerView*, tomemos como exemplo uma coleção de produtos com a descrição e o preço de cada produto. Na Activity, será necessário declarar um atributo *List*<Produto> e então incluir os produtos na coleção.

```
1  package com.exemplo.unidadequatro;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import java.util.ArrayList;
6  import java.util.List;
7
8  public class MainActivity extends AppCompa-
   tActivity {
9
10   private List<Produto> mListProduto;
11
12   @Override
```

```
13    protected void onCreate(Bundle savedInstanceState) {
14        super.onCreate(savedInstanceState);
15        setContentView(R.layout.activity_main);
16
17        // Exemplificamos a criação de três objetos Produto para exibir na listagem
18        mListProduto = new ArrayList<>();
19        mListProduto.add(new Produto("Smartphone", 1800f));
20        mListProduto.add(new Produto("Tablet", 3600f));
21        mListProduto.add(new Produto("TV 50", 2100f));
22    }
23 }
```

Pesquise mais

De acordo com Oracle (2018), um *ArrayList* é uma coleção com tamanho ajustável de objetos, ou seja, o programador não precisa definir um tamanho fixo ao declarar um *ArrayList*. É possível acrescentar objetos no *ArrayList* no decorrer da implementação do código através do método add(). O programador tem acesso aos objetos inseridos no *ArrayList* a partir de seus índices. Para conhecer alguns métodos disponíveis de um objeto ArrayList, assista ao vídeo disponível em: <https://www.youtube.com/watch?v=6BxJrydcOk8>. Acesso em: 20 jun. 2018.

Após definir os dados que deseja exibir no RecyclerView, devemos criar um recurso de *layout* que representará cada item que será exibido na listagem. Portanto, siga os passos abaixo:

1) Clique com o botão direito sobre a pasta *res/layout*.

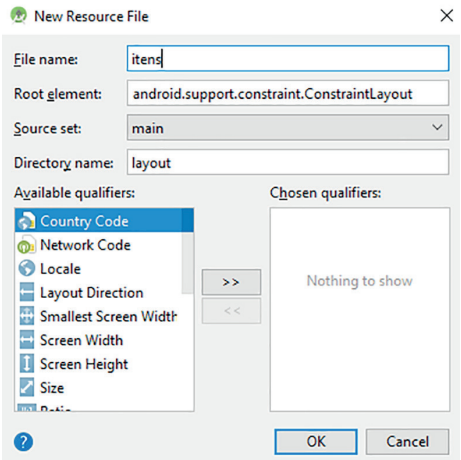
2) Selecione a opção ***New***, em seguida ***Layout resource file***. A janela "*New Resource File*" ("Novo arquivo de recurso") será carregada, conforme exibida na Figura 4.1.

3) Defina o nome "itens" para o arquivo no campo ***File name*** e pressione o botão ***OK***.

4) O elemento-pai desse recurso de *layout* deve possuir o atributo *android:layout_height* definido para *wrap_content*. É importante realizar o ajuste para definir a altura de cada item da listagem.

5) Para finalizar, devemos inserir dois elementos *TextView*. O primeiro *TextView* será responsável por exibir a descrição do produto e o segundo *TextView* será responsável por exibir o valor do produto.

Figura 4.1 | Janela "*New Resource File*"



Fonte: captura de tela do Android Studio.

O próximo passo é implementar o *RecyclerView*. De acordo com Developer (2018), a implementação do *RecyclerView* exige que alguns componentes trabalhem juntos para que os dados sejam exibidos para o usuário.

Vocabulário

Adapter é a classe responsável por fornecer cada item da listagem.

ViewHolder é a classe responsável por instanciar os elementos de layout que são exibidos em cada item.

Para desenvolver um aplicativo Android que utilize um *RecyclerView*, você deverá seguir os seguintes passos:

1) Insira o *RecyclerView* na seção *dependencies* do arquivo build.gradle (Module: app), conforme exibido na linha 30 do código

abaixo. Algumas linhas do código abaixo estão ocultas, pois você já conhece o arquivo build.gradle (Module: app). Note que a versão do *RecyclerView* é compatível com a configuração *compileSdkVersion*.

```
1   apply plugin: 'com.android.application'
2
3   android {
4       compileSdkVersion 27
13      // Linhas ocultas
19  }
20
21  dependencies {
22      // Linhas ocultas
28
29      // Dependência do RecyclerView. Observe que a versão está compatível com a configuração do compileSdkVersion--
30      implementation 'com.android.support:recyclerview-v7:27.1.1'
31  }
```

2) Insira o elemento <*RecyclerView*> no recurso de layout da *Activity* que deseja exibir a listagem de conteúdo. Em seguida, defina os seus atributos para ocupar todo o espaço do layout. O código abaixo representa o recurso de layout **activity_main.xml**.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.v7.widget.RecyclerView xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:id="@+id/rvListagem"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity" />
```

3) Crie uma classe java chamada Adaptador que estenda um objeto *RecyclerView.Adapter<Adaptador.NossoViewHolder>*. A classe Adaptador deve sobrepor os seguintes métodos:

a) *onCreateViewHolder*(): método responsável por inflar o layout criado no passo anterior. O layout será exibido em cada item.

b) *onBindViewHolder*(): método responsável por definir o conteúdo de cada item que será exibido na listagem. Se o RecyclerView conter dez objetos para serem exibidos, então este método será chamado automaticamente dez vezes para criar cada item da listagem.

c) *getItemCount*(): método responsável por informar ao Adapter a quantidade total de itens que devem ser exibidos.

Dica

Observe que o Android Studio aponta um erro na linha que declara a classe Adaptador. Essa linha apresenta um sublinhado em vermelho. Para corrigir, posicione o cursor sobre esta linha e pressione ALT + ENTER. Selecione a opção *Implement methods* para implementar os métodos *onCreateViewHolder*(), *onBindViewHolder*() e *getItemCount*().

4) Crie uma *inner class* estática chamada NossoViewHolder que estenda um objeto *RecyclerView.ViewHolder*. A *inner class* NossoViewHolder deve possuir um construtor que receba como parâmetro o elemento *View* e deve chamar pelo comportamento da classe pai.

Observe a estrutura abaixo das classes Adaptador e NossoViewHolder.

```
1 package com.exemplo.unidadequatro;
2
3 import android.support.annotation.NonNull;
4 import android.support.v7.widget.RecyclerView;
5 import android.view.View;
6 import android.view.ViewGroup;
7
```

```
8  public class Adaptador extends RecyclerView.
   Adapter<Adaptador.NossoViewHolder>{
9
10   @NonNull
11   @Override
12   public NossoViewHolder onCreateViewHolder(@
   NonNull ViewGroup parent, int viewType) {
13      // Este método deve retornar um objeto do
   tipo NossoViewHolder     return null;
14   }
15
16   @Override
17   public void onBindViewHolder(@NonNull
   NossoViewHolder holder, int position) {
18      // Este método recebe dois parâmetros
19      // O primeiro é um objeto NossoViewHolder
   que contém todos as referências do layout
20      // O segundo é a posição de cada item da
   listagem
21
22      // Se o RecyclerView conter dez objetos
   para serem exibidos, então este método será
   chamado automaticamente dez vezes para criar
   cada item da listagem.
23   }
24
25   @Override
26   public int getItemCount() {
27      // Este método retorna a quantidade de
   itens que deverá ser exibida na listagem
28      return 0;
29   }
30
31   // Esta é a inner class
32   public static class NossoViewHolder extends
   RecyclerView.ViewHolder {
33
```

```
34      public ViewHolder(View itemView) {
35        super(itemView);
36        // Aqui devem ser declarados todas as
  referências do arquivo itens.xml
37      }
38    }
39  }
```

Agora devemos começar a implementar o NossoViewHolder. Na *inner class* NossoViewHolder, devemos declarar os elementos do layout que serão exibidos em cada item da listagem. Como desejamos exibir a descrição e o preço do produto, temos um *TextView* para cada informação.

```
1  // Esta é a inner class
   public static class NossoViewHolder extends
2  RecyclerView.ViewHolder {
3
4    // Declaramos os objetos
5    TextView mTextViewDescricao, mTextViewPreco;
6
7    public ViewHolder(View itemView) {
8      super(itemView);
9
       // Aqui devem ser instanciadas todas as
10 referências do layout
       mTextViewDescricao = itemView.findViewByI-
11 d(R.id.textViewDescricao);
       mTextViewPreco = itemView.findViewById(R.
12 id.textViewPreco);
13   }
14 }
```

Vamos começar a implementar a classe Adaptador.

1) Acesse a classe Adaptador e declare um atributo privado que receberá os dados da listagem, definidos no início desta seção.

```
private List<Produto> mListProduto;
```

2) O método onCreateViewHolder() deve retornar o objeto NossoViewHolder que criamos.

a) Declare um objeto *View* que receba o recurso de layout criado para cada item da listagem.

b) Declare o objeto NossoViewHolder que criamos e passe como parâmetro no construtor do objeto, o objeto *view* recém-criado.

c) Retorne o *ViewHolder* que acabamos de criar

```
@NonNull
@Override
public NossoViewHolder onCreateViewHolder(@NonNull
ViewGroup parent, int viewType) {
    // Declaramos um objeto View com o layout que
será exibido para cada item
    View view = LayoutInflater.from(parent.getCon-
text()).inflate(R.layout.itens, parent, false);
    // Declaramos um objeto NossoViewHolder e passamos
como parâmetro o objeto View recém-criado
    NossoViewHolder viewHolder = new NossoViewHold-
er(view);
    return viewHolder;
}
```

3) O método *onBindViewHolder*() é responsável por vincular cada item da listagem com o layout. Observe que este método recebe como parâmetro o objeto NossoViewHolder que contém todos os elementos do layout e um valor inteiro que representa a posição de cada item da listagem.

```
@Override
publicvoidonBindViewHolder(@NonNullNossoViewHolder
holder, int position) {
    // Recuperamos um Produto do objeto List de
acordo com a posição recebida no parâmetro
    Produto produto = mListProduto.get(position);

    // Exibimos os dados do item para o usuário
através das referências criadas no NossoViewHolder
```

```
    holder.mTextViewDescricao.setText(produto.
getDescricao());
    holder.mTextViewPreco.setText(String.
format("R$ %.2f", produto.getPreco()));
}
```

4) Implemente o método getItemCount() para retornar a quantidade total de itens do objeto List.

```
@Override
public int getItemCount() {
    // Este método retorna a quantidade de itens que
deverá ser exibida
    if (mListProduto == null) return 0;
    return mListProduto.size();
}
```

Até o momento temos um objeto *List*<Produto> declarado na *Activity* e outro objeto *List*<Produto> declarado no *Adapter*. Devemos, portanto, vinculá-los.

Na classe Adapter, criaremos os seguintes métodos:

a) atualizarListagemCompleta(List<Produto> listProduto)

b) inserirItemNaListagem(int posicao, List<Produto> listProduto)

c) excluirItemDaListagem(int posicao, List<Produto> listProduto)

d) atualizarItemDaListagem(int posicao, List<Produto> listProduto)

```
public void atualizarListagemCompleta(List<Produto>
mListProduto) {
    // Atualiza a listagem do Adaptador
    this.mListProduto = mListProduto;

    // Notifica o Adaptador para atualizar a listagem
completa
    notifyDataSetChanged();
}

public void inserirItemNaListagem(int position,
List<Produto> mListProduto) {
    this.mListProduto = mListProduto;
```

```
        // Notifica o Adaptador para inserir o item de
acordo com a posição recebida no parâmetro
    notifyItemInserted(position);
}

public void excluirItemDaListagem(int position,
List<Produto> mListProduto) {
    this.mListProduto = mListProduto;

     // Notifica o Adaptador para remover o item de
acordo com a posição recebida no parâmetro
    notifyItemRemoved(position);
}

public void atualizarItemDaListagem(int position,
List<Produto> mListProduto) {
    this.mListProduto = mListProduto;

   // Notifica o Adaptador para atualizar os dados do
item de acordo com a posição recebida no parâmetro
    notifyItemChanged(position);
}
```

De acordo com o código acima, sempre que um item da listagem for modificado, o programador poderá chamar pelo método correspondente para notificar o Adaptador sobre a alteração.

Assimile

Observe no código acima que há um método específico para cada tipo de alteração na listagem. Você deve garantir que cada método seja chamado no momento em que a modificação ocorra, para que o Adaptador possa atualizar a listagem. A partir desses métodos, o Adaptador apresentará uma animação na listagem para que o usuário consiga visualizar a modificação.

Quando o método *notifyDataSetChanged*() é chamado, o Adaptador reconstrói todos os itens da listagem e a animação não é exibida, portando, apenas chame por este método como último recurso.

Finalizamos a construção da classe *Adapter*. Agora devemos configurar o *RecyclerView* na *Activity*.

1. Acesse a classe MainActivity.java.
2. Declare e instancie o objeto *RecyclerView*.
3. Defina um tamanho fixo para o *RecyclerView* ser exibido no layout.
4. Defina um Gerenciador de Layout para posicionar cada item da listagem.
5. Defina um objeto *Adapter*.
6. Insira os dados no *Adapter*.

Reflita

Segundo Developer (2018), um elemento *RecyclerView* utiliza um Gerenciador de Layout para posicionar cada item da listagem.

Por meio de um objeto *RecyclerView.LayoutManager,* é possível posicionar os elementos um após o outro, por exemplo, conforme uma listagem de contatos. Ou ainda, em formato similar a um tabuleiro, como um aplicativo que exibe várias fotos para o usuário. Para atingir estes tipos de exibições no *RecyclerView*, quais gerenciadores de layout devem ser declarados?

```
1  @Override
2  protected void onCreate(Bundle savedInstan-
   ceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_main);
5
6      mListProduto = new ArrayList<>();
7      mListProduto.add(new Produto("Smart-
   phone", 1800f));
8      mListProduto.add(new Produto("Tablet",
   3600f));
9      mListProduto.add(new Produto("TV 50",
   2100f));
10
```

```
11      // Instancia o RecyclerView
12      rvListagem = findViewById(R.id.rvListagem);
13
14      // Define um tamanho fixo no layout
15      rvListagem.setHasFixedSize(true);
16
17      // Define como os itens serão exibidos no
        RecyclerView
        RecyclerView.LayoutManager layoutManager
18      = new LinearLayoutManager(this, LinearLayout-
        Manager.VERTICAL, false);
19      rvListagem.setLayoutManager(layoutManager);
20
21      // Adiciona uma linha entre cada item da
        listagem
        RecyclerView.ItemDecoration itemDecora-
22      tion = new DividerItemDecoration(this, Di-
        viderItemDecoration.VERTICAL);
23      rvListagem.addItemDecoration(itemDecora-
        tion);
24
25      // Define o Adapter para o RecyclerView
26      ListagemAdapter listagemAdapter = new
        ListagemAdapter();
27      rvListagem.setAdapter(listagemAdapter);
28
29      // Atualiza todos os itens para o Adaptador
30      listagemAdapter.atualizarListagemComple-
        ta(mListProduto);
31  }
```

Exemplificando

O código abaixo exemplifica como exibir uma listagem de conteúdo em duas colunas. Você deve modificar apenas a implementação do Gerenciador de Layout.

```
17      // Define como os itens serão exibidos no RecyclerView
        RecyclerView.LayoutManager layoutManager =
18  new GridLayoutManager(this, 2, GridLayoutMan-
    ager.VERTICAL, false);

19  rvListagem.setLayoutManager(layoutManager);
```

Observe que na linha 18 é declarado um GridLayoutManager, que recebe quatro parâmetros:

1) O primeiro parâmetro refere-se a um objeto Context.

2) O segundo parâmetro refere-se à quantidade de colunas que será exibida na listagem.

3) O terceiro à orientação das colunas.

4) E, por último, se a listagem deve ser exibida no modo "reverso", muito utilizado em aplicativos que apresentam uma janela de conversação, dessa forma, o conteúdo da conversa é sempre inserido de baixo para cima.

Caro aluno, finalizamos esta seção com a construção de um recurso de layout utilizando *RecyclerView*. Há muitas funcionalidades disponíveis no *RecyclerView* que merecem ser exploradas. Dedique um tempo para este elemento e prepare-se para a próxima seção. Até lá!

Sem medo de errar

Caro programador, você foi contratado para desenvolver um aplicativo Android para uma rede de academias que atua em vários municípios do Brasil. Os alunos dessa rede de academias podem frequentar qualquer unidade como cortesia da assinatura do plano. Para manter organização e controle do seu público, a rede deseja manter um cadastro de todos os seus alunos por meio de um aplicativo Android.

O desenvolvimento do aplicativo está dividido em três etapas. Nesse momento, você deverá implementar a listagem para exibir todos os alunos cadastrados, bem como as telas do aplicativo.

A primeira tela será um Fragment. Para implementá-lo, adicione o elemento RecyclerView. Este elemento o permitirá exibir todos os alunos cadastrados e atualizar os itens da listagem sempre que novos alunos forem cadastrados. Devem ser exibidas as seguintes informações para cada aluno exibido no RecyclerView: Nome, Idade, Altura, Peso e Unidade de Cadastro.

A segunda tela também será um Fragment que permitirá que novos alunos sejam cadastrados.

Portanto, siga as dicas abaixo para desenvolver a tarefa que foi delegada a você.

1) Crie um projeto no Android Studio com uma *Empty Activity*.

2) Crie dois *Blank Fragments*. O primeiro deverá chamar ListagemFragment e o segundo CadastroFragment. Como o nome sugere, o primeiro fragmento é responsável por exibir a listagem com os alunos cadastrados e o segundo fragmento é responsável por cadastrar alunos.

3) Acesse o recurso de layout fragment_listagem.xml e insira um elemento *RecyclerView*. Modifique os seus atributos para que preencha toda a área do fragmento.

4) Acesse o recurso de layout fragment_cadastro.xml e insira cinco elementos *EditText*. Cada *EditText* representa o Nome, Idade, Altura, Peso e Unidade de Cadastro de cada aluno. Não se esqueça de definir um ID para cada EditText.

5) Também será necessário incluir um elemento Button para que os dados de cada EditText possam ser inseridos no Banco de Dados. Não se preocupe com essa funcionalidade, pois a implementaremos em outra etapa de desenvolvimento.

6) Acesse o recurso de layout activity_main.xml e insira dois elementos Button e um elemento FrameLayout.

6.1) O elemento *FrameLayout* é responsável por receber os *Fragments* criados.

6.2) Os elementos Button são responsáveis por alternar entre o ListagemFragment e o CadastroFragment.

7) Acesse a classe MainActivity.java e adicione um *Listener* para cada botão para que seja possível alternar entre os Fragments.

Agora devemos preparar o nosso elemento *RecyclerView*.

8) Crie um recurso de layout com o nome item_aluno.xml.

8.1) Certifique-se de que o elemento-pai tenha o atributo *android:layout_height* definido para *wrap_content*.

8.2) Insira cinco elementos *TextView*. Cada *TextView* representa o Nome, Idade, Altura, Peso e Unidade de Cadastro do aluno. Não se esqueça de definir um ID para cada *TextView*.

9) Crie uma classe java chamada AcademiaAdaptador que estenda um objeto RecyclerView.Adapter<AcademiaAdaptador. AcademiaViewHolder>.

10) Na classe AcademiaAdaptador, sobreponha os métodos: *onCreateViewHolder*(), *onBindViewHolder*() e *getItemCount*().

11) Implemente cada método conforme recomendado.

12) Crie uma inner class estática chamada AcademiaViewHolder que estenda um objeto *ViewHolder*.

13) Na inner class AcademiaViewHolder, declare e instancie todos os *TextView* criados no recurso de layout item_aluno.xml.

14) Na classe AcademiaAdaptador, declare os métodos responsáveis por notificar o Adapter das possíveis modificações nos itens da listagem.

Para finalizar a estrutura da primeira etapa de desenvolvimento, acesse a classe ListagemFragment.java.

15) Declare um objeto List<Aluno> para recuperar do banco de dados todos os alunos cadastrados. Esta funcionalidade será implementada em outra etapa deste projeto.

16) Declare o objeto *RecyclerView* e o objeto AcademiaAdaptador.

17) Em seguida, sobreponha o método *onActivityCreated*().

18) No método *onActivityCreated*(), instancie e configure o objeto *RecyclerView* conforme dicas abaixo:

18.1) Defina um tamanho fixo para o *RecyclerView* ser exibido no layout.

18.2) Defina um Gerenciador de Layout. Para que cada item da listagem de conteúdo seja exibido um após o outro, utilize um objeto *LinearLayoutManager* com orientação definida para vertical.

18.3) Se desejar, adicione uma linha decorativa para separar um item do outro através do objeto *RecyclerView.ItemDecoration*.

18.4) Instancie o objeto AcademiaAdaptador declarado e o vincule ao *RecyclerView*.

18.5) Chame pelo método do Adapter responsável por atualizar todos os itens da listagem, conforme definido no passo 14.

Chegamos ao final da primeira etapa de desenvolvimento do aplicativo para a rede de academias.

Você poderá verificar uma opção de código-fonte para o problema proposto por meio do link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u4/s1/sem-medo-de-errar.pdf> ou QR Code.

Neste momento, o nosso projeto apresenta a estrutura e o layout necessários para darmos continuidade no aplicativo. Bom trabalho!

Avançando na prática

Listagem de Fornecedores.

Descrição da situação-problema

Você foi contratado para desenvolver uma aplicativo Android para uma distribuidora de alimentos que deseja manter o controle de todos os fornecedores com quem trabalham. É importante que o aplicativo apresente todos os fornecedores cadastrados em uma listagem. Para cada fornecedor da listagem, devem ser exibidas as

seguintes informações: CNPJ, Razão Social, localização, telefone para contato. Crie um projeto no Android Studio com uma Empty Activity. O layout desta Activity deve receber o elemento RecyclerView para exibir a listagem com todos os fornecedores. Também será necessário criar um recurso de layout chamado item_fornecedor.xml, que será responsável por definir como as informações de cada fornecedor estarão dispostas no RecyclerView. Em seguida, crie as classes Adapter e ViewHolder para que o RecyclerView possa exibir os fornecedores cadastrados. Para finalizar, instancie o RecyclerView e o configure no método onCreate() da classe MainActivity.java.

Resolução da situação-problema

Para desenvolver o aplicativo Android para a distribuidora de alimentos, certifique-se de completar todas as dicas abaixo:

1) Crie um projeto no Android Studio com uma *Empty Activity*.

2) Crie uma classe java chamada Fornecedor e declare os atributos CNPJ, Razão Social, localização, telefone para contato que serão apresentados para cada fornecedor na listagem.

3) Insira o elemento *RecyclerView* na seção *dependencies* do arquivo build.gradle (Module: app).

4) Acesse o arquivo activity_main.xml e insira o elemento *RecyclerView*. Modifique os seus atributos para que ocupe todo o layout.

5) Crie um recurso de layout chamado item_fornecedor.xml e insira quatro elementos *TextView* para exibir as informações de cada fornecedor. O elemento-pai do recurso de *layout* deve ter o atributo *android:layout_height* definido para *wrap_content*.

6) Crie uma classe java chamada ListagemAdaptador que estenda um objeto RecyclerView.Adapter<ListagemAdaptador.ListagemViewHolder>. Sobreponha e implemente os métodos *onCreateViewHolder*(), *onBindViewHolder*() e *onItemCount*().

7) Declare também um método para notificar o Adaptador que todos os itens foram atualizados. Nesse método, você deve receber como parâmetro um objeto List<Fornecedor> e deve chamar por notifyDataSetChanged().

8) Crie uma inner class Java estática chamada ListagemViewHolder que estenda um objeto RecyclerView.ViewHolder. Crie um construtor

que receba como parâmetro um objeto *View*. Implemente o construtor de modo que chame pelo comportamento da classe pai e instancie todos os elementos do layout de cada item.

9) Acesse a classe MainActivity.java e implemente o método *onCreate*() para configurar o objeto *RecyclerView*.

Para finalizar, declare um objeto List<Fornecedor> na classe MainActivity.java e insira alguns objetos Fornecedor para testar o RecyclerView. Bom trabalho!

Faça valer a pena

1. RecyclerView é um elemento View utilizado em aplicativos Android para exibir uma listagem de conteúdo para o usuário. Segundo Developer (2018), a construção de um RecyclerView exige que alguns componentes trabalhem juntos para exibir os dados.

Qual alternativa apresenta as classes que devem ser criadas para exibir dados em um RecyclerView?

a) Adapter e ViewHolder.
b) onCreateViewHolder() e onBindViewHolder().
c) ViewHolder e onCreateViewHolder().
d) ViewHolder e onBindViewHolder().
e) Adapter e getCountItem().

2. De acordo com Developer (2018), RecyclerView é um elemento avançado e flexível para exibir uma listagem de dados para o usuário. O RecyclerView exige a criação de um Adapter para incluir, excluir e atualizar cada item da listagem com os dados apropriados.

Qual alternativa apresenta os métodos responsável por notificar o Adapter quando um item é incluído, modificado ou excluído?

a) notifyDataSetChanged().
b) notifyItemChanged().
c) notifyItemInserted(), notifyItemChanged(), notifyItemRemoved().
d) notifyItemInserted(), notifyItemRemoved(), notifyItemChanged().
e) notifyItemInserted(),notifyDataSetChanged() e notifyItemRemoved().

3. Ao declarar um objeto RecyclerView na Activity ou no Fragment, é necessário vincular um Gerenciador de Layout. A responsabilidade do Gerenciador de Layout é definir como cada item da listagem de conteúdo estará posicionado. É possível exibir cada item um após o outro conforme uma listagem de feed de notícias. Ou ainda, em formato similar a um tabuleiro, como um aplicativo que exibe várias fotos para o usuário.

Selecione a alternativa que apresenta o Gerenciador de Layout responsável por dispor os itens um após o outro.

a) LinearLayout.
b) LinearLayoutManager.
c) LayoutManager.
d) GridLayout.
e) GridLayoutManager.

Seção 4.2

Introdução ao banco de dados na nuvem

Diálogo aberto

Caro aluno, seja bem-vindo a mais uma seção! Você já precisou acessar as informações daquele seu aplicativo Android favorito em diferentes dispositivos móveis? Os aplicativos Android são capazes de manter os dados do usuário sincronizados em todos os seus dispositivos móveis. Dessa forma, é possível compartilhar as suas informações entre diferentes dispositivos móveis ao utilizar o mesmo aplicativo. Nesta seção, criaremos um banco de dados na nuvem a partir do *Realtime Database* para podermos sincronizar as informações do usuário em qualquer dispositivo móvel.

Na primeira etapa de desenvolvimento do projeto para a rede de academias, desenvolvemos o *layout* do aplicativo Android. Dando continuidade, é hora de definirmos qual banco de dados utilizar para que todas as informações cadastradas estejam sincronizadas. É sugerido que seja utilizado um banco de dados na nuvem. Dessa forma, todas as informações armazenadas estarão atualizadas em tempo real.

Na etapa de desenvolvimento, o gerente do projeto solicita aos programadores duas tarefas. Na primeira tarefa, você deverá realizar a conexão do projeto com o banco de dados. E, na segunda tarefa, você terá que implementar a classe Java que representará o objeto Aluno com as informações que serão armazenadas no banco de dados. Conforme ficou definido no diálogo que você participou com o representante da rede de academias, deverão ser salvos no banco de dados os atributos nome, idade, altura e peso do aluno e qual rede de academia o cadastro está sendo realizado. Portanto, finalize a segunda tarefa que foi delegada a você implementando a classe Java de modo que cada Aluno seja identificado como único no aplicativo Android.

Para finalizar a segunda etapa de desenvolvimento do projeto, conheceremos a plataforma *Firebase* para Android e o produto *Realtime Database*. Por meio destes recursos, você poderá desenvolver aplicativos de alta qualidade! Bons estudos!

Não pode faltar

Firebase para Android é a plataforma da empresa Google que lhe oferece produtos para desenvolver aplicativos com alta-performance. De acordo com Firebase (2018), a plataforma disponibiliza diversos produtos que podem ser combinados para oferecerem mais funcionalidades para os usuários. Listamos alguns dos produtos oferecidos pelo Firebase:

- **Realtime Database:** "banco de dados hospedado na nuvem. Os dados são armazenados como JSON e sincronizados em tempo real com todos os clientes conectados" (FIREBASE, 2018, p. 1).
- **Authentication:** serviço de autenticação de usuários. "Ele oferece suporte à autenticação por meio de senhas, números de telefone e provedores de identidade federados como Google, Facebook, Twitter e muito mais" (FIREBASE, 2018, p. 1).
- **Cloud Storage** é um poderoso serviço de armazenamento de objetos. Os usuários podem armazenar documentos como vídeos ou fotos e acessá-los a qualquer momento.

Para que possamos utilizar os produtos disponíveis na plataforma em nossos aplicativos, devemos completar duas etapas:

1) Configurar no projeto do Android Studio o Firebase e os produtos que desejamos utilizar.
2) Ativar na plataforma do Firebase os produtos que serão utilizados, disponível em: <https://console.firebase.google.com/>. Acesso em: 27 ago. 2018.

Assimile

O que é a nuvem? Nuvem "é um termo utilizado para descrever uma rede global de servidores" (AZURE, 2018, [s.d.]). Ainda segundo o mesmo autor,

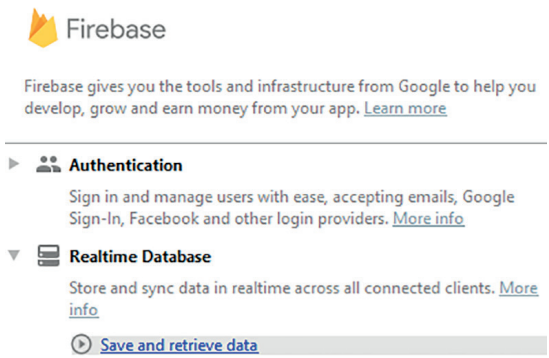
estes servidores são responsáveis por armazenar e gerenciar dados, executar aplicativos ou fornecer conteúdos ou serviços, como transmissão de vídeos, webmail, software de produtividade ou mídias sociais. Em vez de acessar arquivos e dados do local ou de um PC, você pode acessá-los online, de qualquer dispositivo com acesso à Internet. As informações estarão disponíveis em qualquer lugar, a qualquer hora. (AZURE, 2018, [s.d.])

Para completarmos a primeira etapa, abra um projeto no Android Studio e siga os passos abaixo:

1) Acesse o menu ***Tools*** ("Ferramentas"), em seguida ***Firebase***.

2) A Figura 4.2 ilustra a janela ***Assistent*** responsável por auxiliá-lo a incluir o Firebase. Nessa janela, são listados os produtos disponíveis na plataforma. Localize o produto *Realtime Database* e selecione a opção ***Save and retrieve data*** ("Salvar e recuperar dados").

Figura 4.2 | Janela Assistent Firebase



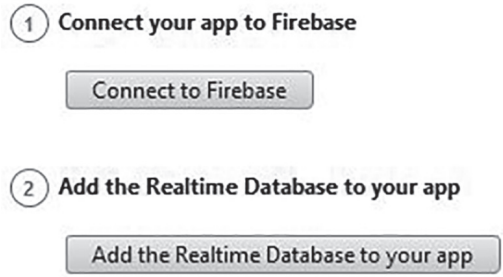
Fonte: captura de tela do Android Studio.

3) Será carregado um passo-a-passo conforme exibido na figura 4.3. Os dois primeiros passos devem ser seguidos:

3.1) *Connect your app to Firebase* ("Conecte seu aplicativo ao Firebase")

3.2) *Add the Realtime Database to your app* ("Adicione o Realtime Database ao seu aplicativo")

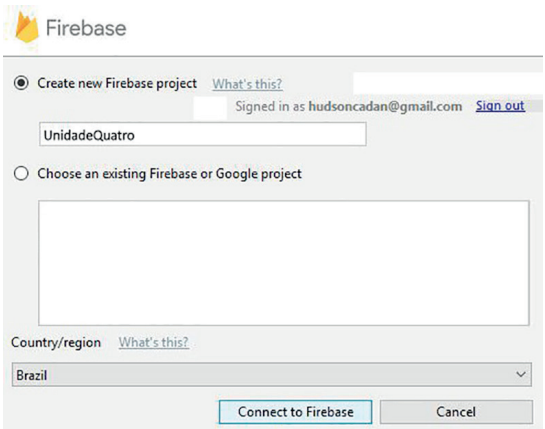
Figura 4.3 | Passo-a-passo para configurar o Realtime Databse



Fonte: captura de tela do Android Studio.

Selecione a primeira opção ***Connect to Firebase.*** Nessa opção, será necessário realizar *login* em uma conta Google. Portanto, caso você não esteja conectado a uma conta, o navegador de internet será aberto automaticamente para que você possa realizar o *login*. Após finalizar o *login,* será aberta no Android Studio a janela para se conectar ao *Firebase*, conforme Figura 4.4.

Figura 4.4 | Janela *Connect to Firebase*



Fonte: captura de tela do Android Studio.

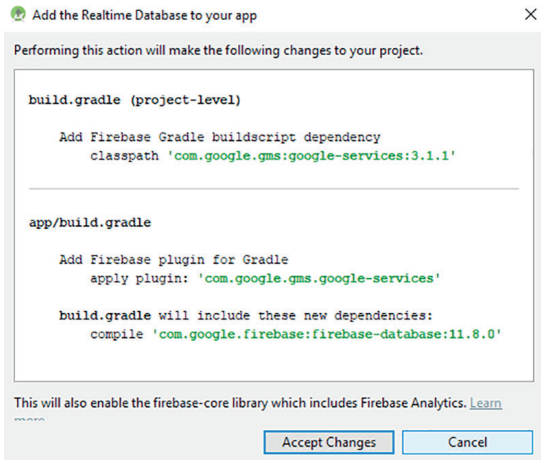
A primeira opção ***Create new Firebase project*** o permite criar um projeto na plataforma Firebase. A segunda opção, ***Choose an existing Firebase or Google Project,*** é utilizada para selecionar um projeto Firebase existente.

Matenha selecionado *Create new Firebase project* e defina o nome para um novo projeto no *Firebase*. Em seguida, clique no botão ***Connect to Firebase.*** Aguarde até que o Android Studio configure o *Firebase* em seu projeto.

Para concluir o segundo passo, conforme exibido na Figura 4.3, selecione a opção ***Add the Realtime Database to your app.***

Em alguns casos, será necessário modificar algumas configurações no arquivo build.gradle. O Android Studio o notificará, conforme exibido na Figura 4.5. Portanto, aceite as modificações necessárias clicando no botão ***Accept Changes.***

Figura 4.5 | Janela de atualização de configuração do build.gradle



Fonte: captura de tela do Android Studio.

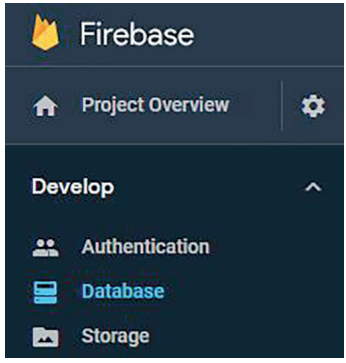
Aguarde até que o Android Studio configure o *Realtime Database* no seu projeto do Android Studio.

Agora é hora de completarmos a segunda etapa definida no início desta seção.

1) Acesse o console do Firebase, disponível em: <https://console.firebase.google.com/>. Acesso em: 27 ago. 2018.

2) Você verá listado o projeto Firebase criado. Abra-o. Na lateral esquerda, selecione ***Develop,*** em seguida ***Database,*** conforme Figura 4.6.

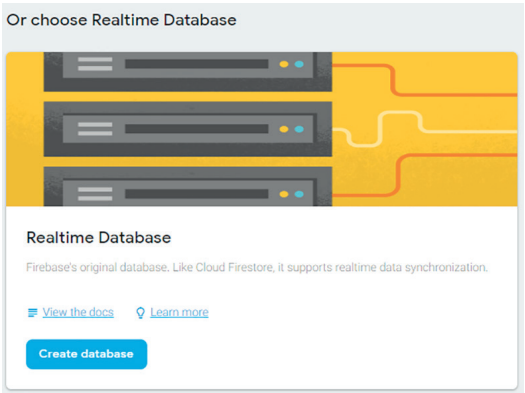
Figura 4.6 | Menu lateral do Console Firebase



Fonte: captura de tela do Firebase.

3) Deslize a página e localize a opção para criar o Realtime Database, conforme exibido na Figura 4.7. Clique no botão ***Create database.***

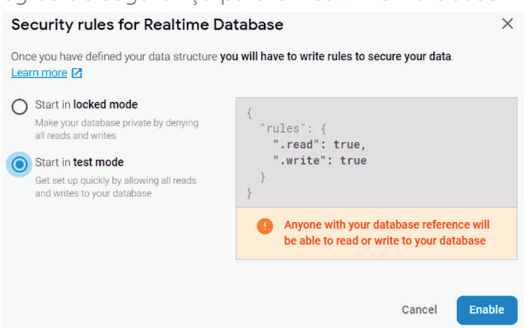
Figura 4.7 | Página para criar o Realtime Database



Fonte: captura de tela do Firebase.

4) Será solicitado que você defina as regras de segurança. Certifique-se de selecionar ***Start in test mode*** ("Iniciar em modo de teste"), conforme exibido na Figura 4.8. Firebase (2018) nos ensina que o Realtime Database oferece regras de segurança que definem quais usuários podem ler e escrever no banco de dados. Por padrão, apenas usuários que utilizam o produto Authentication possuem permissão. Contudo, você pode configurar para iniciar o Realtime Database no modo de teste para que quaisquer usuários tenham acesso de leitura e escrita. Certifique-se de restringir novamente as regras de segurança no banco de dados quando você configurar o Authentication.

Figura 4.8 | Regras de segurança para o Realtime Database



Fonte: captura de tela do Firebase.

Finalizamos as etapas necessárias para configurarmos o Firebase e o Realtime Database no Android Studio. Nesse momento, devemos definir como e quais informações serão armazenadas no banco de dados na nuvem.

De acordo com Firebase (2018, p. 1) "o Realtime Database é um banco de dados NoSQL e, por isso, tem otimizações e funcionalidades diferentes de um banco de dados relacional".

Assimile

Afinal, o que é banco de dados NoSQL? Atualmente, aplicativos móveis geram um grande volume de dados. De acordo com Spring (2018), o termo "NoSQL" refere-se ao fato de que os bancos de dados relacionais não são adequados para todas as soluções, particularmente aquelas que envolvem grandes volumes de dados. Amazon (2018) indica os bancos de dados NoSQL para muitos aplicativos de big data, mobilidade e web.

Segundo MongoDB (2018), os bancos de dados NoSQL utilizam modelos de armazenamentos de dados como pares chave-valor, grafos, documentos ou coluna larga.

Firebase (2018) garante que planejar como os dados serão salvos e recuperados tornam o processo de desenvolvimento o mais fácil possível. No início desta seção, aprendemos que o Realtime Database armazena os dados como JSON. Portanto, no Realtime Database não existem tabelas e registros. A sua estrutura é composta de *key-value* (chave-valor).

Exemplificando

Para exemplificar a estrutura do Realtime Database, vamos analisar as seguintes informações armazenadas:

raizDoProjeto:

+ produtos:

+ -LFUXkJLOgdPKYw7JINs

descricao: "Smartphone"
preco: 1800
+ -LFUXkb5xcNyAISw3k2d
descricao: "Tablet"
preco: 2200

O Realtime Database possui uma raiz com o nome do projeto. Todos os dados adicionados ao banco de dados estão associados a esta raiz.

Observe que há um node "produtos". Em seguida há um valor que representa uma chave única para cada produto. Finalmente são armazenados as descrições e preços de cada produto.

Para adicionar dados no Realtime Database, você pode enviar ao banco de dados uma das seguintes opções:

1) Objeto Java personalizado que atenda às seguintes características:

a) Possua um construtor que não receba parâmetros;

b) Possua Getters públicos para cada atributo.

2) Dados compatíveis aos tipos JSON disponíveis:

a) String

b) Long

c) Double

d) Boolean

e) Map<String, Object>

f) List<Object>

Reflita

De acordo com Spring (2018), programadores que utilizam soluções NoSQL não necessariamente dispensam os bancos de dados relacionais. Deve-se identificar como se dá o uso correto de cada um dos bancos de dados para utilizar na aplicação que está sendo

desenvolvida. Segundo o mesmo autor, o termo "NoSQL" siginifica "Not Only SQL" ("Não Apenas SQL").

Em um aplicativo Android, você poderá utilizar banco de dados relacional por meio da biblioteca Room para armazenamento local e acrescentar o Realtime Database para armazenamento na nuvem?

Ao enviar um objeto Java para o Realtime Database, os atributos dele serão convertidos para a estrutura JSON automaticamente. Também é possível recuperar um único objeto ou uma coleção de objetos do Realtime Database.

Pesquise mais

De acordo com Oracle (2018), uma coleção representa um grupo de objetos. Algumas coleções permitem objetos duplicados e outras não.

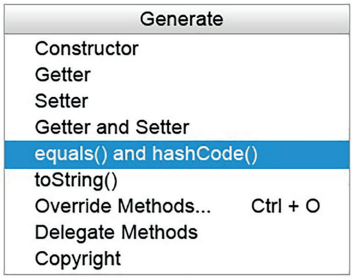
Ao trabalhar com um objeto Java personalizado, por exemplo, um objeto Produto, você deve sobrepor os métodos equals() e hashCode() para garantir que as coleções identifiquem quais objetos são iguais. Acompanhe o conteúdo disponível a partir do segundo minuto no vídeo: <https://www.youtube.com/watch?v=T008B4vURk4&t=1031s>. Acesso em: 2 jul. 2018.

Quando recuperarmos uma coleção de objetos do Realtime Database, devemos garantir que sejam identificados quais objetos são iguais para que o aplicativo não apresente comportamentos inesperados pelo usuário. Portanto, sobreponha os métodos *equals*() e *hashCode*() da classe Java.

Dica

Para sobrepor os métodos equals() e hashcode() em uma classe Java utilizando o Android Studio, utilize as teclas de atalho ALT + INSERT, conforme Figura 4.9.

Figura 4.9 | Atalho para sobrepor os métodos equals() e hashCode()



Fonte: captura de tela do Android Studio.

```
1  package com.exemplo.unidadeiv;
2
3  import java.util.Objects;
4
5  class Produto {
6
7    private String id;
8    private String descricao;
9    private float preco;
10
11   public Produto() {
12   }
13
14   public Produto(String descricao, float preco) {
15     this.descricao = descricao;
16     this.preco = preco;
17   }
18
19   public String getId() {
20     return id;
21   }
22
23   public void setId(String id) {
24     this.id = id;
```

```
25    }
26
27    public String getDescricao() {
28      return descricao;
29    }
30
31    public void setDescricao(String descricao) {
32      this.descricao = descricao;
33    }
34
35    public float getPreco() {
36      return preco;
37    }
38
39    public void setPreco(float preco) {
40      this.preco = preco;
41    }
42
43    @Override
44    public boolean equals(Object o) {
45      if (this == o) return true;
        if (o == null || getClass() != o.get-
46  Class()) return false;
47      Produto produto = (Produto) o;
48
        // Se os códigos dos produtos forem
49  iguais, retorna true
        return Objects.equals(getId(), produto.
50  getId());
51    }
52
53    @Override
54    public int hashCode() {
55      return Objects.hash(getId());
56    }
57  }
```

Pesquise mais

Conheça mais detalhes sobre como estruturar dados utilizando o Realtime Database na documentação disponível em: <https://firebase.google.com/docs/database/android/structure-data>. Acesso em: 2 jul. 2018.

Chegamos ao final desta seção. Agora você, programador, poderá incluir o Realtime Database a um projeto do Android Studio. Na próxima seção, estudaremos como enviar e recuperar dados do banco de dados na nuvem. Continue praticando e o vejo na próxima seção!

Sem medo de errar

Você está atuando em uma equipe de programadores responsável por desenvolver um aplicativo Android para uma rede de academias. Na segunda etapa de desenvolvimento, foram designadas a você duas tarefas. Na primeira tarefa, você terá que conectar a plataforma Firebase e incluir o Realtime Database ao projeto no Android Studio. Na segunda tarefa, você terá que criar a classe Java que representará o objeto com as informações que serão armazenadas no banco de dados.

Siga os passos a seguir para concluir a primeira tarefa que foi delegada a você.

1) Acesse o projeto no Android Studio criado na primeira etapa de desenvolvimento.

2) Acesse o menu ***Tool***, em seguida ***Firebase***.

3) A janela ***Assistent*** será aberta. Localize o produto *Realtime Database* e selecione ***Save and retrieve data.***

4) Siga os dois passos apresentados na janela *Assistent*.

4.1) Para o primeiro passo, conecte-se ao Firebase utilizando uma conta Google.

4.2) Para o segundo passo, adicione o Realtime Database ao aplicativo.

5) Acesse o Console do Firebase, disponível em: <https://console.firebase.google.com/>.

6) No menu exibido na lateral esquerda, acesse a opção ***Develop***, em seguida ***Database.***

7) Localize na página aberta a opção ***Realtime Database*** e clique no botão ***Create Database*.**

8) A janela ***Security rules for Realtime Database*** será aberta. Certifique-se de manter selecionada a opção ***Start in test mode.***

Ao concluir a primeira tarefa, o Realtime Database estará incluído no projeto do Android Studio e o Firebase estará configurado para trabalhar com o Realtime Database. Para concluir a segunda tarefa, siga os seguintes passos:

1) Crie uma classe Java chamada Aluno.

2) Declare um construtor público que não receba parâmetros. Este construtor é exigido pelo *Realtime Database*.

3) Declare os atributos privados que representarão as informações de cada aluno.

3.1) Declare um atributo privado do tipo String para representar o id do aluno. O atributo será gerado automaticamente pelo banco de dados e é responsável por identificar quais objetos Aluno são iguais.

3.2) Declare os demais atributos nome, idade, altura, peso e rede de cadastro.

4) Crie os *Getter and Setter* para cada atributo. Utilize as teclas de atalho ALT + INSERT.

5) Sobreponha os métodos equals() e hashCode(). Utilize a tecla de atalho ALT + INSERT. Ao sobrepor estes métodos, é importante verificar se apenas o atributo *id* está incluído. Este é o atributo responsável por identificar se os objetos são iguais ou não em uma *Collection*.

```
@Override
public boolean equals(Object o) {
  if (this == o) return true;
  if (o == null || getClass() != o.getClass())
return false;
  Aluno aluno = (Aluno) o;

  // Se o objeto aluno possui o mesmo id, re-
torna true
```

```
    return Objects.equals(getId(), aluno.get-
Id());
  }

  @Override
  public int hashCode() {
    return Objects.hash(getId());
```

Finalizamos a segunda etapa de desenvolvimento para a rede de academias. Você fez um bom trabalho nesta etapa, agora o projeto do Android Studio está conectado ao Realtime Database e temos a classe Aluno que será enviada ao banco de dados na nuvem. Aguardo-o na última etapa de desenvolvimento, até logo!

Avançando na prática

Concessionária de Veículos.

Descrição da situação-problema

Uma empresa especializada em desenvolvimento para dispositivos móveis o contratou para atuar em uma equipe de programadores. A equipe está responsável por desenvolver um aplicativo Android para uma concessionária de veículos. O objetivo do aplicativo é utilizar a nuvem para compartilhar entre todos os usuários conectados ao aplicativo informações sobre os veículos que estão à venda. A sua tarefa dentro da equipe é criar um projeto no Android Studio capaz de armazenar dados no Realtime Database. Você também deverá criar a classe Java que representará os objetos que serão armazenados na nuvem.

Resolução da situação-problema

Para finalizar as tarefas que foram delegadas a você, crie um projeto no Android Studio e siga os passos a seguir:

1) Acesse o menu Tools, em seguida Firebase.

2) Através da janela *Assistent, c*onecte-se ao Firebase e adicione o Realtime Database no projeto.

3) Acesse o Firebase Console disponível em: <https://console.firebase.google.com/>. Acesso em: 28 ago. 2018.

4) Acesse o menu ***Database*** e localize a opção ***Create Database.***

5) A janela ***Security rules for Realtime Database*** será aberta. Certifique-se de manter selecionada a opção ***Start in test mode.***

6) Retorne ao Android Studio e crie uma classe Java chamada Veiculos que atenda às seguintes características:

6.1) Possua um construtor que não receba parâmetros.

6.2) Possua atributos privados e Getter and Setter públicos para cada atributo.

6.3) Sobreponha os métodos equals() e hashCode().

Ao finalizar suas tarefas, apresente o trabalho para o restante da equipe e aguarde por novas instruções. Bom trabalho!

Faça valer a pena

1. A plataforma Firebase para Android oferece produtos que auxiliam o desenvolvimento de aplicativos Android. É possível, por exemplo, utilizar a nuvem para armazenar e sincronizar dados de aplicativos, autenticar usuários de forma segura, enviar mensagens e notificações para um grupo de usuários e ainda corrigir problemas através de relatórios avançados de erros.

Qual é o produto disponível no Firebase responsável por armazenar conteúdo NoSQL na nuvem?

a) Analytics.
b) Authentication.
c) Storage.
d) Realtime Database.
e) Crashlytics.

2. O Realtime Database é um banco de dados NoSQL que armazena dados na nuvem no formato JSON. Firebase (2018) garante que todos os dados são sincronizados em tempo real para todos os clientes que estão conectados ao banco de dados. Caso algum usuário fique sem acesso à internet, os dados permanecerão disponíveis no dispositivo móvel.

Qual alternativa apresenta as etapas necessárias para utilizar o Realtime Database em um aplicativo Android?

a) O programador deve apenas incluir o Realtime Database no projeto do Android Studio.
b) O programador deve apenas incluir o Firebase no projeto do Android Studio.
c) O programador deve configurar apenas o Firebase no projeto do Android Studio e ativar o Realtime Database no console do Firebase.
d) O programador deve configurar o Firebase e o Realtime Database no projeto do Android Studio e ativar o Realtime Database no console do Firebase.
e) O programador deve incluir o Realtime Database no projeto do Android Studio e ativá-lo no console do Firebase.

3. Firebase (2018) garante que planejar como os dados serão salvos e recuperados tornam o processo de desenvolvimento o mais fácil possível. Você pode recuperar um ou vários objetos do Realtime Database e adicioná-los em uma coleção Java. De acordo com Oracle (2018), uma coleção representa um grupo de objetos. Algumas coleções permitem objetos duplicados e outras não.

Assinale a alternativa que apresenta os métodos que devem ser sobrepostos em uma classe Java para que uma coleção identifique dois objetos iguais.

a) onCreate() e equals().
b) onCreate() e hashCode().
c) onResume() e onStart().
d) onResume() e equals().
e) equals() e hashCode().

Seção 4.3

Trabalhando com banco de dados na nuvem

Diálogo aberto

Caro aluno, chegamos ao final desta unidade! Ao utilizar um aplicativo para troca de mensagens instantâneas instalado no seu dispositivo móvel, você é capaz de enviar e de receber mensagens em tempo real. Você deve abrir a janela de conversação com algum contato disponível no aplicativo e, a partir da tela, as mensagens poderão ser enviadas e recebidas.

Nesta seção, serão apresentados conteúdos para que você possa gravar e ler informações de um banco de dados na nuvem. Dessa forma, todos os dados serão enviados e recebidos automaticamente em tempo real, de maneira similar à janela de conversação do aplicativo de mensagens instantâneas.

Na primeira etapa de desenvolvimento do aplicativo Android para a rede de academias, foram implementados o layout para exibir todos os alunos cadastrados por meio do elemento RecyclerView, bem como o layout para cadastrar alunos. Na segunda etapa, conectamos o projeto do Android Studio à plataforma Firebase e ao banco de dados Realtime Database. Agora é hora de combinarmos todas as etapas já implementadas.

Nesta etapa, programador, você deverá acessar o banco de dados na nuvem para cadastrarmos os alunos. Também a partir do banco de dados na nuvem, você deverá ler todos os alunos cadastrados e exibir na listagem implementada na primeira etapa deste projeto.

Para finalizarmos o aplicativo Android para a rede de academias, conheceremos os objetos FirebaseDatabase e DatabaseReference. Com esses objetos conseguiremos gravar e ler dados do bando de dados Realtime Database. Dedique um tempo especial para explorar as funcionalidades disponíveis por estes objetos, dessa forma, você conseguirá manipular dados armazenados na nuvem. Bons estudos!

Não pode faltar

Conforme estudamos na seção anterior, o Realtime Database é um banco de dados NOSQL que armazena dados na nuvem. Os dados armazenados na nuvem são sincronizados em tempo real com todos os usuários conectados ao banco de dados.

Também destacamos na seção anterior que o Realtime Database armazena os dados em uma estrutura JSON através de chave-valor (*key-value)*. Vamos analisar a estrutura a seguir para compreendemos como os dados estão armazenados.

```
raizDoProjeto:
    + produtos:
        +  -LFUXkJLOgdPKYw7JINs
            descricao:    "Smartphone"
            preco:          1800
        +  -LFUXkb5xcNyAISw3k2d
            descricao:    "Tablet"
            preco:          2200
```

A estrutura do Realtime Database possui uma raiz inicial de referência. A partir desta raiz, podemos acessar os demais dados armazenados no banco de dados. Cada informação armazenada é representada por uma referência. Por exemplo, se acessarmos a raiz do projeto e em seguida a referência "produtos", teremos aceso a todos os produtos cadastrados na estrutura acima.

Como acessar o banco de dados Realtime Database?

Para que possamos gravar ou ler dados no Realtime Database, devemos recuperar uma instância do banco de dados.

FirebaseDatabase é o objeto que nos fornece o ponto de entrada para acessarmos o Realtime Database. Portanto, declare um objeto FirebaseDatabase e chame pelo método getInstance().

```
FirebaseDatabase database = FirebaseDatabase.
getInstance();
```

Através da instância do FirebaseDatabase, podemos acessar uma instância de DatabaseReference. De acordo com Firebase (2018), **DatabaseReference** é uma localização específica no banco de dados que pode ser usada para ler ou gravar dados.

```
FirebaseDatabase database=FirebaseDatabase.getInstance();
DatabaseReference reference = database.getReference();
```

Nesse momento, a instância de DatabaseReference nos retorna a raiz do projeto da estrutura do Realtime Database.

Como inserir dados no Realtime Database?

A partir da instância do DatabaseReference, podemos inserir dados no Realtime Database. Vamos conhecer três métodos essenciais para implementarmos essa funcionalidade.

O método **child()** é responsável por criar ou acessar uma referência específica dentro da raiz do projeto do Realtime Dabatase.

O método **push()** também é responsável por criar uma referência específica dentro da raiz do projeto do Realtime Database, contudo, esse método gera uma chave exclusiva para o objeto que será armazenado.

O método **setValue()** armazena o objeto na referência especificada. O objeto enviado para o Realtime Database é convertido automaticamente para a estrutura do JSON.

Observe o código abaixo responsável por armazenar um objeto Produto no Realtime Database.

```
// Declaramos um objeto produto para ser armazena-
do no banco de dados
Produto produto = new Produto("Smartphone",
1800f);

// Recuperamos uma instância do Realtime Database
FirebaseDatabase database = FirebaseDatabase.ge-
tInstance();

// Recuperamos a raiz do projeto
DatabaseReference reference = database.getReferen-
ce();
```

```
// Através do método child(), acessamos a referência "produtos".
// Em seguida, através do método push(), inserimos uma referência gerada automaticamente para identificar o produto
// Finalizamos inserindo o objeto Produto no banco de dados
reference.child("produtos").push().setValue(produto);
```

A Figura 4.10 ilustra o retorno de cada método dentro do Realtime Database.

Figura 4.10 | Estrutura do Realtime Database

Fonte: elaborada pelo autor.

Reflita

A partir de um objeto DatabaseReferente é possível acessar uma referência no Realtime Database. Em alguns casos, esta referência poderá ser construída de forma mais complexa. Como você acessaria no Realtime Database uma referência de "raizDoProjeto/mercadorias/produtos/adicionais/"?

Como ler dados do Realtime Database?

A partir da instância do DatabaseReference, podemos definir uma referência para lermos dados do Realtime Database. Os métodos abaixo nos auxiliam a acessar referências específicas no Realtime Database.

O método **orderByChild()** recebe como parâmetro uma String e ordena os resultados pelo valor informado no parâmetro. O código abaixo retorna todas as referências de "produtos" ordenadas pelo atributo "descricao".

```
reference.child("produtos").
orderByChild("descricao");
```

O método **orderByKey()** ordena todos os resultados de uma referência de acordo com as suas chaves-filhas. O código abaixo retorna todas as referências de "produtos" ordenadas pelas chaves-filhas.

```
reference.child("produtos").orderByKey();
```

Caso não deseje retornar todas as referências, você poderá combinar os métodos a seguir para filtrar dados do Realtime Database. Para utilizar os métodos a seguir, você obrigatoriamente deverá combinar com algum dos métodos *orderByChild*() ou orderByKey().

O método **equalTo()** recebe como parâmetro uma String e limita os resultados da referência de acordo com o parâmetro informado. O código abaixo retorna todas as referências de "produtos", cujo atributo "descricao" seja igual a "Smartphone".

```
reference.child("produtos").
orderByChild("descricao").equalTo("Smartphone");
```

O método **limitToFirst()** define um limite máximo de itens que devem ser lidos do banco de dados. O código abaixo retorna cinco objetos de produtos ordenados pelo atributo "descricao".

```
reference.child("produtos").
orderByChild("descricao").limitToFirst(5);
```

Até o momento, criamos referências do Realtime Database. Agora vamos conhecer a técnica que nos permite acessar os objetos recuperados do banco de dados.

Segundo Firebase (2018), **ChildEventListener** é responsável por monitorar alterações em uma determinada referência do banco de dados, ou seja, é possível detectar sempre que um objeto é incluído, atualizado, excluído ou movido.

O código abaixo declara um objeto ChildEventListener. Observe que há um método de retorno de chamada para cada tipo de modificação no banco de dados, conforme comentários disponíveis no código abaixo.

```
// Declaramos o ChildEventListener

private ChildEventListener childEventListener =
new ChildEventListener() {


    @Override
    public void onChildAdded(@NonNull DataSnapshot
dataSnapshot, @Nullable String s) {
        // Método responsável por detectar sempre
que um objeto é incluído
    }

    @Override
    public void onChildChanged(@NonNull DataS-
napshot dataSnapshot, @Nullable String s) {
        // Método responsável por detectar sempre
que um objeto é atualizado
    }

    @Override
    public void onChildRemoved(@NonNull DataS-
napshot dataSnapshot) {
        // Método responsável por detectar sempre
que um objeto é excluído
    }

    @Override
    public void onChildMoved(@NonNull DataSnapshot
dataSnapshot, @Nullable String s) {
        // Método responsável por detectar sempre
que um objeto é movido
    }

    @Override
    public void onCancelled(@NonNull DatabaseError
databaseError) {
        // Método chamado quando ocorre um erro
com o banco de dados
    }
});
```

Dica

Para facilitar a implementação do ChildEventListener(), siga as dicas abaixo:

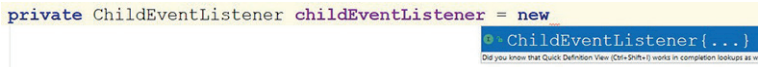
1) Declare um objeto ChildEventListener como private.

2) Digite a palavra *new* e dê um espaço.

3) Pressione CTRL + SHIT + ESPAÇO. Será aberta a janela de sugestões, conforme exibido na Figura 4.11.

4) Selecione a opção *ChildEventListener {...}* e o Android Studio escreverá todos os métodos de retorno automaticamente.

Figura 4.11 | Declaração do ChildEventListener()



Fonte: Captura de tela do Android Studio, elaborada pelo autor.

Para utilizar o ChildEventListener devemos definir uma referência do banco de dados através do objeto DatabaseReference. Em seguida, devemos chamar pelo método **addChildEventListener()** e passar como parâmetro o ChildEventListener criado. O código abaixo é capaz de detectar todas as inclusões, exclusões ou modificações que ocorrem na referência de "produtos". Observe que o ChildEventListener está sendo atribuído no método onStart().

```
@Override
protected void onStart() {
    super.onStart();

     // Adicionamos o ChildEventListener para a
referência do Realtime Database
    reference.child("produtos")

        .addChildEventListener(childEventListener);

  }
```

Cada método de retorno recebe como parâmetro um objeto **DataSnapshot**. De acordo com Firebase (2018), uma instância desse objeto contém informações de uma determinada referência do

Realtime Database. Sempre que você ler dados do banco de dados, você receberá um objeto DataSnapshot.

Para acessar os dados contidos no objeto DataSnapshot, chame pelo método **getValue()** e informe como parâmetro a classe Java que representa o objeto que se deseja recuperar. Por exemplo, para recuperar um objeto Produto, implemente conforme o código abaixo.

```
@Override
public void onChildAdded(@NonNull DataSnapshot da-
taSnapshot, @Nullable String s) {
    // Método responsável por detectar sempre que
um objeto é incluído
    Produto produto = dataSnapshot.getValue(Produ-
to.class);
}
```

A partir desse momento, você conseguirá detectar todos os produtos adicionados no Realtime Database e poderá manipular os objetos em seu aplicativo Android.

Exemplificando

Também é possível combinar todos os métodos apresentados até o momento. O código abaixo exemplifica como monitorar por inclusões, exclusões ou modificações na referência "produtos" ordenados pelo atributo "descricao".

```
@Override
protected void onStart() {
    super.onStart();
    // Adicionamos o ChildEventListener para a refe-
rência do Realtime Database
    reference.child("produtos")

        .orderByChild("descricao")

        .addChildEventListener(childEventListener);
}
```

Remover Listener

Ao adicionar um ChildEventListener, o aplicativo receberá os objetos sempre que houver modificações nas referências do Realtime Database. Contudo, o usuário poderá sair do aplicativo

e navegar para outras telas. Portanto, é importante informar ao aplicativo que pare de "ouvir" as modificações que ocorrem no banco de dados.

Essa técnica deve ser implementada para economizar dados de internet do usuário e para evitar que o aplicativo continue recebendo atualizações dos dados, principalmente quando o usuário não está com a tela do aplicativo em primeiro plano.

Os passos abaixo indicam como remover um listener de uma referência do Realtime Database.

1) Sobreponha o método onStop() correspondente ao ciclo de vida da Activity.

2) Chame pela mesma referência na qual o listener foi adicionado.

3) Chame pelo método **removeEventListener()** e passe como parâmetro o listener que será removido.

```
@Override
protected void onStop() {
    super.onStop();
    reference.child("produtos")

        .removeEventListener(childEventListener);
}
```

Como excluir dados no Realtime Database?

A partir da instância do DatabaseReference, podemos excluir dados do banco de dados. Defina qual a referência no Realtime Database que deseja remover e chame pelo método **removeValue().** O código abaixo exclui todos os produtos do banco de dados.

```
reference.child("produtos").removeValue();
```

Há casos em que desejamos remover apenas uma referência do Realtime Database. Esse tipo de abordagem exige mais esforço do programador. Por exemplo, para remover o produto "Tablet" do banco de dados, devemos seguir os seguintes passos:

1) Crie a referência que deseja remover.

2) Chame pelo método **addListenerForSingleValueEvent()** e passe como parâmetro um **ValueEventListener()**. A Interface

ValueEventListener() possui os métodos **onDataChange()** e **onCancelled()**. O método onDataChange() é responsável por receber um objeto DataSnapshot que contém todas as referências do objeto DatabaseReference, ou seja, o objeto dataSnapshot contém uma listagem com todos os produtos. Se chamarmos pelo método dataSnapshot.removeValue(), todos os produtos serão excluídos. Este não é o nosso objetivo, portanto devemos criar uma estrutura de repetição para percorrer cada referência do objeto dataSnapshot. De acordo com Firebase (2018), mesmo quando há apenas uma referência, que é o caso de "Tablet", o dataSnapshot ainda é uma lista com apenas um item que deverá ser percorrida através de uma estrutura de repetição. O código abaixo remove apenas o produto "Tablet" do banco de dados.

```
1  reference.child("produtos").orderByChild("-
   descricao").equalTo("Tablet")
2      .addListenerForSingleValueEvent(new
   ValueEventListener() {
3          @Override
4          public void onDataChange(@NonNull
   DataSnapshot dataSnapshot) {
5
6              // Percorremos todas as referências
   existentes através do método getChildren()
7              for (DataSnapshot data : dataSnap-
   shot.getChildren()) {
8
9                  // Removemos o produto "Tablet" do
   Realtime Database
10                 data.getRef().removeValue();
11             }
12         }
13         @Override
14         public void onCancelled(@NonNull Data-
   baseError databaseError) {
15         }
16     });
```

O método addListenerForSingleValueEvent() é executado apenas uma vez para recuperar os dados do banco de dados. Vamos analisar cada linha do código acima:

Linha 1: declaramos uma referência de "produtos", ordenada pelo atributo "descricao" e que seja igual a "Tablet".

Linha 2: chamamos pelo método addListenerForSingleValueEvent(), pois desejamos recuperar apenas uma vez os dados do banco de dados. Como parâmetro, declaramos um ValueEventListener().

Linha 4: implementamos o método onDataChange().

Linha 7: criamos uma estrutura de repetição através do *foreach*. No *foreach*, percorremos cada referência do DataSnaptshot chamando pelo método *getChildren()*.

Linha 10: temos a referência do produto "Tablet" que estamos procurando, portanto é seguro chamar pelo método removeValue().

Assimile

Neste conteúdo, foram apresentadas as Interfaces Listeners **ChildEventListener e ValueEventListener** para recuperar dados do Realtime Database.

Com o uso do Listener **ChildEventListener** você conseguirá implementar diferentes comportamentos para o aplicativo Android através dos métodos **onChildAdded()**, **onChildChanged**(), **onChildRemoved**(), **onChildMoved**().

ValueEventListener nos fornece apenas o método **onDataChange()** sempre que modificações ocorrerem no Realtime Database.

Como atualizar dados no Realtime Database?

A partir da instância do DatabaseReference, podemos atualizar dados do banco de dados. Defina qual a referência no Realtime Database que deseja atualizar e chame pelo método **updateChildren().** Esse método é o mais complexo para se trabalhar com o Realtime Database, pois ele recebe como parâmetro um objeto Map<String,

Object>. O objeto Map<String, Object> é responsável por criar a estrutura *key-value* com os valores atualizados para enviarmos para o banco de dados.

O código abaixo atualiza o produto "Smartphone" para "Smartphone Android" e o seu valor para "1500".

```
reference.child("produtos").orderByChild("descri-
cao").equalTo("Smartphone")
        .addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull Da-
taSnapshot dataSnapshot) {

                // Verificamos todas as referências
existentes através do método getChildren()
                for (DataSnapshot data : dataS-
napshot.getChildren()) {

                    // Objeto com a estrutura cha-
ve-valor
                    // String é a chave
                    // Object é o valor
                    Map<String, Object> valoresA-
tualizados = new HashMap<>();

                    // Informamos qual atributo
desejamos atualizar e o valor atualizado
                    valoresAtualizados.put("des-
cricao", "Smartphone Android");
                    valoresAtualizados.put("pre-
co", 1500f);

                    // Atualizamos o produto
                    data.getRef().updateChildren(-
valoresAtualizados);
                }
            }

            @Override
            public void onCancelled(@NonNull Data-
baseError databaseError) {

            }
        });
```

Pesquise mais

Conheça mais conteúdo de leitura e gravação de dados no Realtime Database por meio da documentação oficial, disponível em: <https://firebase.google.com/docs/database/android/read-and-write>. Acesso em: 12 jul. 2018.

Chegamos ao final desta seção e finalizamos a implementação de funcionalidades para manipularmos dados no Realtime Database. Seja sempre curioso e pesquise por mais funcionalidades disponíveis na plataforma Firebase.

Agora você conseguirá desenvolver aplicativos conectados à nuvem. Estou ansioso para fazer o *download* de seus aplicativos no *Google Play*.

Sem medo de errar

Esta é a última etapa do desenvolvimento do aplicativo Android para a rede de academias.

Na primeira etapa do projeto, implementamos o layout com o elemento RecyclerView para exibir todos os alunos cadastrados e implementamos o layout para cadastrar alunos.

Na segunda etapa, criamos o banco de dados Realtime Database e o conectamos ao nosso projeto. Nesta etapa, nós também criamos a classe Aluno, responsável por conter as informações de cada aluno que serão armazenadas no banco de dados.

Agora é hora de combinarmos todos as etapas já implementadas para que o aplicativo consiga gravar e ler as informações dos alunos.

1) Acesse a classe ListagemFragment.java e recupere uma instância do Realtime Database através do objeto DatabaseFirebase.

2) Em seguida, recupere um objeto DatabaseReference para ler dados do banco de dados.

3) Declare também um objeto ChildEventListener como private e o implemente conforme dicas a seguir:

3.1) Para os métodos onChildAdded(), onChildChanged() e onChildRemoved(), recupere o objeto Aluno e defina o ID do aluno recebido pelo DataSnapshot.

3.2) No método onChildAdded():

3.2.1) Insira na listagem List<Aluno> o objeto Aluno recebido.

3.2.2) Recupere a posição em que está armazenado na listagem.

3.2.3) Através do objeto academiaAdaptador, chame pelo método inserirAlunoNaListagem().

```
@Override
public void onChildAdded(DataSnapshot dataS-
napshot, String s) {
    // Recuperamos o objeto Aluno
    Aluno aluno = dataSnapshot.getValue(Aluno.
class);

    // Definimos o ID de acordo com o valor do ban-
co de dados
    aluno.setId(dataSnapshot.getKey());

    // Inserimos o aluno na listagem
    mListAluno.add(aluno);

    // Recuperamos a posição do objeto inserido
    int posicao = mListAluno.indexOf(aluno);

    // Notificamos o Adaptador para atualizar a
listagem
    academiaAdaptador.inserirAlunoNaListagem(posi-
cao, mListAluno);

}
```

3.3) No método onChildChanged():

3.3.1) Recupere a posição em que o objeto Aluno está armazenado na listagem.

3.3.2) Utilize o método set() disponível no ArrayList para substituir o objeto aluno da listagem.

3.3.3) Através do objeto academiaAdaptador, chame pelo método atualizarAlunoDaListagem().

3.4) No método onChildRemoved():

3.4.1) Recupere a posição em que o objeto Aluno está armazenado na listagem.

3.4.2) Utilize o método remove() disponível no ArrayList para remover o objeto aluno da listagem.

3.4.3) Através do objeto academiaAdaptador, chame pelo método excluirAlunoDaListagem().

4) No método onActivityCreated() instancie o elemento List<Aluno> como um ArrayList.

5) Sobreponha o método onStart(). Implemente o método de acordo com os passos a seguir:

5.1) Através objeto DatabaseReference, chame pelo método *child()* e passe como parâmetro a referência "Alunos".

5.2) Também chame pelo método orderByChild() e passe como parâmetro o atributo "nome" para que sejam recuperados todos os alunos em ordem alfabética.

5.3) Adicione o Listener declarado no passo 3 chamando pelo método addChildEventListener().

6) Sobreponha o método onStop() e remova o Listener da referência do DatabaseReferenceb.

7) Acesse a classe CadastroFragment.java e declare o objeto Button e todos os EditText inseridos no layout.

8) Sobreponha o método onActivityCreated() e instancie todos os objetos declarados no item anterior.

9) Ainda no método onActivityCreated(), adicione um Listener para detectar a interação de toque do usuário com o botão.

10) O Listener será responsável por inserir os dados digitados pelo usuário no banco de dados.

10.1) Recupere todos os valores digitados pelo usuário nos elementos EditText.

10.2) Através desses valores, crie um objeto Aluno.

10.1) Recupere uma instância do Realtime Database,

10.2) Declare um objeto DatabaseReference que atenda às seguintes características:

10.2.1) Chame pelo método *child()* e passe como parâmetro a referência "Alunos".

10.2.2) Chame pelo método push() para criar um ID automaticamente.

10.2.3) Chame pelo método setValue() e passe como parâmetro o objeto Aluno criado no passo anterior.

10.2.4) Opcionalmente, você poderá chamar pelo método addOnSuccessListener() para verificar a resposta do banco de dados caso o objeto seja inserido com sucesso.

Você poderá verificar uma opção de código-fonte para o problema proposto por meio do link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/desenvolvimento-para-dispositivos-moveis/u4/s3/codigo_1.pdf> ou QR Code.

Finalizamos a última etapa do aplicativo Android para a rede de academias. Nesta etapa, agrupamos todos os processos já implementados no aplicativo e finalizamos as funcionalidades de gravação e leitura no banco de dados.

Você poderá executar o aplicativo em vários dispositivos móveis para testar os dados em tempo real. Se desejar, acesse o Firebase Console, disponível em: <https://console.firebase.google.com/>, e remova alguns alunos para testar o comportamento do aplicativo. Você fez um ótimo trabalho, parabéns!

Avançando na prática

Aplicativo Android para Imobiliária

Descrição da situação-problema

Uma empresa especializada no setor imobiliário deseja disponibilizar um aplicativo Android para os seus clientes. O objetivo do aplicativo é listar um catálogo de imóveis disponíveis para venda e aluguel. Você foi contratado para atuar com a equipe de programadores responsável por este projeto. Em reunião com

a equipe, ficou decidido que o aplicativo usará um banco de dados na nuvem para gravar e ler os dados de cada imóvel disponível. Dessa forma, o catálogo sempre estará atualizado entre todos os usuários conectados ao aplicativo. Sua tarefa nesta etapa de desenvolvimento é criar um projeto no Android Studio e conectá-lo ao Realtime Database. Em seguida, você deverá instanciar um objeto DatabaseReference para que você consiga gravar e ler os dados do banco de dados na nuvem. Ao finalizar, apresente para a equipe de programadores para que o projeto possa continuar. Bom trabalho!

Resolução da situação-problema

Para iniciar a tarefa que foi delegada a você pela equipe de programadores, crie um projeto no Android Studio.

1) Acesse o menu Tools, em seguida Firebase.

2) A partir da janela Assistent, conecte o projeto ao produto Realtime Database.

3) Acesse a classe MainActivity.java e declare um objeto FirebaseDatabase para recuperar uma instância do banco de dados.

4) Declare também um objeto DatabaseReference para recuperar a raiz do projeto na estrutura do Realtime Database.

5) Para finalizar, chame pelo método child() e informe como parâmetro a String "Imovel" para que todos os imóveis possam ser cadastrados nesta referência.

Essa é a primeira etapa finalizada do projeto. Quais são os próximos passos que você sugere para a equipe de desenvolvimento?

Faça valer a pena

1. Realtime Database é o banco de dados NOSQL da plataforma Firebase. Firebase (2018) garante que todos os usuários conectados ao banco de dados são capazes de receber atualizações instantâneas sempre que novos dados são inseridos.
O código abaixo exemplifica como recuperar uma instância do Realtime Database e em seguida como definir uma referência para que o programador possa gravar ou ler dados no banco de dados.

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference reference = database.getReference();
```

Qual alternativa apresenta a linha de comando correta para inserir um objeto Venda com um código gerado automaticamente na referência "raizDoProjeto/movimentacao/venda/"?

a) reference.child("venda").push().setValue(movimentacao).
b) reference.child("movimentacao").child("venda").child(objetoVenda).
c) reference.child("movimentacao/venda").child("venda").
d) reference.child("movimentacao").setValue(objetoVenda).
e)reference.child("movimentacao").child("venda").push().setValue(objetoVenda).

2. ChildEventListener é uma interface utilizada para receber alterações em uma determinada referência do Realtime Database. Essa interface oferece métodos específicos para que o programador possa recuperar sempre que um objeto é incluído, atualizado, excluído ou movido. Cada método de retorno recebe como parâmetro um objeto DataSnapshot que contém dados da referência do Realtime Database.

Assinale a alternativa que contém todos os métodos disponibilizados pela interface ChildEventListener.

a) onChildInserted(), onChildUpdated(), onChildRemoved(), onChildMoved().
b) onChildInserted(), onChildRemoved(), onChildMoved(), onChildCancelled().
c) onChildAdded(), onChildChanged(), onChildRemoved(), onChildMoved(), onCancelled().
d) onChildAdded(), onChildChanged(), onChildRemoved(), onChildMoved().
e) onChildAdded(), onChildChanged(), onChildRemoved(), onCancelled().

3. Por meio de um objeto DatabaseReference é possível gravar e ler dados do Realtime Database. Caso o programador queira monitorar os dados em tempo real de alguma referência do banco de dados é possível adicionar um objeto ChildEventListener ou ValueEventListener. O código abaixo monitora em tempo real todas as modificações da referência "raizDoProjeto/pessoas/colaboradores/".

```
referente.child("pessoas").child("colaboradores")
.addChildEventListener(mChildEventListener);
```

Assinale a alternativa que contém o código responsável por remover o Listener adicionado a uma referência do Realtime Database.

a) reference.removeEventListener(mChildEventListener).
b) reference.removeAllListeners().
c) reference.removeChildEventListener().
d) reference.remove().
e) reference.getReference().removeEventListener().

Referências

ANDROID Studio. Version 3.1.3. [S.l.]: Google, JetBrains, 2018. Disponível em: <https://developer.android.com/studio/>. Acesso em: 24 jun. 2018.

AMAZON. **O que é o NoSQL?** Disponível em: <https://aws.amazon.com/pt/nosql/>. Acesso em: 29 jun. 2018.

AZURE, Microsoft. **O que é nuvem?** Disponível em: <https://azure.microsoft.com/pt-br/overview/what-is-the-cloud/>. Acesso em: 29 jun. 2018.

DEITEL, P.; DEITEL, H.; WALD, A. **Android 6 para programadores:** uma abordagem baseada em aplicativos. 3. ed. Porto Alegre: Bookman, 2016.

DEVELOPER, A. **Create a List with RecyclerView.** Disponível em: <https://developer.android.com/guide/topics/ui/layout/recyclerview>. Acesso em: 16 jun. 2018.

DEVELOPER, Android. **RecyclerView.** Disponível em: <https://developer.android.com/reference/android/support/v7/widget/RecyclerView>. Acesso em: 16 jun. 2018.

DICIO. **Informação.** Disponível em: <https://www.dicio.com.br/informacao/>. Acesso em: 23 jun. 2018.

DOCS, Oracle. **Interface List.** Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/util/List.html>. Acesso em: 20 jun. 2018.

FIREBASE, Google. **Firebase Console**. Disponível em: <https://console.firebase.google.com/>. Acesso em: 5 jul. 2018.

FIREBASE, Google. **Ler e gravar dados no Android**. Disponível em: <https://firebase.google.com/docs/database/android/read-and-write>. Acesso em: 6 jul. 2018.

FIREBASE, Google. **Estruturar o banco de dados**. Disponível em: <https://firebase.google.com/docs/database/android/structure-data>. Acesso em: 2 jul. 2018.

FIREBASE, Google. **Trabalhar com listas de dados no Android**. Disponível em: <https://firebase.google.com/docs/database/android/lists-of-data>. Acesso em: 6 jul. 2018.

FIREBASE. **Getting Started with the Firebase Realtime Database on Android** - Firecasts. S.i.: Firebase, 2016. (8 min.), son., color. Legendado. Disponível em: <https://www.youtube.com/watch?v=lpFDFK44pX8>. Acesso em: 9 jul. 2018.

MONGODB. **NoSQL Databases Explained.** Disponível em: <https://www.mongodb.com/nosql-explained>. Acesso em: 29 jun. 2018.

ORACLE. **Class Object.** Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals(java.lang.Object)>. Acesso em: 2 jul. 2018.

ISBN 978-85-522-1107-5