PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

AUTOR
RODRIGO PLOTZE

PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

AUTOR
RODRIGO PLOTZE

1ª EDIÇÃO
SESES
RIO DE JANEIRO 2017

Estácio

Conselho editorial ROBERTO PAES E PAOLA GIL DE ALMEIDA

Autor do original RODRIGO PLOTZE

Projeto editorial ROBERTO PAES

Coordenação de produção PAOLA GIL DE ALMEIDA, PAULA R. DE A. MACHADO E ALINE KARINA RABELLO

Projeto gráfico PAULO VITOR BASTOS

Diagramação ULISSES VITTORI

Revisão linguística BFS MEDIA

Imagem de capa ZAOZAA19 | SHUTTERSTOCK.COM

Dados Internacionais de Catalogação na Publicação (CIP)

P729p PLOTZE, RODRIGO

Programação para dispositivos móveis. / Rodrigo Plotze.

Rio de Janeiro: SESES, 2017.

136 P: IL.

ISBN: 978-85-5548-428-5

1.PROGRAMAÇÃO. 2. MOBILE. 3. APLICATIVOS. 4. DADOS. I. SESES. II. ESTÁCIO.

CDD 004

Diretoria de Ensino — Fábrica de Conhecimento

Rua do Bispo, 83, bloco F, Campus João Uchôa

Rio Comprido — Rio de Janeiro — RJ — CEP 20261-063

Sumário

Prefácio

Prezados(as) alunos(as),

Na disciplina de Programação para Dispositivos Móveis você aprenderá sobre a construção de aplicativos para aparelhos celulares (smartphones) e tablets. Você estudará a plataforma de desenvolvimento Android, criada pela Google, e explorará os diversos recursos para desenvolvimento dos mais variados tipos de aplicativos.

O conteúdo da disciplina começa apresentando o cenário tecnológico da área de programação para dispositivos móveis, evidenciando as várias tecnologias existentes, e detalhando as características e funcionalidades da plataforma Android. Você aprenderá a instalar e configurar um ambiente de desenvolvimento integrado que será utilizado para criar os aplicativos para dispositivos móveis. A disciplina utilizará a linguagem de programação Java para desenvolvimento dos projetos, e a codificação será realizada no ambiente Android Studio.

A construção de aplicativos para Android é baseada em um conjunto essencial para confecção das interfaces gráficas com o usuário. Você compreenderá como utilizar controles para entrada de dados, tais como campos de texto, selecionadores e botões. Além disso, você conhecerá os recursos da plataforma para exibição de mensagens para o usuário por meio de caixas de diálogo. A disciplina aborda ainda, o uso de controles para exibição de grandes conjuntos de dados.

Na disciplina você estudará também, os recursos para construir aplicativos com múltiplas telas, ou, no contexto da plataforma Android, aplicativos com múltiplas atividades. Finalmente, você aprenderá a desenvolver aplicativos capazes de armazenar dados de maneira persistente. Para isso, você utilizará o banco de dados SQLite, o qual é oferecido nativamente em todos os dispositivos que executam o sistema operacional Android.

Bons estudos!

1

Introdução a Plataforma Android

Introdução a Plataforma Android

A construção de aplicações para dispositivos móveis tem se tornado o mais importante segmento no cenário de desenvolvimento de *software*. Nos últimos anos, a quantidade de aplicações construídas para estes dispositivos aumentou exponencialmente. Neste capítulo você aprenderá sobre os fundamentos da plataforma Android, bem como, as etapas de criação, compilação e execução dos aplicativos no Android.

OBJETIVOS

- Conhecer a plataforma para desenvolvimento de aplicativos Android.
- Entender o processo de instalação e configuração do ambiente de desenvolvimento Android Studio;
- Aprender a criar, compilar e executar aplicativos Android.

Introdução a Programação para Dispositivos Móveis

Os avanços tecnológicos no cenário da computação tem impactado diretamente o cotidiano das pessoas. Este fato pode ser notado pela popularização dos computadores, em função da diminuição dos preços, bem como pela facilidade de interação entre os usuários e as máquinas, o que possibilitou um aumento expressivo na quantidade de pessoas que usam estes dispositivos.

Nos últimos anos, os avanços tecnológicos tem buscado a mobilidade na utilização dos computadores, de forma que qualquer usuário possa usar um dispositivo computacional em qualquer lugar. Neste contexto, os dispositivos móveis, tais como, telefones celulares ou tablets, tem se tornado verdadeiros computadores pessoais. Os telefones celulares, por exemplo, que antes eram utilizados essencialmente para o envio e recebimento de chamadas de voz e texto, hoje em dia tem como principal funcionalidade o uso de aplicativos para os mais variados propósitos. A evolução é tão surpreendente que os aparelhos celulares são conhecidos atualmente como smartphones (telefones inteligentes).

Para atender esta demanda foram criadas diversas plataformas para construção de aplicações para dispositivos móveis. Estas plataformas, fornecem ao

desenvolvedor recursos para elaboração de aplicações tão sofisticadas quanto as aplicações construídas para computadores desktop ou para internet. Nesse contexto, as principais soluções para construção de aplicações para dispositivos móveis e portáteis são:

ADOBE AIR	Plataforma da Adobe para desenvolvimento e execução de aplicativos construídos em HTML, JavaScript, ActionScript, Adobe Flash em qualquer ambiente computacional.
ANDROID	Plataforma da Adobe para desenvolvimento e execução de aplicativos construídos em HTML, JavaScript, ActionScript, Adobe Flash em qualquer ambiente computacional. Android: sistema operacional para dispositivos móveis desenvolvido pelo Google. Este sistema possibilita a construção de aplicativos para *smartphones* e *tablets*, além de permitir o desenvolvimento para televisores (Android TV), carros (Android Auto) e relógios de pulso (Android Wear). Os aplicativos são construídos na linguagem de programação Java e C/C++.
BLACKBERRY	Os aplicativos desta plataforma são focados no mercado corporativo e são executados em um sistema operacional específico conhecido como BlackBerry OS. A construção pode ser realizada utilizando a linguagem de programação C e C++.
IOS	Sistema operacional da Apple integrado nos dispositivos iPhone, iPod, iPad e Apple TV. O sistema oferece diversos recursos para construção de aplicativos, os quais são executados exclusivamente nos aparelhos da Apple.
JAVA ME	Representa a porção da plataforma Java específica para a construção de *software* embarcados. As aplicações são elaboradas por meio da linguagem de programação Java.
SYMBIAN	Sistema operacional da Nokia para smartphones.
WINDOWS PHONE	Sistema operacional desenvolvido pela Microsoft que faz parte da família Windows Mobile (ou Windows CE), e permite a elaboração de aplicações com as linguagens de programação C#, Visual Basic, C e C++

CONCEITO

A plataforma Java ME foi por muitos anos a principal forma de desenvolvimento de aplicações para dispositivos móveis. Atualmente, o foco da plataforma não é a construção de aplicações para smartphones, mas sim, o desenvolvimento de aplicações para dispositivos embarcados. Este novo cenário da plataforma explora uma área da computação conhecida como Internet das Coisas (Internet of Things), que possibilita a execução de aplicativos, e o acesso a internet, nos mais variados tipos de dispositivos, tais como, microcontroladores, sensores, televisores, impressoras, entre outros.

CONEXÃO

O sistema operacional iOS da Apple tem como principal atrativo a facilidade de interação, o que possibilita que os mais variados tipos de usuário utilizem o sistema. No site oficial do fabricante é possível encontrar detalhes adicionais sobre o sistema, bem como, os requisitos para construção de aplicações: <https://www.apple.com/ios/>.

Nesta disciplina você aprenderá a programar dispositivos móveis, mais especificamente smartphones e tablets, utilizando a plataforma Android.

Fundamentos da Plataforma Android

A plataforma Android é a solução de código aberto (*open-source*) do Google para o desenvolvimento de aplicações para dispositivos móveis. Segundo Lecheta (2010), o Android fornece uma solução completa para construção de aplicativos para *smartphone*s e *tablets*, incluindo um sistema operacional baseado em Linux, com uma interface visual rica e uma série de sensores de movimento, ambiente e posicionamento. A programação dos aplicativos pode ser realizada utilizando a linguagem Java, o que permite ao desenvolvedor explorar os mais variados tipos de recursos oferecidos pela linguagem.

As funcionalidades oferecidas pela plataforma Android são definidas por um consórcio de empresas conhecido como *Open Handset Alliance* (OHA). Este consórcio é formado por mais de oitenta empresas, tais como, fabricantes de aparelhos celulares (HTC, LG, Motorola, Samsung, Sony, entre outras), operadoras de

telefonia móvel (China Mobile, Sprint Nextel, Telecom Italia, Telefónica, Vodafone, entre outras), companhias de desenvolvimento de *software* (eBay, Google, Myriad, Nuance, PacketVideo, entre outras) e fabricantes de semicondutores (ARM, Broadcom, Intel, Nvidia, Qualcomm, Texas Instruments, Athreos, entre outras). Os padrões especificados pelo consórcio garantem a compatibilidade dos recursos da plataforma, e facilitam, o processo de desenvolvimento e execução de aplicativos (OHA, 2014).

A primeira versão da plataforma Android (Alpha 1.0) foi disponibilizada pelo consórcio OHA em setembro de 2008, a qual era executada no *smartphone HTC Dream* da empresa HTC, também conhecido como *T-Mobile G1*. Os recursos oferecidos pela plataforma ainda eram limitados, mas já demonstravam grandes novidades no cenário de desenvolvimento de aplicações para dispositivos móveis. Esta versão utilizava a *API level 1* da plataforma Android.

A versão 1.1 (API level 2) foi disponibilizada em fevereiro de 2009, e em abril deste mesmo ano foi apresentada a versão 1.5 *Cupcake* (API level 3), incluindo importantes características na plataforma, tais como, opção de auto-rotação e transição de telas animadas. Ainda em 2009 foi lançado o Android 1.6 (API level 4).

As versões 2.X da plataforma Android começaram com o Android 2.0 Eclair (API level 5) até o Android 2.3.7 Gingerbread (API level 10). Segundo Android Developers (2014a), existem cerca de 10% dos aparelhos utilizando as API´s da versão 2.X do Android. As versões 3.X da plataforma foram disponibilizadas para execução especificamente em *tablets*. O primeiro dispositivo a utilizar a versão Android 3.0 Honeycomb (API level 13) foi o Motorola Xoom em fevereiro de 2011.

As versões da série 4.X começaram com o Android 4.0 e 4.0.2 denominado Ice Cream Sandwich (API level 14), incluindo grandes modificações visuais na interface gráfica. A última versão lançada da série foi Android 4.4 KitKat (API level 20). Por fim, a versão atual da plataforma disponibilizada para comunidade é a versão Android 5.0 Lollipop (API level 21).

A plataforma Android possui uma arquitetura modular divididas em cinco camadas, as quais são denominadas: Linux Kernel; Libraries; Android RunTime; Application Framework e Applications. A figura 1 ilustra a arquitetura da plataforma Android.

? CURIOSIDADE

Os nomes das versões da plataforma Android possuem uma curiosidade interessante. A partir da versão 1.5, todos os lançamentos representam um nome de uma sobremesa

(desert) na língua inglesa, tais como, Android Cupcake 1.5, Android Donut 1.6, Android Eclar 2.0, entre outros. Outro detalhe é que a primeira letra de cada versão segue a ordem alfabética.

CONEXÃO

O site oficial de desenvolvedores Android, denominado Android Developers, fornece um quadro de informações sobre o número de dispositivos que utilizam determinadas versões da plataforma. Estes dados são muito importantes para os desenvolvedores, pois trazem indicativos a respeito do uso da plataforma. Mais detalhes podem ser visualizados em:

<https://developer.android.com/about/dashboards/index.html>.

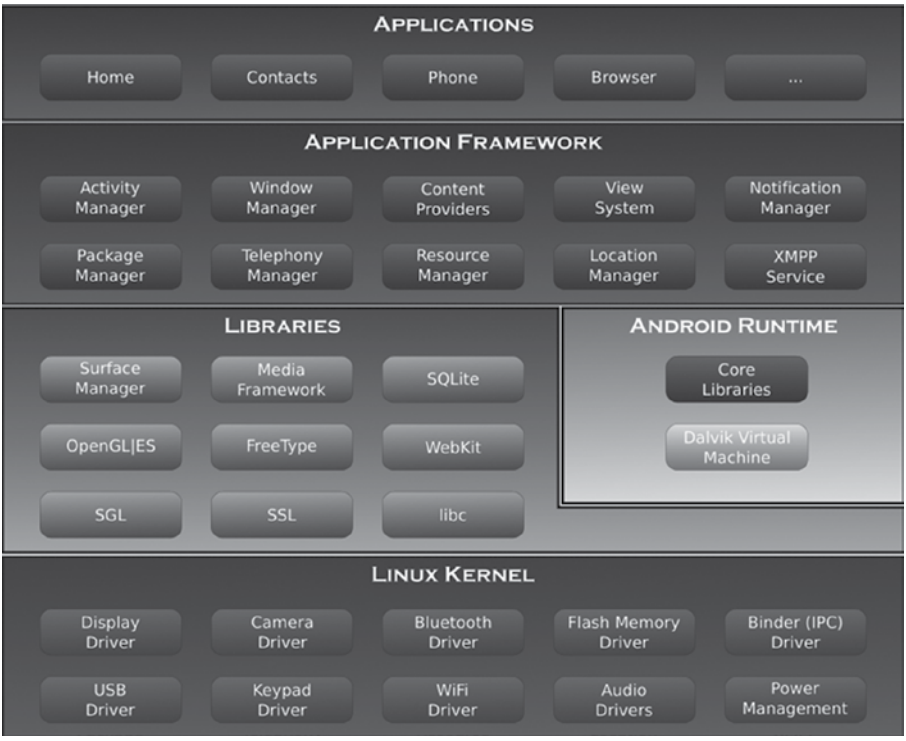

Figura 1.1 – Arquitetura da Plataforma Android

A primeira camada da plataforma é denominada Linux Kernel e fornece todos os recursos fundamentais do sistema operacional Linux para execução das

aplicações no dispositivo móvel. Nesta camada, são encontrados os drivers para compatibilidade dos aparelhos com as aplicações, os serviços de gerenciamento de energia, memória e processo, além dos protocolos para comunicação em rede.

Na camada de bibliotecas (libraries) são disponibilizados os recursos para construção das aplicações. Com isso, grande parte da complexidade de acesso aos recursos do sistema é simplificado, além disso, são oferecidos nativamente um sistema de banco de dados (SQLite), um navegador web (Webkit), geração de gráficos 2D e 3D, entre outros.

No Android RunTime, o desenvolvedor encontra as funcionalidades da linguagem de programação Java. Assim, é possível incluir os pacotes da plataforma Java (Coleções, Entrada/Saída, entre outros) durante a construção das aplicações no Android. Outra característica importante desta camada é a máquina virtual Java, conhecida na plataforma Android como Dalvik. Cada processo na plataforma Android é executada na sua própria instancia da máquina virtual. A máquina virtual Dalvik é responsável pela execução dos arquivos com extensão .DEX, os quais são otimizados para o consumo mínimo de memória dos dispositivos. Esses arquivos são semelhantes aos arquivos .CLASS gerados no processo de compilação das aplicações Java.

Na camada *Application Framework*, são oferecidos aos desenvolvedores API´s (*Application Programming Interface*) para acessar os recursos dos dispositivos de maneira simples e descomplicada. Com isso, é possível construir aplicações que podem, por exemplo, transmitir facilmente dados utilizando bluetooh, ou ainda, exibir avisos para o usuário na janela de notificações. Esta camada fornece ainda, controles para elaboração das interfaces gráficas, tais como, caixas de texto, botões, listas, entre outros.

A camada de Aplicação (*Application*) disponibiliza para o usuário os aplicativos que podem ser utilizados no dispositivo. Assim, cada aplicativo instalado no dispositivo fica acondicionado na camada de aplicação. Como exemplo de aplicativos temos, gerenciador de contatos, cliente para e-mail, navegador, calculadora, bloco de anotações, jogos, entre outros.

Ambiente de Desenvolvimento

A principal linguagem de programação utilizada para construção de aplicações para Plataforma Android é a linguagem Java. Dessa forma, o primeiro passo, antes mesmo de escolher o ambiente de desenvolvimento, é realizar a instalação e configuração da Plataforma Java. Inicialmente, você pode verificar qual a versão do

Java Runtime Environment (JRE), que está instalado no seu computador. O JRE é responsável pela execução das aplicações Java e é fortemente recomendado que você tenha a versão mais atual instalada no seu computador. Para isso, realize o seguinte procedimento:

1. Visitar a página oficial da Oracle, que é o fornecedor da plataforma Java, para verificação da instalação e configuração da plataforma Java no seu computador. O endereço é: <http://java.com>.

2. Clique no botão "Download Gratuito do Java", você será conduzido a página de download. Assim, é possível obter a versão mais atual clicando no botão "Concordar e Iniciar Download Gratuito". Por fim, execute o arquivo que foi efetuado *download* para iniciar o processo de instalação.

O próximo passo é realizar a instalação e configuração do Kit de Desenvolvimento Java, do inglês Java Development Kit, ou simplesmente JDK. Para isso, execute as seguintes etapas:

1. Acessar a página da Oracle para download do programa de instalação do JDK. O endereço é :<http://www.oracle.com/technetwork/pt/java/javase/downloads/index.html>.

Na página de *download*, você deve clicar no ícone Java Platform (JDK), tal como é apresentado na figura 1.2.

Figura 1.2 – Download do Java Development Kit (JDK).

Em seguida, você deve aceitar o contrato de licença (*Accept License Agreement*) e escolher a versão do JDK de acordo com as características do sistema operacional que você tem instalado no seu computador, conforme ilustrado na figura 1.3. Com isso, o processo de *download* do JDK é iniciado. Escolha uma pasta no seu computador para salvar o arquivo.

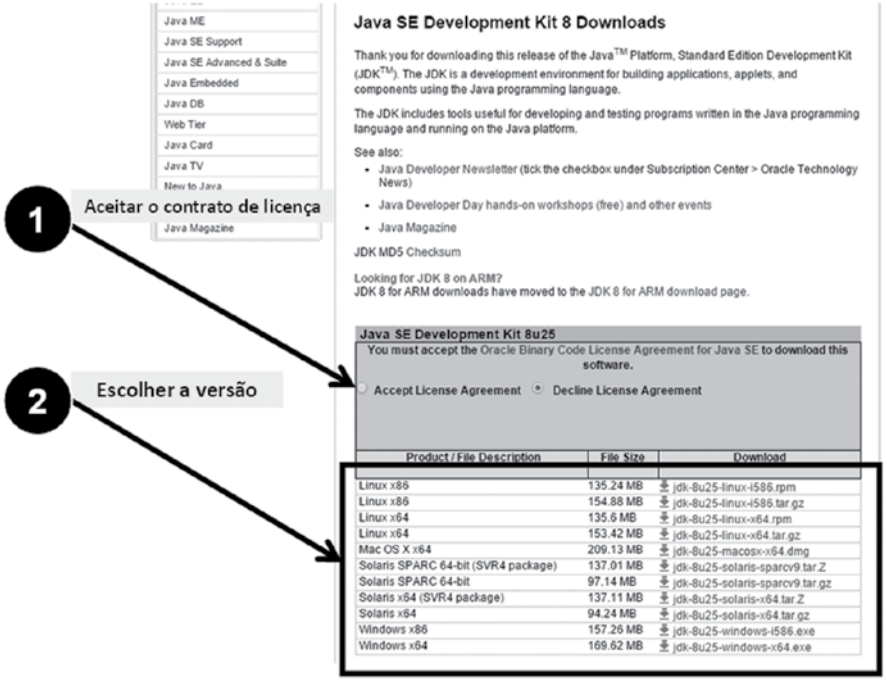

Figura 1.3 – Download do Java Development Kit (JDK).

2. Finalizado o processo de download do JDK você pode realizar a instalação clicando duas vezes no programa. O processo de instalação é bem simples, e você terá que apenas clicar algumas vezes no botão denominado próximo (next). A figura 1.4 ilustra as telas da instalação do Java Development Kit.

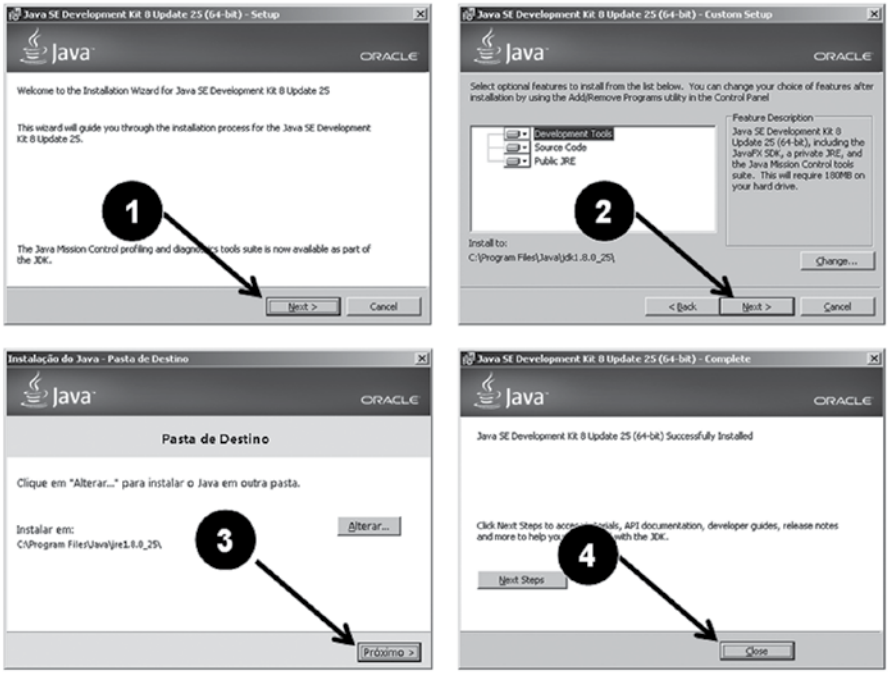

Figura 1.4 – Instalação do Java Development Kit (JDK).

3. Para terminar este processo é necessário configurar a variável de ambiente JAVA_HOME, a qual será utilizada pelo ambiente de desenvolvimento para acessar as bibliotecas disponíveis na plataforma Java. No sistema operacional Windows esta configuração é realizada acessando a janela Sistema > Configurações Avançadas do Sistema, em seguida, deve ser acionada a aba Avançado e escolher o botão Variáveis de Ambiente. Na tela Variáveis de Ambiente, utilize o botão novo do grupo Variáveis do sistema para adicionar a variável JAVA_HOME. Você precisa também, atualizar a variável PATH, adicionando o caminho da instalação do JRE. Este processo é demonstrado detalhadamente na figura1.5.

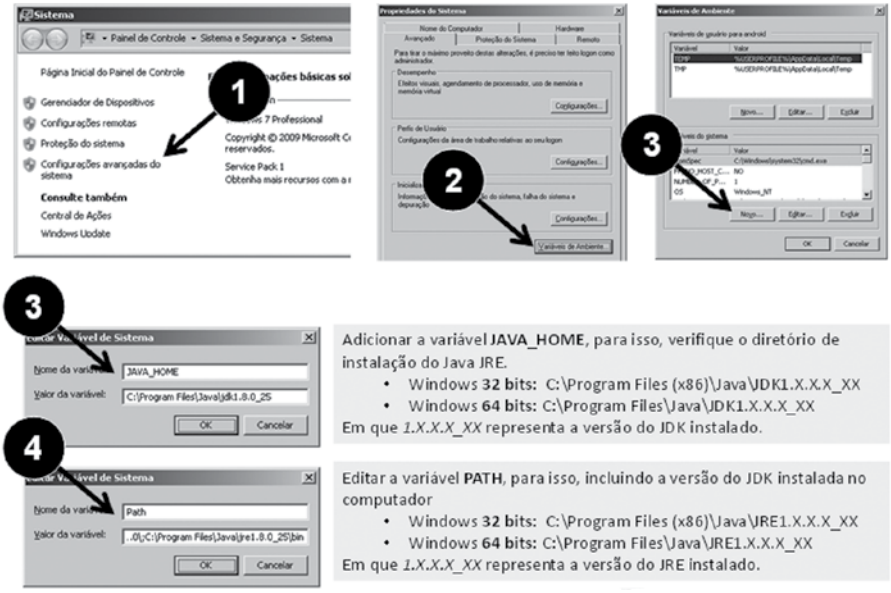

Figura 1.5 – Configuração das Variáveis de Ambiente da Plataforma Java.

Concluída a instalação do JDK você já pode escolher o ambiente de desenvolvimento para começar a construir as suas primeiras aplicações na plataforma Android.

Atualmente, os dois principais ambientes de desenvolvimento integrados para plataforma Android são o Eclipse e o Android Studio. Estes ambientes podem ser obtidos gratuitamente na internet e estão disponíveis para os principais sistemas operacionais, tais como, Windows, Linux e MacOS.

O Eclipse é um ambiente para desenvolvimento muito flexível e permite a construção de aplicativos em várias linguagens, tais como Java, C/C++, PHP, Python, entre outras. Para o desenvolvimento de aplicativos Android no Eclipse é necessário a instalação de um plugin conhecido como Android Development Tools (ADT). Este *plugin* realiza uma personalização no ambiente, facilitando a construção de aplicações para Android, bem como, reduzindo o tempo para desenvolvimento e execução dos projetos. No site oficial de desenvolvedores Android é possível obter uma instalação do Eclipse que incluir o ADT. O endereço para acesso é:

<https://developer.android.com/sdk/index.html>

Embora o ambiente Elipse seja um ambiente adequado para construção de projetos para plataforma Android. Nesta disciplina utilizaremos o ambiente Android Studio, uma vez, que no próprio site oficial de desenvolvedores existe uma mensagem indicando que o suporte ao Eclipse com o ADT já não está mais ativo, sendo recomendando o uso do Android Studio como ambiente oficial.

O ambiente Android Studio é uma nova plataforma para desenvolvimento de aplicações para Android. Este ambiente oferece recursos adicionais em relação ao Eclipse e segundo recomendações do Google será o ambiente oficial para construção de aplicações. O Android Studio é baseado no ambiente IntelliJ IDEA e está disponível para as plataformas Windows, MacOS e Linux. Conforme observado anteriormente, nesta disciplina será utilizado o ambiente Android Studio.

Android Software Development Kit (Android SDK)

Para o desenvolvimento de aplicações para Android é necessária a instalação de um kit específico conhecido como Android SDK. Neste kit é encontrada todas as bibliotecas para construção de aplicações, bem como, as ferramentas para os desenvolvedores. Além disso, o Android SDK fornece um emulador para teste das aplicações. Para obter o Android SDK, realize o seguinte procedimento:

1. Acessar a página para download do Android SDK e clicar no item "Get the SDK for an Existing IDE", em seguida acionar o botão "Download the stand-alone Android SDK Tools for Windows". Após aceitar os temos e as condições de uso o download será iniciado. A página para obtenção do Android SDK é: <https://developer.android.com/sdk/index.html>

2. Concluído o download do Android SDK, você pode clicar duas vezes sobre o arquivo que o processo de instalação será iniciado. As etapas de instalação são simples, assim, você pode clicar na opção próximo seguidas vezes. Ao final, você visualizará o botão finish, e certifique-se que a opção Start SDK Manager esteja marcada.

A aplicação *SDK Manager* é responsável pelo gerenciamento das plataformas disponíveis para o desenvolvimento dos aplicativos no Android. Com esta aplicação, você poderá realizar o *download* das plataformas, de acordo com a sua necessidade. Por exemplo, caso você tenha interesse em desenvolver um aplicativo para versão Android 4.1 *Jelly Bean* (API level 14), você poderá realizar o *download* das

bibliotecas específicas para esta versão. Além disso, é possível obter também um emulador com as características de *hardware* semelhantes a esta versão. No SDK *Manager* você encontra também aplicações Extras como, por exemplo, para distribuição de um aplicativo na Google Play, ou ainda, o *driver* USB para execução dos aplicativos diretamente nos dispositivos. Caso ainda não tenha iniciado o SDK Manager, você pode realizar esta tarefa no sistema operacional Windows a partir do *Menu Iniciar* > *Android SDK Tools* > *SDK Manager*. A figura 1.6 apresenta a tela inicial do SDK Manager. Na Figura 6 é importante notar a localização da instalação do SDK, veja em SDK Path o caminho completo em que os arquivos foram instalados.

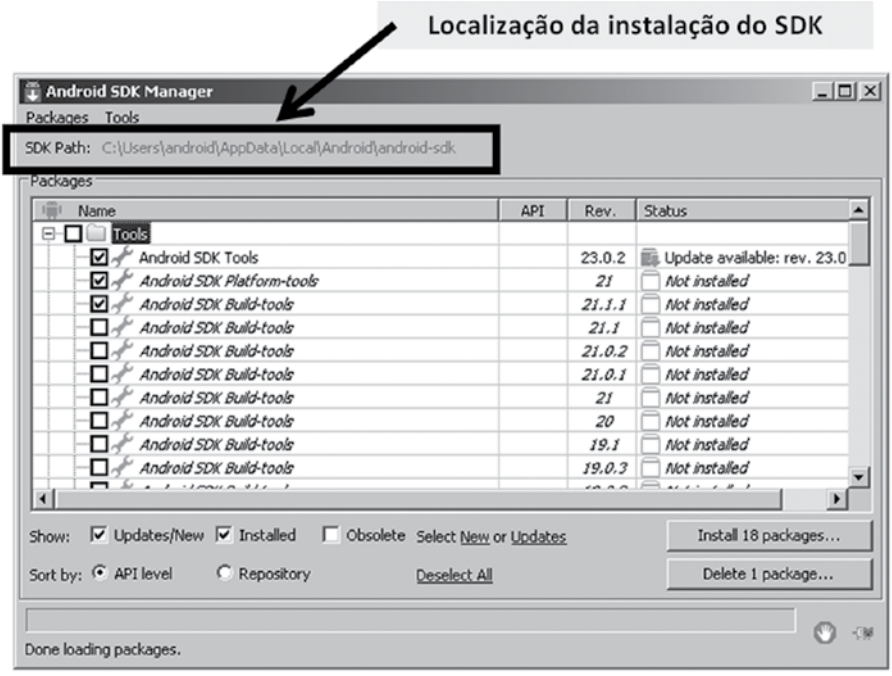

Figura 1.6 – Android SDK Manager.

Neste momento você realizará o *download* das versões da plataforma Android que deseja desenvolver aplicativos. Você pode escolher qualquer versão de API, no entanto, por questões de estabilidade é aconselhável utilizar versões a partir da API 16 (Android 4.1.2). Todos os projetos apresentados ao longo dos próximos capítulos serão totalmente compatíveis com a API 16 ou superior. A figura 1.7 ilustra a seleção e instalação a versão 4.1.2 no Android SDK Manager. Veja na figura 1.7 que é fundamental selecionar todos os itens que fazem parte da API 16:

SDK Plataform, Samples for SDK, ARM EABI System Image, Intel x86 Atom System Image, MIPS System Image, Google APIs e Sources for Android SDK.

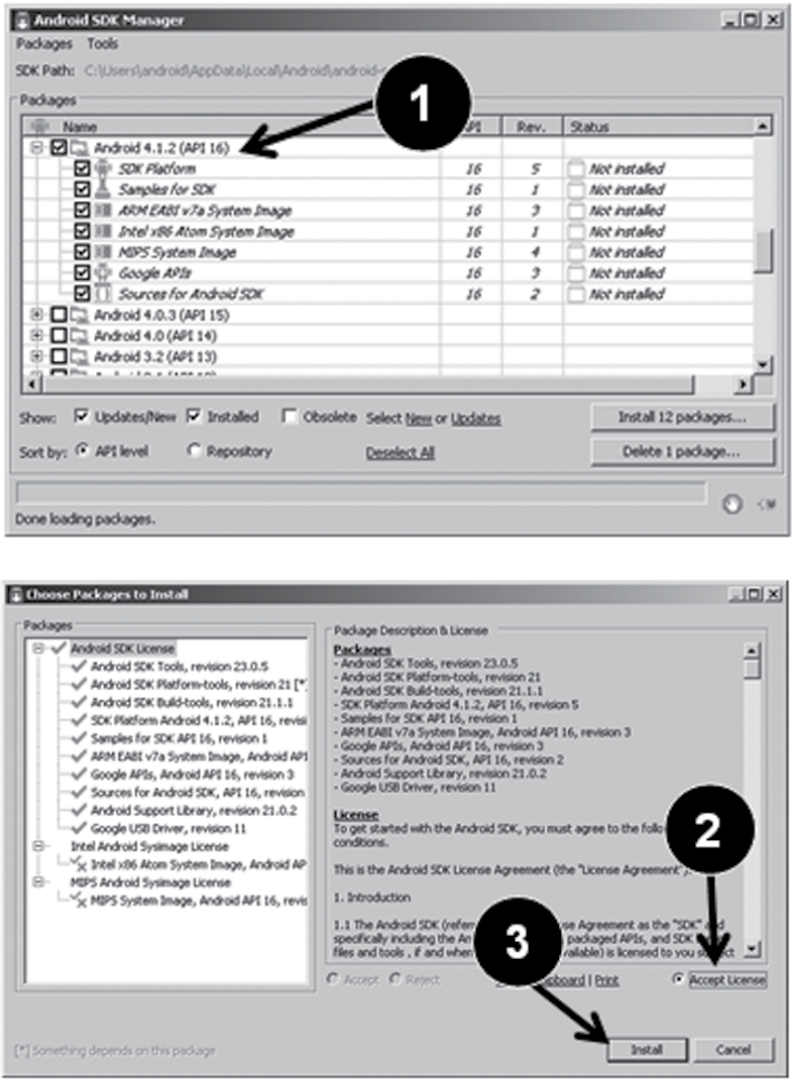

Figura 1.7 – Instalação da API no SDK Manager.

Instalação e Configuração do Android Studio

Para iniciar o desenvolvimento de aplicações para Android com o ambiente Android Studio o primeiro passo é a obtenção do arquivo de instalação. Para isso, realize as seguintes etapas:

3. Acionar o botão "Download Android Studio" no seguinte endereço: <https://developer.android.com/sdk/installing/studio.html>.

4. Após concordar com os termos e condições para uso do software o processo de download será iniciado. Você deverá escolher uma pasta no seu computador para salvar o arquivo.

5. Finalizado o *download* do arquivo, você precisará descompactar o instalador. Uma sugestão é descompactar o arquivo no diretório raiz do seu computador (C:\). A figura 1.8 apresenta o resultado da descompactação do arquivo de instalação.

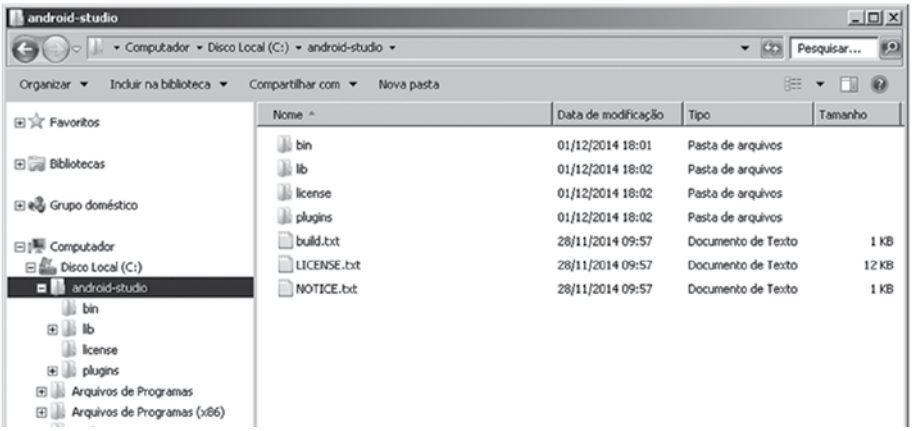

Figura 1.8 – Descompactação do arquivo de Instalação do Android Studio.

Para iniciar a execução do ambiente você deverá utilizar o arquivo que está dentro da pasta bin do diretório de instalação. Na pasta você encontrará dois arquivos que podem ser utilizados, e você deverá selecionar o arquivo de acordo com a versão do sistema operacional instalado no seu computador. Para simplificar as próximas inicializações do ambiente é recomendada a criação de um atalho na área de trabalho. A figura 1.9 ilustra o arquivos para iniciar o ambiente de desenvolvimento Android Studio. Você deverá utilizar o arquivo ***studio.exe*** para a versão 32 bits do Windows, enquanto o arquivo ***studio64.exe*** é indicado para a versão 64 bits.

Figura 1.9 – Inicialização do Ambiente Android Studio.

Finalmente, você visualizará a tela inicial do ambiente em que será possível criar novos aplicativos ou ainda, abrir projetos previamente existentes. A figura 1.10 apresenta a tela inicial do ambiente Android Studio.

Figura 1.10 – Tela Inicial do Ambiente Android Studio.

Na primeira vez que você iniciar o ambiente Android Studio é necessário indicar o diretório em que o Android SDK foi instalado. Caso tenha dúvidas, veja

na Figura 1.6 como obter a localização da instalação do Android SDK. Na tela inicial do Android Studio, você deverá acionar o item *Configure* > *Project Defaults* > *Project Structure*. No campo Android SDK Location, você deverá indicar o diretório de instalação do SDK, e ao final clicar no botão OK.

Criação, Compilação, Execução de Aplicativos

Para criação de um novo projeto no Android Studio, você deverá selecionar o item *New Project* na tela inicial do ambiente. O próximo passo é a configuração do seu novo projeto. No campo *Application name* você pode especificar o nome *App01*, e o *Company Domain* defina o nome do pacote base como sendo *android.br*. Em seguida, clique no botão *next*.

Na próxima etapa você precisará escolher para qual versão da plataforma a sua aplicação é compatível. Neste momento, você precisa indicar uma versão disponível no *SDK Manager*. Conforme comentando anteriormente, todos os projetos apresentados ao longo dos próximos capítulos serão compatíveis com a versão API 16 ou superior. Assim, selecione a API 16 e clique no botão próximo.

A próxima tela permite a definição do tipo de atividade que será criada. Na programação Android uma Atividade (*Activity*) representa uma tela da aplicação, você pode selecionar o item *Blank Activity* e selecionar o botão *next*. Finalmente, você poderá definir o nome da atividade (*Activity Name*), o qual deve ser especificado com o nome de Principal. Em seguida, clique no botão *Finish* para concluir a criação do projeto. A Figura 1.11 apresenta o processo de criação de um novo projeto no Android Studio. Na figura é possível visualizar a etapa inicial e a etapa final.

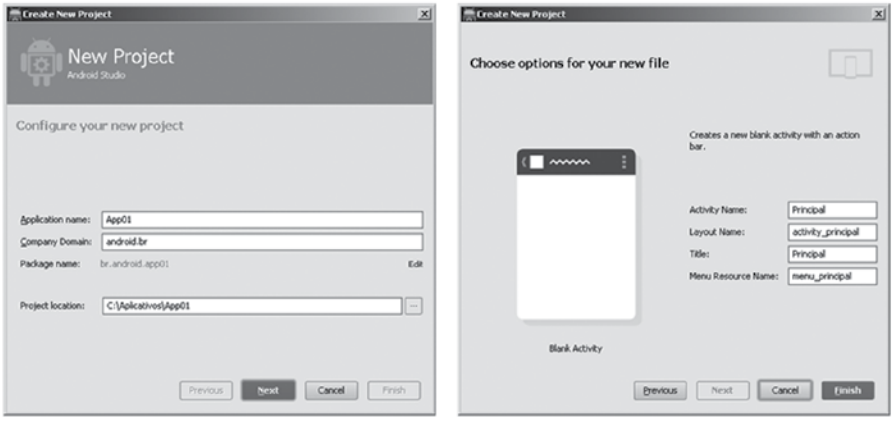

Figura 1.11 – Etapas de Criação de um Novo Projeto no Android Studio.

Após acionado o botão *finish* você precisará aguardar alguns instantes até que todos os arquivos necessários para execução do projeto estejam sincronizado no seu computador. Neste momento, alguns arquivos são obtidos na internet, assim, é fundamental que sua conexão esteja ativa e operante.

O processo de compilação de projetos no ambiente Android Studio é bastante simples. Para isso, é necessário acionar o menu *Build* em seguida selecionar a opção *Make Project*, ou simplesmente pressionar a tecla de atalho CTRL+F9.

A execução dos aplicativos construídos no ambiente Android Studio pode ser realizada de duas maneiras. Na primeira você pode criar um dispositivo virtual que emula todas as características de *hardware* e *software* do aparelho. Para isso, é necessário utilizar uma aplicação disponível no Android SDK conhecida como Android Virtual Device (AVD). A segunda opção é realizar a execução diretamente no aparelho. Neste caso, é necessário configurar o seu aparelho para permitir a execução de aplicativos de fontes desconhecidas (Menu de configurações de aplicativos) e, ativar a depuração USB no menu de desenvolvimento. Nos exemplos apresentados ao longo dos próximos capítulos utilizaremos a primeira opção.

Para criação de um novo dispositivo virtual para emulação da aplicação, no ambiente Android Studio acesse o menu T*ools* > *Android* > *AVD Manager*, em seguida clique no botão *Create a Virtual Device*. Na tela de seleção do *hardware* (*Select Hardware*), escolha a categoria Phone, e na listagem de aparelhos você pode escolher Nexus S, em seguida clique no botão *next*. Você deverá então escolher a imagem do dispositivo para emulação. Neste momento, serão listadas as imagens de acordo com as versões da API que você instalou no *SDK Manager*. Neste, e nos demais exemplos, utilizaremos a imagem Jelly Bean, para API level 16, para o processador armeabi-v7a, com a versão alvo (*target*) *Android 4.1.2*. Clicando no botão próximo, você poderá conferir todas as configurações do dispositivo virtual, além disso, é possível especificar o nome do AVD. Neste momento, você pode clicar no botão *finish* para concluir a criação. A figura 1.12 apresenta o resultado do processo de criação do Android Virtual Device (AVD).

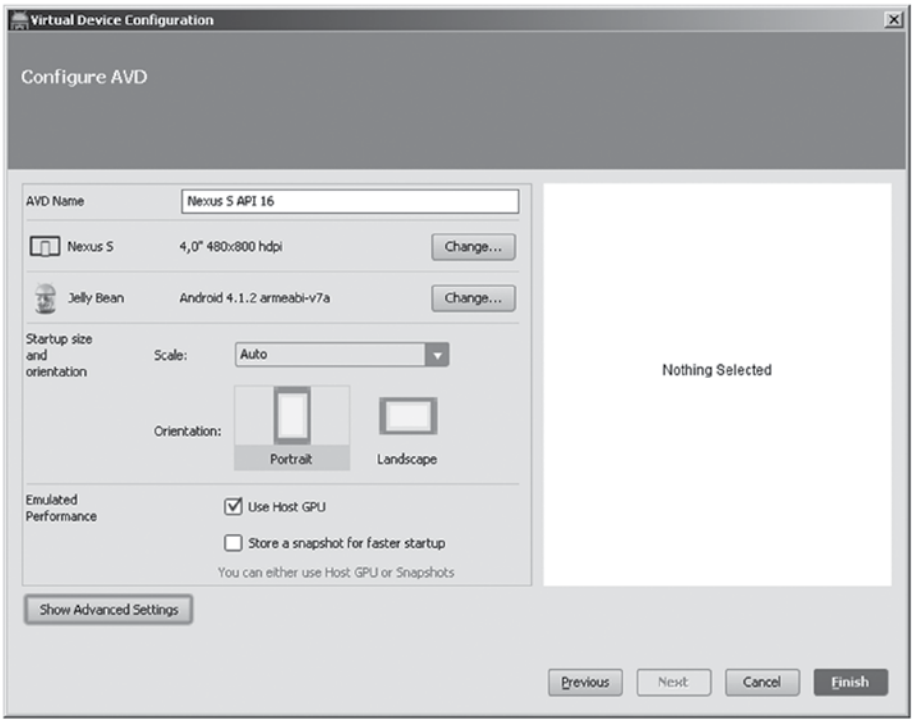

Figura 1.12 – Criação de um Android Virtual Device (AVD).

Finalmente podemos executar a aplicação acessando o menu *Run* > *Run 'app'*, ou simplesmente pressionando a tecla de atalho SHIFT + F10. Você visualizará a tela *Choose Device*, em que você pode escolher em qual dispositivo seu aplicativo será executado. Na tela você encontrará a opção *Launch Emulator*, em que é possível indicar qual AVD deverá ser utilizado na execução do aplicativo.

Na primeira vez que o emulador é executado o processo é bem demorado. Assim, tenha paciência até que seu aplicativo apareça no emulador. Este tempo pode ser influenciado pelas características de *hardware* e *software* do seu computador. O resultado final, com a aplicação em execução, pode ser visualizado na figura 1.13.

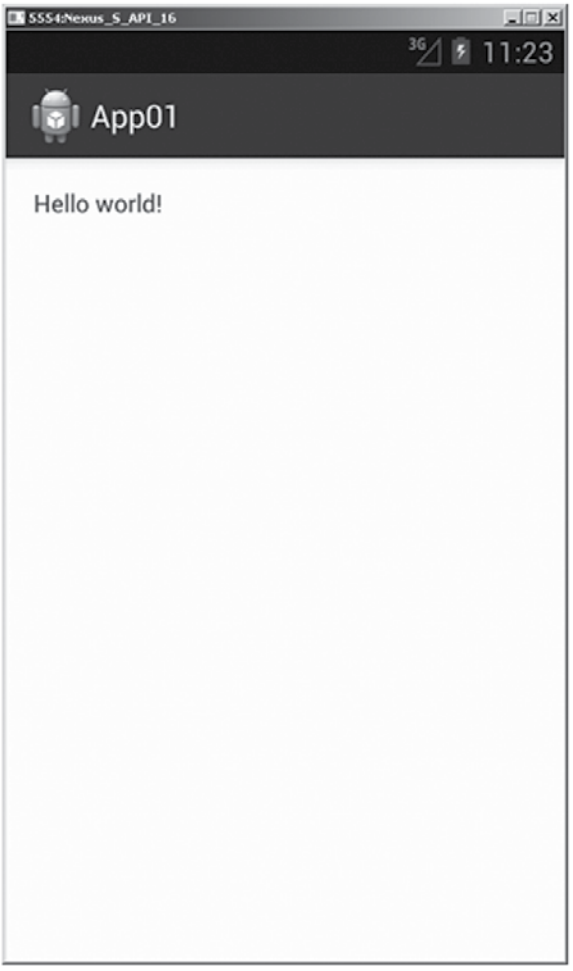

Figura 1.13 – Aplicativo Executado no Emulador.

Com isso, finalizamos o conteúdo deste primeiro capítulo. Para o próximo capítulo é fundamental que você tenha conseguido instalar e configurar todos os recursos apresentados neste capítulo.

ATIVIDADES

Com base no conteúdo apresentado neste capítulo, responda as seguintes questões objetivas:

01. Qual plataforma para desenvolvimento de aplicativos para dispositivos móveis tem como foco o mercado corporativo?

a) Android
b) BlackBerry
c) iOS
d) Java ME
e) Windows Phone

02. Complete a sentença: Atualmente, a plataforma _____ _____ tem como principal objetivo o desenvolvimento de aplicações para dispositivos da Internet das Coisas.

a) Android
b) BlackBerry
c) iOS
d) Java ME
e) Windows Phone

03. Qual versão da plataforma Android tem o codinome KitKat?

a) 2.3.3
b) 3.0
c) 4.0
d) 4.1
e) 4.4

REFLEXÃO

A computação vestível, do inglês, weareable computer, é uma das grandes apostas do mercado de desenvolvimento de aplicações para dispositivos móveis. O objetivo é construir dispositivos que possam ser vestidos pelas pessoas, da mesma maneira que você precisa vestir uma blusa ou uma calça. Neste contexto, faça uma reflexão sobre a seguinte questão: Como a computação vestível influenciará no cotidiano das pessoas?

LEITURA

Para complementar seu aprendizado sobre a plataforma Android você pode consultar a obra Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK do autor Ricardo R. Lecheta. A referência completa é a seguinte: LECHETA, R. R. Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK. 2 ed. São Paulo: Novatec Editora, 2010.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDROID DEVELOPERS. Dashboards. 2014a. Disponível em: <https://developer.android.com/about/dashboards/index.html>, acesso em novembro de 2014.

LECHETA, R. R. Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK. 2 ed. São Paulo: Novatec Editora, 2010.

OHA. Open Handset Alliance - Members. Disponível em <http://www.openhandsetalliance.com/oha_members.html>, acesso em novembro de 2014.

2

Construção de Aplicativos para Android: Controles Fundamentais

Construção de Aplicativos para Android: Controles Fundamentais

Neste capítulo você aprenderá sobre como construir aplicações utilizando a plataforma Android. Para isso, é necessário que você já tenha instalado e configurado o ambiente de desenvolvimento Android Studio no seu computador. Você conhecerá os controles fundamentais para construção de interfaces gráficas e utilizará o emulador para visualizar as suas aplicações em funcionamento.

OBJETIVOS

Com o estudo deste capítulo você aprenderá sobre:

- os controles básicos para construção de aplicações para Android, tais como: TextView, EditText e Button;
- os modelos para tratamento de eventos nas interfaces gráficas e a exibição de mensagens utilizando caixas de diálogos.

Componentes das Aplicações Android

A plataforma Android tem uma estrutura de componentes altamente modular, o que permite o reaproveitamento de código entre as aplicações. Os componentes são divididos em quatro tipos, os quais possuem características e funcionalidades distintas. Esses componentes são denominados:

- Atividades (*Activities*)
- Serviços (*Services*)
- Provedores de Conteúdo (*Content Providers*)
- Receptores de Transmissão (*Broadcat Receivers*)

Nas aplicações Android uma Atividade (*Activity*) é utilizada para representar uma tela de interface com o usuário. As aplicações podem ser formadas por várias atividades, cada qual funcionando de maneira independente e podem realizar a troca de informações entre elas. Em termos de programação, toda atividade é derivada (ou herda) uma classe denominada *Activity* do pacote *android.app.Activity.* Esta superclasse fornece o método fundamental para criação da

atividade conhecido como onCreate. No método *onCreate* é possível especificar todas as característica que deverão ser inicializadas na *interface*. Em aplicações com múltiplas atividades, é necessário indicar qual atividade é definida como principal, ou seja, qual atividade a primeira atividade que será carregada quando a aplicação for inicializada. A indicação da atividade principal é realizada no arquivo *AndroidManifest.xml*, o qual será abordado nas seções seguintes (ACTIVITIES, 2014).

O componente de Serviço (*Service*) tem como principal objetivo a execução de tarefas em *background*. Um serviço pode ser utilizado, por exemplo, para executar um arquivo de música no formato MP3, enquanto o usuário interage com a aplicação. Uma característica importante dos serviços é o fato de não possuir *interface* gráfica, além disso, permite a execução de operações durante um longo tempo sem interferir na interação do usuário com a aplicação (SERVICES, 2014).

Os provedores de conteúdo (*Content Providers*) são mecanismos utilizados para tornar os dados da aplicação disponíveis para outras aplicações no dispositivo. O compartilhamento de dados é realizado apenas quando desejado e possibilita, por exemplo, que dados armazenados em um arquivo possam ser consultados ou modificados por outros aplicativos. Esse recurso também pode ser empregado em base de dados criadas no banco de dados SQLite, ou em termos gerais, em qualquer tipo de dados armazenado de maneira persistente. Outro exemplo interessante do uso de provedores de conteúdo, é o acesso as informações dos contatos armazenados no dispositivos. O provedor pode ser empregado para inserir, ou editar, informações dos contatos, desde que a aplicação tenha permissões suficientes para realizar esta tarefa (CONTENT PROVIDERS, 2014).

? CURIOSIDADE

O Google Play <play.google.com> e o serviço do Google para distribuição de conteúdo para dispositivos móveis. Neste serviço, que funciona como uma loja virtual, é possível encontrar aplicações para os mais variados propósitos, além disso, o usuário pode realizar o download de conteúdo gratuitos ou pago, tais como livros, filmes, músicas e jogos.

O componente *Broadcast Receiver* é utilizado no processo de comunicação para o recebimento de mensagens enviadas pela sistema operacional Android para as aplicações. Este componente realiza as tarefas em segundo plano (*background*)

e pode ser utilizado, por exemplo, para receber uma mensagem de "bateria fraca" transmitida pelo sistema operacional do dispositivo, ou ainda, notificar o usuário que o *download* de uma aplicação obtida na *Google Play* terminou. Como o componente de broadcast receiver não possui *interface* gráfica, o tratamento e exibição das mensagens pode ser realizado utilizando notificações na barra de status (BROADCAST RECEIVER, 2014).

Manifesto da Aplicação (AndroidManifest.xml)

As configurações gerais das aplicações Android são definidas em um arquivo específico denominado Manifesto da Aplicação. Este arquivo é armazenado na pasta base da aplicação, com o nome de AndroidManifest.xml, e é o primeiro arquivo a ser carregado durante a inicialização da aplicação. No arquivo é possível configurar, por exemplo, a permissão de um aplicativo para acessar a internet, ou ainda, qual é a atividade principal da aplicação. A listagem Código 1 apresenta um exemplo de um arquivo de manifesto.

Código 1

```
1.    <?xml version="1.0" encoding="utf-8"?>
2.    <manifest
3.    xmlns:android=http://schemas.android.com/apk/res/andro
4.    id package="br.android.app" >
5.           <application
6.              android:allowBackup="true"
7.              android:icon="@drawable/ic_launcher"
8.              android:label="@string/app_name"
9.              android:theme="@style/AppTheme" >
10.             <activity
11.                 android:name=".Principal"
12.                 android:label="@string/app_name" >
13.                 <intent-filter>
14.                        <action
15.   android:name="android.intent.action.MAIN" />
16.                        <category
17.   android:name="android.intent.category.LAUNCHER" />
18.                 </intent-filter>
```

```
19.              </activity>
20.          </application>
21.
22.    </manifest>
```

Estrutura Básica de um Projeto Android

Antes de começar o estudo aprofundamento nos controles para construção de interfaces gráficas para aplicativos Android, você precisa entender claramente os elementos que compõe a estrutura básica de um projeto. Para isso, você deve iniciar o ambiente de desenvolvimento Android Studio e carregar o projeto App01 criado no capítulo 1. Na figura 2.1 você pode visualizar o resultado da abertura do projeto App01 no ambiente Android Studio. Na interface gráfica do ambiente você pode notar a estrutura dos arquivos pertencentes ao projeto, destacado como número 1; a área de edição dos arquivos fonte – número 2; a visualização prévia da interface gráfica da aplicação – número 3; e o resultado do processo de compilação – número 4.

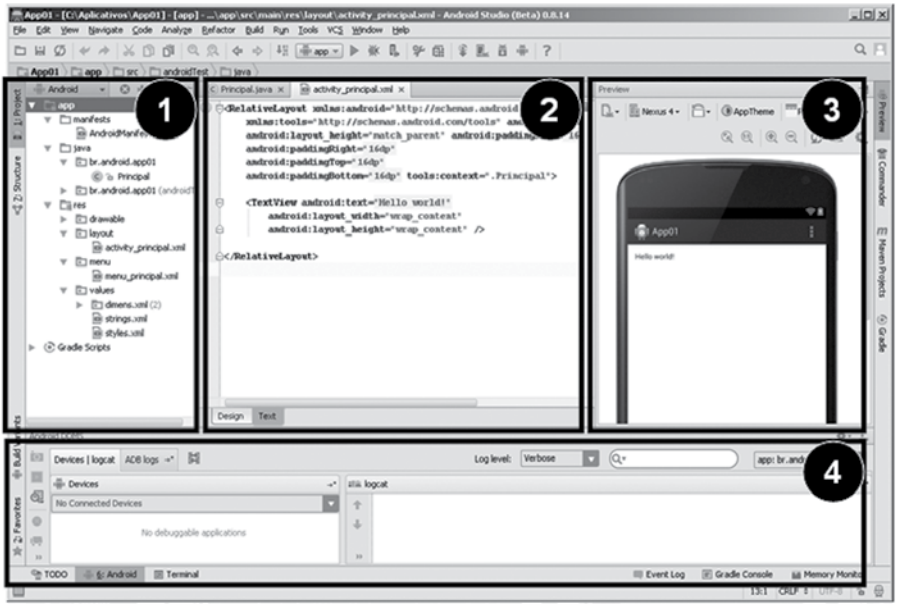

Figura 2.1 – Ambiente Android Studio e a Estrutura do Projeto.

Na composição da estrutura do projeto é possível identificar três grupos principais:

• ***Manifests***: armazena o arquivo de manifesto da aplicação, o qual é denominado *AndroidManifest.xml*. O conteúdo do manifesto pode ser facilmente editado clicando duas vezes sobre o arquivo.

• ***Java:*** contém todos os arquivos da lógica do aplicativo. Nesta pasta são armazenadas as classes, com seus respectivos código na linguagem de programação Java. Todas as vezes que uma nova Atividade é adicionada no aplicativo, um novo arquivo com a extensão .java também é criado. A listagem Código 2 apresenta o conteúdo básico do arquivo lógico de uma Atividade.

Código 2

```
1.     package br.android.app01;
2.
3.     import android.app.Activity;
4.     import android.os.Bundle;
5.     import android.view.Menu;
6.     import android.view.MenuItem;
7.
8.
9.     public class Principal extends Activity {
10.
11.           @Override
12.           protected void onCreate(Bundle savedInstanceState)
13.    {
14.            super.onCreate(savedInstanceState);
15.            setContentView(R.layout.activity_principal);
16.           }
17.
18.
19.           @Override
20.           public boolean onCreateOptionsMenu(Menu menu) {
21.
22.
23.    getMenuInflater().inflate(R.menu.menu_principal,
24.    menu);
```

```
25.              return true;
26.           }
27.
28.     @Override
29.         public boolean onOptionsItemSelected(MenuItem
30.    item) {
31.              int id = item.getItemId();
32.
33.              //noinspection SimplifiableIfStatement
34.              if (id == R.id.action_settings) {
35.              return true;
36.       }
37.
38.         return super.onOptionsItemSelected(item);
39.           }
40.    }
```

Todas as atividades das aplicações Android são derivadas da classe Activity (linha 09) e possuem um método principal denominado onCreate (linha 12). Esse método é utilizado para apresentar a tela da aplicação, a qual é definida em um arquivo de *layout* armazenado na pasta layout do grupo res. Na classe ainda é possível visualizar o método *onCreateOptionsMenu* (linha 20) que permite a exibição de um menu suspenso quando o usuário aciona a tela, os itens que fazem parte do menu devem ser definidos em um arquivo XML e armazenados na pasta menu no grupo res. O tratamento de qual item do menu foi selecionado pelo usuário é realizado com o método on*OptionsItemSelected*, na linha 29.

• *Res*: nesta pasta são armazenados todos os recursos que serão carregados pelo aplicativo. Nesse grupo é possível, por exemplo, incluir imagens para serem utilizadas na construção das atividades. Todas as imagens devem colocadas na pasta drawable e possuir o formato *Portable Network Graphics* (PNG). Na pasta layout são especificados os arquivos relativos as telas da interface. Para cada atividade da aplicação, uma arquivo de layout e definido, assim, é possível definir independentemente os elementos que compõe cada tela do aplicativo. A listagem Código 3 apresenta um exemplo de arquivo de layout. Nesta listagem é possível visualizar o tipo de *layout* utilizado para disposição dos elementos, o qual é denominado *RelativeLayout*. Além disso, o controle *TextView* é utilizado para exibição de um texto na tela do aplicativo.

? CURIOSIDADE

As imagens dos aplicativos Android são armazenadas de acordo com a resolução. Com isso, é possível adequar a qualidade da imagem em função da resolução do aparelho. As resoluções mais utilizas são: MDPI – 160dpi ou 320x480 pixels; HDI – 240dpi ou 480x800 pixels; XHDPI – 320dpi ou 1280x720 pixels (High Definition) e XXHDPI – 480dpi ou 1920x1080 (Full HD).

Código 3

```
1.      <RelativeLayout
2.      xmlns:android=”http://schemas.android.com/apk/res/andro
3.      id”
4.             xmlns:tools=”http://schemas.android.com/tools”
5.      android:layout_width=”match_parent”
6.           android:layout_height=”match_parent”
7.      android:paddingLeft=”@dimen/activity_horizontal_margin
8.      ”
9.
10.     android:paddingRight=”@dimen/activity_horizontal_margi
11.     n”
12.
13.     android:paddingTop=”@dimen/activity_vertical_margin”
14.
15.     android:paddingBottom=”@dimen/activity_vertical_margin
16.     ” tools:context=”.Principal”>
17.
18.       <TextView android:text=”@string/hello_world”
19.            android:layout_width=”wrap_content”
20.            android:layout_height=”wrap_content” />
21.
22.     </RelativeLayout>
23.
```

No grupo de recursos (*Res*) ainda é possível visualizar a pasta menu que pode ser utilizada para armazenar arquivos XML para definição de menus do aplicativo.

A pasta *values* é usada para especificação de valores constantes que podem ser carregados na aplicação. Na pasta é possível, por exemplo, criar o arquivo *strings.xml* contendo toda a informação textual utilizada na aplicação. A listagem Código 4 apresenta um exemplo do arquivo strings.xml, em que pode ser notada as constantes textuais da aplicação, como por exemplo, o nome da aplicação (app_name).

Código 4

```
1.      <?xml version="1.0" encoding="utf-8"?>
2.      <resources>
3.
4.          <string name="app_name">App01</string>
5.          <string name="hello_world">Hello world!</string>
6.          <string name="action_settings">Settings</string>
7.
8.      </resources>
```

CONEXÃO

Na pasta values permite a definição de uma série de arquivos estáticos que podem ser carregados no aplicativo. Dentre os recursos disponíveis estão a especificação de lista de cores; a definição de valores relativos a dimensão dos elementos; entre outros. A lista completa com os recursos que podem ser especificados na pasta values pode ser consultada no endereço:

<https://developer.android.com/guide/topics/resources/more-resources.html>.

Controles para Construção de Interfaces

A construção de interfaces gráficas na plataforma Android é realizada por meio de controles denominados widgets. Esses elementos de interação fazem parte do pacote android.widget e podem ser utilizados para criação dos mais variados tipos de interface. Os controles fundamentais são:

- **Botão (*Button*)**: utilizado para seleção de ações por meio do pressionamento.
- **Campo de Texto (*Text Field*)**: empregado para entrada de dados a partir de um tipo pré-definido: texto, endereço de e-mail, identificador uniforme de recurso (URI), número ou telefone.

• ***CheckBox***: usado para seleção de um item, o qual é identificado por uma marca de verificação. Este controle permite a seleção de múltiplos itens.

• ***Radio Button***: possibilita a seleção mutuamente exclusiva de um item, a partir de um conjunto. Assim, o usuário deve obrigatoriamente escolher um único item.

• ***Spinner***: permite a apresentação de um conjunto de elementos por meio de uma lista suspensa, na qual o usuário pode selecionar o elemento desejado.

Para adicionar os controles nas interfaces gráficas da aplicação é necessário definir qual o tipo de *layout* que será utilizado. O *layout* especifica como os controles serão estruturados visualmente na interface. Os principais tipos de *layout* são:

• ***Linear Layout:*** este é o tipo padrão para construção de interfaces no Android. Os elementos que fazem parte da tela são inseridos sequencialmente a partir da orientação (*orientation*) vertical ou horizontal. Além disso, é possível definir as propriedades de largura (*layout_width*) e altura (*layout_height*) do layout.

• ***Relative Layout***: neste tipo de layout os elementos são organizados na tela de maneira relativa. Com isso, a posição de um controle da interface é definida em relação ao controle adicionado anteriormente.

• ***Table Layout***: utilizado para apresentação de dados na forma tabular, uma vez que é possível especificar linhas e colunas para inserção dos controles.

Outros tipos de layout também podem ser encontrados na plataforma, tais como: *FrameLayout, GridLayout*.

Para demonstrar a utilização dos controles básicos da plataforma Android, bem como, a definição do layout da aplicações você construirá uma nova aplicação denominada App02. O nome da atividade principal será Principal e a versão da API utilizada será a 16. Caso tenha dúvida sobre este processo, você poderá consultar a seção Criação, Compilação e Execução do Capítulo 1. O objetivo do aplicativo será calcular o Índice de Massa Corpórea de uma pessoa. Para isso, o usuário precisará entrar com os dados de peso e altura, o programa efetuará os cálculos necessários, e apresentará o resultado na tela.

O primeiro passo para construção do aplicativo é a especificação das informações textuais que serão utilizadas na atividade. Para isso, na pasta de recursos (res), abra o arquivo strings.xml e codifique as informações de acordo com a listagem Código 5.

Código 5

```
1.      <?xml version=”1.0” encoding=”utf-8”?>
2.      <resources>
3.
4.           <string name=”app_name”>Índice de Massa Corpórea
5.      (IMC)</string>
6.
7.             <string name=”txt_peso”>Peso</string>
8.             <string name=”txt_altura”>Altura</string>
9.             <string name=”txt_resultado”>Resultado</string>
10.
11.            <string name=”btn_calcular”>Calcular</string>
12.            <string name=”btn_limpar”>Limpar</string>
13.
14.            <string
15.     name=”action_settings”>Configurações</string>
16.
17.     </resources>
18.
```

Em seguida, você deverá incluir os elementos que fazem parte da interface gráfica. Para isso, realize as edições no arquivo activity_principal.xml na pasta de recursos *layout*. Na listagem Código 6 você pode conferir as especificações da *interface* gráfica do aplicativo. É importante destacar que cada elemento da interface que será programado, necessita de um identificador (Id). Este identificador é especificado na propriedade id e possui o seguinte formato: android:id=”@+id/nome_do_identificador”.

Código 6

```
1.      <LinearLayout
2.      xmlns:android=”http://schemas.android.com/apk/res/andr
3.      oid”
4.         xmlns:tools=”http://schemas.android.com/tools”
5.      android:layout_width=”match_parent”
6.         android:layout_height=”match_parent”
7.      android:paddingLeft=”@dimen/activity_horizontal_margin”
```

```
8.      android:paddingRight="@dimen/activity_horizontal_margin"
9.      android:paddingTop="@dimen/activity_vertical_margin"
10.     android:paddingBottom="@dimen/activity_vertical_margin"
11.        tools:context=".Principal"
12.        android:orientation="vertical" >
13.
14.        <TextView android:text="@string/txt_peso"
15.          android:layout_marginTop="20sp"
16.          android:layout_width="match_parent"
17.          android:layout_height="wrap_content" />
18.
19.        <EditText
20.            android:id="@+id/edtPeso"
21.            android:layout_width="match_parent"
22.            android:layout_height="wrap_content"
23.            android:inputType="numberDecimal"  />
24.
25.        <TextView android:text="@string/txt_altura"
26.            android:layout_marginTop="20sp"
27.            android:layout_width="match_parent"
28.            android:layout_height="wrap_content" />
29.
30.        <EditText
31.            android:id="@+id/edtAltura"
32.            android:layout_width="match_parent"
33.            android:layout_height="wrap_content"
34.            android:inputType="numberDecimal" />
35.
36.        <TextView android:text="@string/txt_resultado"
37.            android:layout_marginTop="20sp"
38.            android:layout_width="match_parent"
39.            android:layout_height="wrap_content" />
40.
```

```
41.       <EditText
42.           android:id="@+id/edtResultado"
43.           android:enabled="false"
44.           android:layout_marginBottom="20sp"
45.           android:layout_width="match_parent"
46.           android:layout_height="wrap_content"
47.           android:inputType="numberDecimal" />
48.
49.        <LinearLayout
50.           android:layout_width="match_parent"
51.           android:layout_height="wrap_content"
52.           android:orientation="horizontal" >
53.
54.       <Button
55.           android:id="@+id/btnCalcular"
56.           android:text="@string/btn_calcular"
57.           android:layout_width="wrap_content"
58.           android:width="120sp"
59.           android:layout_height="wrap_content" />
60.
61.       <Button
62.           android:id="@+id/btnLimpar"
63.           android:text="@string/btn_limpar"
64.           android:layout_width="wrap_content"
65.           android:width="120sp"
66.               android:layout_height="wrap_content" />
67.           </LinearLayout>
68.    </LinearLayout>
```

Neste exemplo, a estrutura de layout utilizada foi LinearLayout com o tipo de orientação vertical (linha 12). Na tela do aplicativo foram adicionados controles do tipo *TextView* (linhas 14, 25 e 36), os quais permitem a exibição de informações textuais como se fossem rótulos de texto. Além disso, para a entrada de dados foi usado o controle *EditText* em que o tipo de entrada foi configurada como número (linhas 19, 30 e 41). O campo de texto para exibição do resultado não foi habilidade, assim, a propriedade enabled foi definida como *false*. Alguns

elementos da interface foram configurados com a propriedade *layout_margin* que permite especificar distâncias entre os elementos (linha 15). Por fim, dois botões foram criados para realização dos cálculos e limpeza dos elementos da tela (linhas 54 e 61). Para cada elemento adicionado na interface é necessário definir o tipo de layout utilizado para largura (*layout_width*) e altura (layout_height). Os parâmetros possíveis para estas propriedades são *match_parent* – em que o elemento da interface ocupa toda a área do dispositivo; *wrap_content* – no qual o tamanho do elemento é definido de acordo com a quantidade de informação contida nele.

A próxima etapa da codificação do aplicativo é a implementação da lógica da aplicação. Para isso, é necessário editar o arquivo Principal.java localizado na pasta java. A listagem Código 7 apresenta a lógica de negócios do aplicativo.

Código 7

```
1.     package br.android.app02;
2.
3.     import android.app.Activity;
4.     import android.os.Bundle;
5.     import android.view.Menu;
6.     import android.view.MenuItem;
7.     import android.view.View;
8.     import android.widget.Button;
9.     import android.widget.EditText;
10.
11.
12.    public class Principal extends Activity implements
13.    View.OnClickListener {
14.
15.         //ATRIBUTOS
16.           private EditText edtPeso;
17.           private EditText edtAltura;
18.           private EditText edtResultado;
19.           private Button btnCalcular;
20.           private Button btnLimpar;
21.
22.           @Override
23.           protected void onCreate(Bundle savedInstanceState)
```

```
24.    {
25.                super.onCreate(savedInstanceState);
26.             setContentView(R.layout.activity_principal);
27.
28.             //REFERENCIAR OS CONTROLES
29.             edtPeso                                    =
30.    (EditText)findViewById(R.id.edtPeso);
31.             edtAltura                          =
32.    (EditText)findViewById(R.id.edtAltura);
33.             edtResultado                               =
34.    (EditText)findViewById(R.id.edtResultado);
35.             btnCalcular                        =
36.    (Button)findViewById(R.id.btnCalcular);
37.             btnLimpar                          =
38.    (Button)findViewById(R.id.btnLimpar);
39.
40.             //ASSOCIAR TRATADOR DE EVENTOS
41.               btnCalcular.setOnClickListener(this);
42.             btnLimpar.setOnClickListener(this);
43.    }
44.
45.    @Override
46.    public boolean onCreateOptionsMenu(Menu menu) {
47.
48.    getMenuInflater().inflate(R.menu.menu_principal,
49.    menu);
50.              return true;
51.           }
52.
53.           @Override
54.           public boolean onOptionsItemSelected(MenuItem
55.    item) {
56.                  int id = item.getItemId();
57.                  if (id == R.id.action_settings) {
58.                         return true;
59.                  }
```

```
60.
61.                 return super.onOptionsItemSelected(item);
62.     }
63.
64.           // TRATATOR DE EVENTOS
65.           @Override
66.           public void onClick(View v) {
67.                 if ( v.getId() == R.id.btnCalcular){
68.                       double peso =
69.     Double.parseDouble(edtPeso.getText().toString());
70.                       double altura =
71.     Double.parseDouble(edtAltura.getText().toString());
72.                 double resultado =
73.     (peso/Math.pow(altura,2));
74.
75.     edtResultado.setText(String.format("%.2f",resultado));
76.                 }else if ( v.getId() == R.id.btnLimpar){
77.                       edtPeso.setText("");
78.                       edtAltura.setText("");
79.                       edtResultado.setText("");
80.                       edtPeso.requestFocus();
81.                 }
82.           }
83.     }
```

No código é possível notar a declaração de cinco atributos (linhas 16 até 20): *edtPeso, edtAltura, edtResultado, btnCalcular e btnLimpar*. Estes atributos serão utilizados para referenciar os elementos da interface gráfica do aplicativo. Para isso, é preciso realizar a chamada do método *findViewById* que recebe como parâmetro o Id do elemento da interface.

Para realizar o tratamento de eventos dos botões é necessário implementar a *interface View.OnClickListener* (linha 12), a qual cria automaticamente o método *OnClick* (linha 66). Neste método serão incluídos os códigos relativos a interação do usuário com os botões. O resultado da execução da aplicação no emulador é apresentado na figura 2.2.

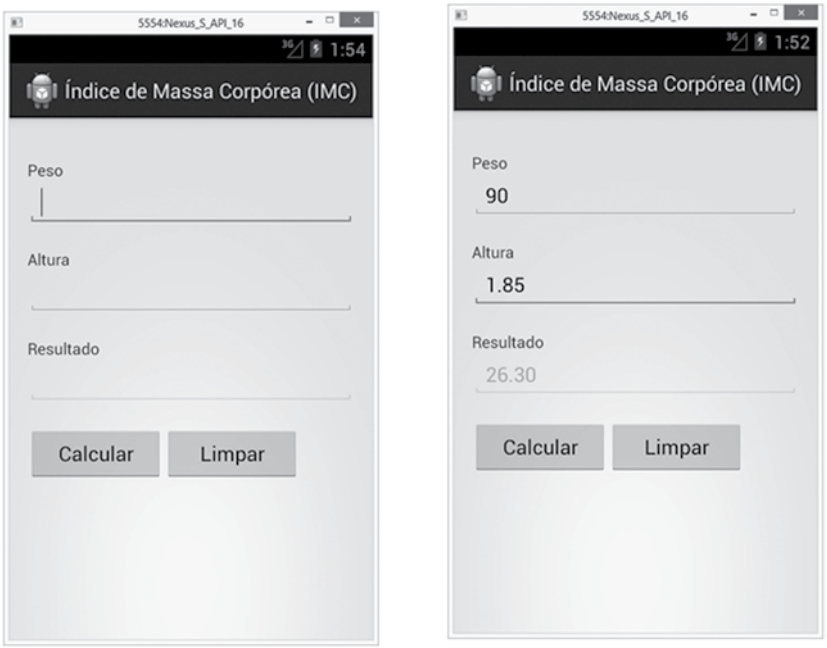

Figura 2.2 – Resultado da Execução do Aplicativo no Emulador.

Caixas de Diálogos (Dialogs)

As caixas de diálogos são recursos fundamentais na construção de interfaces gráficas dos aplicativos, uma vez, que podem ser utilizadas para exibir mensagens para o usuário. A plataforma Android possui uma série de classes para exibição de caixas de diálogos, dentre elas:

- ***AlertDialog:*** para exibição de mensagens de texto;
- ***DatePickerDialog***: para seleção de uma data;
- ***TimePickerDialog***: para escolha de uma hora;
- ***ProgressDialog***: para construção de uma barra de progresso, a qual pode ser utilizada para indicar que algum tipo de processamento está sendo realizado pelo aplicativo.

Outro recurso que pode ser utilizado para exibição de mensagens de texto é o uso da classe *Toast*. Com isso, é possível criar uma pequena mensagem que será exibida ao usuário durante alguns segundos. A vantagem do *Toast* é a sua simplicidade de criação, além disso, não necessita interação do usuário, pois desaparece

após algum tempo. A listagem Código 8 apresenta o uso da classe *AlertDialog* para exibição do resultado do cálculo do Índice de Massa Corpórea da App02. Neste exemplo, ao invés de exibir o resultado dentro do campo de texto (edtResultado), uma caixa de diálogo é utilizada para apresentação do IMC. No evento do botão Calcular é demonstra o uso do *AlertDialog*, enquanto, no botão Limpar é apresentada a utilização da classe *Toast.*

CONEXÃO

A estrutura modular dos aplicativos Android, permite a personalização de diversos elementos da interface gráfica. Na internet você pode encontrar diversos serviços para geração automática de recursos gráficos para as aplicações. Com estes geradores é possível criar esquemas de cores personalizados para os controles que fazem parte do aplicativo. Um bom exemplo deste serviço é o Android Holo Colors Generator, que pode ser acessado em: <http://android-holo-colors.com/>.

Código 8

```
1.      @Override
2.      public void onClick(View v) {
3.             if ( v.getId() == R.id.btnCalcular){
4.                    double peso =
5.      Double.parseDouble(edtPeso.getText().toString());
6.                    double altura =
7.      Double.parseDouble(edtAltura.getText().toString());
8.                    double resultado =
9.      (peso/Math.pow(altura,2));
10.
11.                   AlertDialog.Builder dlg = new
12.     AlertDialog.Builder(this);
13.                   dlg.setTitle(R.string.app_name);
14.                   dlg.setMessage(String.format("IMC= %.2f",
15.     resultado));
16.                   dlg.setPositiveButton("OK",new
17.     DialogInterface.OnClickListener() {
18.                           @Override
```

```
19.                         public void onClick(DialogInterface
20.    dialog, int which) {
21.                         }
22.                 });
23.                 dlg.show();
24.
25.         }else if ( v.getId() == R.id.btnLimpar){
26.                 edtPeso.setText("");
27.                 edtAltura.setText("");
28.                 edtResultado.setText("");
29.                 edtPeso.requestFocus();
30.
31.           Toast.makeText(this,
32.                 "Os dados foram apagados com sucesso!",
33.                         Toast.LENGTH_LONG).show();
34.           }
35.    }
```

O resultado da execução da aplicação pode ser visualizado na figura 2.3. Na figura é possível visualizar o uso da classe *AlertDialog*, bem como, da mensagem com a classe *Toast*.

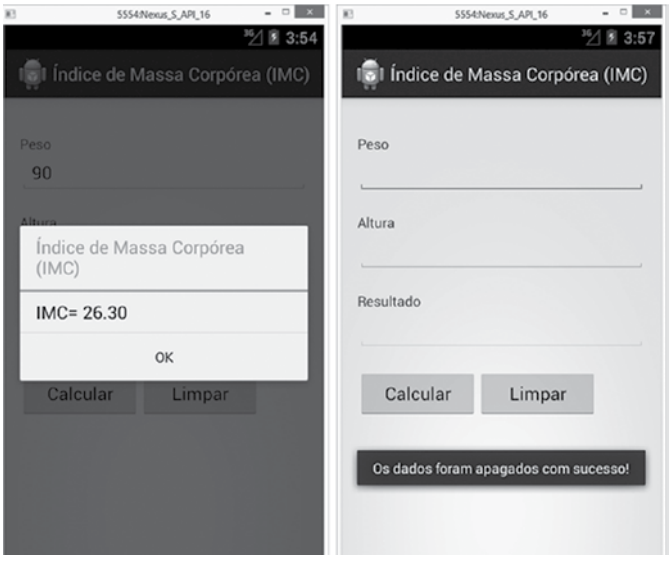

Figura 2.3 – Resultado da Execução do Aplicativo no Emulador.

Com isso, finalizamos este capítulo em que você aprendeu como construir o primeiro aplicativo para plataforma Android.

ATIVIDADES

Para complementar o seu conhecimento, responda as seguintes questões objetivas:

01. Qual controle da plataforma Android permite a entrada de informações textuais?

a) Label
b) EditText
c) TextBox
d) TextField
e) TeatArea

02. Qual componente das aplicações Android é utilizado tornar os dados da aplicação disponíveis para outras aplicações no dispositivo?

a) Activities
b) Android Manifest
c) Broadcast Receivers
d) Content Providers
e) Services

03. Complete a sentença: O componente _____ tem como principal objetivo a execução de tarefas em background.

a) Activities
b) Android Manifest
c) Broadcast Receivers
d) Content Providers
e) Services

REFLEXÃO

O desenvolvimento de aplicações para Android fornece ao programador um universo de possibilidades. No que diz respeito a interfaces gráficas com o usuário a plataforma tem um conjunto extenso de elementos, os quais podem ser utilizados para elaboração de interfaces

dos mais variados tipos. Neste contexto, faça uma reflexão a respeito da seguinte questão: Como é possível desenvolver interfaces gráficas de qualidade para dispositivos móveis?

LEITURA

A plataforma Android oferece ao desenvolvedor uma série de controles visuais para construção de interfaces. Para aprofundar seu conhecimento é sugerida a leitura da documentação oficial da plataforma a respeito do tópico User Interface. O conteúdo está disponível em:

ANDROID DEVELOPERS. User Interface. Disponível em <https://developer.android.com/guide/topics/ui/index.html>, acesso em novembro de 2014.

REFERÊNCIAS BIBLIOGRÁFICAS

ACTIVITIES. Android Developer Guide: Activies. Disponível em <http://developer.android.com/reference/android/app/Activity.html>, acesso em novembro de 2014.

BROADCAST RECEIVER. Android Developer Guide: BroadCast Receiver. Disponível em <http://developer.android.com/reference/android/content/BroadcastReceiver.html>, acesso em novembro de 2014.

CONTENT PROVIDERS. Android Developer Guide: Content Providers. Disponível em <http://developer.android.com/guide/topics/providers/content-providers.html>, acesso em novembro de 2014.

SERVICES. Android Developer Guide: Services. Disponível em <http://developer.android.com/guide/components/services.html>, acesso em novembro de 2014.

3

Construção de Aplicativos para Android: Controles de Seleção

Construção de Aplicativos para Android: Controles de Seleção

Neste capítulo você continuará o aprendizado sobre a construção de aplicativos para plataforma Android. Você estudará sobre o controle para seleção de elementos, em que o usuário poderá escolher itens a partir de opções apresentadas na interface gráfica. Além disso, você conhecerá um controle para exibição de muitos elementos, o qual pode ser utilizado para exibição de conjuntos de dados.

OBJETIVOS

Ao final deste capítulo você será capaz de:

- Utilizar o controle da plataforma Android para seleção múltipla de elementos *CheckBox*.
- Apresentar itens para seleção mutuamente exclusiva utilizando o controle *RadioButton* e o componente *RadioGroup*.
- Exibir conjuntos de elemento por meio de um menu suspenso utilizando o controle *Spiner*.

Controles para Seleção

Os controles de seleção podem ser utilizados para construir interfaces gráficas em que o usuário necessita realizar algum tipo de escolha. Com isso, as opções disponíveis para o usuário aparecem na tela, sendo necessária apenas a seleção do item desejado. A maneira de seleção do item depende do tipo de controle utilizado, assim, existe a possibilidade de: não selecionar o item; de selecionar um ou mais itens; ou ainda de selecionar (obrigatoriamente) apenas um item. Na plataforma Android os controles que permitem seleção são:

- ***CheckBox***: este controle permite que o usuário realize a seleção de um ou mais itens a partir de um conjunto de elementos. O usuário ainda pode optar por não selecionar nenhum item. Caso o usuário tenha escolhido algum item disponível no controle, este receberá um símbolo de marcado (ou checado) (CHECKBOXES, 2014).
- ***Radio Button***: neste tipo de controle o usuário precisa obrigatoriamente de selecionar um item. Além disso, a seleção é mutuamente exclusiva, o que significa que o usuário poderá escolher um único elemento do conjunto (RADIO BUTTONS, 2014).

Para demonstrar o uso do controle de seleção *CheckBox* você criará uma nova aplicação no ambiente de desenvolvimento Android Studio. Este exemplo deverá ser nomeado **App03**. O nome da atividade principal será definido como Principal. Na listagem Código 1 é especificado o conteúdo do arquivo de texto da aplicação, o qual é nomeado *strings.xml*. A listagem Código 2 apresenta o layout da interface gráfica da aplicação. No código é possível notar o uso do controle *CheckBox*, bem como das propriedades que precisam ser configuradas.

CONEXÃO

A plataforma Android possui um tipo de controle que permite a seleção de dois estados, por exemplo, ligado ou desligado. Este tipo de controle é conhecido como *ToggleButton*. A partir da versão 4.0 da plataforma foi introduzido um outro controle para seleção entre dois estados, o qual é denominado Switch. Para mais detalhes você pode consultar: <https://developer.android.com/guide/topics/ui/controls/togglebutton.html>.

Código 1

```
1.      <?xml version=”1.0” encoding=”utf-8”?>
2.      <resources>
3.
4.          <string name=”app_name”>Temas</string>
5.
6.          <string name=”txt_titulo”>Quais os temas do seu
7.      interesse?</string>
8.
9.          <string name=”ckb_cinema”>Cinema</string>
10.         <string name=”ckb_futebol”>Futebol</string>
11.         <string
12.     name=”ckb_gastronomia”>Gastronomia</string>
13.         <string
14.     name=”ckb_informatica”>Informática</string>
15.         <string name=”ckb_literatura”>Literatura</string>
16.         <string name=”ckb_teatro”>Teatro</string>
17.
18.        <string name=”txt_selecionados”>Total de temas
```

```
19.     selecionados</string>
20.
21.         <string name=”btn_exibir”>Exibir</string>
22.         <string name=”btn_desmarcar”>Desmarcar</string>
23.
24.         <string
25.     name=”action_settings”>Configurações</string>
26.
27.     </resources>
```

Código 2

```
1.      <LinearLayout
2.      xmlns:android=”http://schemas.android.com/apk/res/andr
3.      oid”
4.          xmlns:tools=”http://schemas.android.com/tools”
5.      android:layout_width=”match_parent”
6.          android:layout_height=”match_parent”
7.      android:paddingLeft=”@dimen/activity_horizontal_margin
8.      ”
9.
10.     android:paddingRight=”@dimen/activity_horizontal_margi
11.     n”
12.
13.     android:paddingTop=”@dimen/activity_vertical_margin”
14.
15.     android:paddingBottom=”@dimen/activity_vertical_margin
16.     ”
17.         tools:context=”.Principal”
18.         android:orientation=”vertical” >
19.
20.         <TextView
21.             android:text=”@string/txt_titulo”
22.             android:layout_width=”wrap_content”
23.             android:layout_height=”wrap_content”
24.             android:textColor=”#000080”
25.             android:textSize=”18sp”
```

```
26.             android:layout_marginBottom=”20sp”/>
27.
28.         <CheckBox
29.             android:id=”@+id/ckbCinema”
30.             android:layout_width=”match_parent”
31.             android:layout_height=”wrap_content”
32.             android:text=”@string/ckb_cinema”
33.             android:textSize=”16sp”/>
34.
35.         <CheckBox
36.             android:id=”@+id/ckbFutebol”
37.             android:layout_width=”match_parent”
38.             android:layout_height=”wrap_content”
39.             android:text=”@string/ckb_futebol”
40.             android:textSize=”16sp”/>
41.
42.         <CheckBox
43.             android:id=”@+id/ckbGastronomia”
44.             android:layout_width=”match_parent”
45.             android:layout_height=”wrap_content”
46.             android:text=”@string/ckb_gastronomia”
47.             android:textSize=”16sp”/>
48.
49.         <CheckBox
50.             android:id=”@+id/ckbInformatica”
51.             android:layout_width=”match_parent”
52.             android:layout_height=”wrap_content”
53.             android:text=”@string/ckb_informatica”
54.             android:textSize=”16sp”/>
55.
56.         <CheckBox
57.             android:id=”@+id/ckbLiteratura”
58.             android:layout_width=”match_parent”
59.             android:layout_height=”wrap_content”
60.             android:text=”@string/ckb_literatura”
61.             android:textSize=”16sp”/>
```

```
62.
63.         <CheckBox
64.             android:id=”@+id/ckbTeatro”
65.             android:layout_width=”match_parent”
66.             android:layout_height=”wrap_content”
67.             android:text=”@string/ckb_teatro”
68.             android:textSize=”16sp”/>
69.
70.         <LinearLayout
71.             android:layout_width=”match_parent”
72.             android:layout_height=”wrap_content”
73.             android:orientation=”horizontal”
74.             android:layout_marginTop=”20sp”
75.             android:layout_marginBottom=”20sp”>
76.
77.             <Button
78.                 android:id=”@+id/btnExibir”
79.                 android:text=”@string/btn_exibir”
80.                 android:layout_width=”wrap_content”
81.                 android:width=”120sp”
82.                 android:layout_height=”wrap_content” />
83.
84.             <Button
85.                 android:id=”@+id/btnDesmarcar”
86.                 android:text=”@string/btn_desmarcar”
87.                 android:layout_width=”wrap_content”
88.                 android:width=”120sp”
89.                 android:layout_height=”wrap_content” />
90.         </LinearLayout>
91.
92.         <TextView
93.             android:id=”@+id/txtTotalSelecionados”
94.             android:layout_width=”match_parent”
95.             android:layout_height=”wrap_content”
96.             android:text=”@string/txt_selecionados”/>
97.
98.     </LinearLayout>
```

A listagem Código 3 demonstra o uso do controle *CheckBox* na lógica de programação da aplicação. Além disso, o exemplo apresenta o uso do tratador de eventos *OnCheckedChangeListener* que permite identificar a mudança de estado no controle *CheckBox*.

CONCEITO

O estado de um controle *CheckBox* pode ser modificado programaticamente, utilizando o método *setChecked*. Com isso, é possível alterar o estado do controle de selecionado, *setChecked(true)*, ou não selecionado, *setChecked(false)*. Para verificar o status do controle, você pode utilizar o método *isChecked()*.

Código 3

```
1.      package br.android.app03;
2.
3.      import android.app.Activity;
4.      import android.app.AlertDialog;
5.      import android.os.Bundle;
6.      import android.view.Menu;
7.      import android.view.MenuItem;
8.      import android.view.View;
9.      import android.widget.Button;
10.     import android.widget.CheckBox;
11.     import android.widget.CompoundButton;
12.     import android.widget.TextView;
13.     import android.widget.Toast;
14.
15.
16.     public class Principal extends Activity implements
17.     View.OnClickListener,
18.     CheckBox.OnCheckedChangeListener {
19.
20.         private CheckBox ckbCi;
21.         private CheckBox ckbFu;
22.         private CheckBox ckbGa;
23.         private CheckBox ckbIn;
```

```
24.         private CheckBox ckbLi;
25.         private CheckBox ckbTe;
26.         private TextView txtSel;
27.         private Button btnEx;
28.         private Button btnDes;
29.         private int cont;
30.
31.     @Override
32.     protected void onCreate(Bundle savedInstanceState) {
33.             super.onCreate(savedInstanceState);
34.             setContentView(R.layout.activity_principal);
35.
36.             ckbCi = (CheckBox)findViewById(R.id.ckbCi);
37.             ckbFu = (CheckBox)findViewById(R.id.ckbFu);
38.             ckbGa = (CheckBox)findViewById(R.id.ckbGa);
39.             ckbIn = (CheckBox)findViewById(R.id.ckbIn);
40.             ckbLi = (CheckBox)findViewById(R.id.ckbLi);
41.             ckbTe = (CheckBox)findViewById(R.id.ckbTe);
42.             txtSel = (TextView)findViewById(R.id.txtSel);
43.             btnEx = (Button)findViewById(R.id.btnEx);
44.             btnDes = (Button)findViewById(R.id.btnDes);
45.             ckbCi.setOnCheckedChangeListener(this);
46.             ckbFu.setOnCheckedChangeListener(this);
47.             ckbGa.setOnCheckedChangeListener(this);
48.             ckbIn.setOnCheckedChangeListener(this);
49.             ckbLi.setOnCheckedChangeListener(this);
50.             ckbTe.setOnCheckedChangeListener(this);
51.             btnEx.setOnClickListener(this);
52.             btnDes.setOnClickListener(this);
53.
54.             cont = 0;
55.             exibirSelecionados();
56.     }
57.
58.     private void exibirSelecionados(){
59.             //Recuperar o texto contido no TextView
60.             String txt =
```

```
61.     getResources().getString(R.string.txt_selecionados);
62.     txtSel.setText(String.format("%s= %d", txt, cont));
63.     }
64.
65.     @Override
66.     public void onClick(View v) {
67.             switch(v.getId()){
68.             case R.id.btnEx:
69.             String txt = "Temas selecionados\n\n";
70.             txt += ckbCi.isChecked()?"Cinema\n":"";
71.             txt += ckbFu.isChecked()?"Futebol\n":"";
72.             txt += ckbGa.isChecked()?"Gastronomia\n":"";
73.             txt += ckbIn.isChecked()?"Informática\n":"";
74.             txt += ckbLi.isChecked()?"Literatura\n":"";
75.             txt += ckbTe.isChecked()?"Teatro\n":"";
76.
77.             AlertDialog.Builder dlg = new
78.     AlertDialog.Builder(this);
79.             dlg.setMessage(txt);
80.             dlg.setPositiveButton("OK",null);
81.             dlg.show();
82.             break;
83.             case R.id.btnDes:
84.             ckbCi.setChecked(false);
85.             ckbFu.setChecked(false);
86.             ckbGa.setChecked(false);
87.             ckbIn.setChecked(false);
88.             ckbLi.setChecked(false);
89.             ckbTe.setChecked(false);
90.             break;
91.             }
92.     }
93.
94.     @Override
95.     public void onCheckedChanged(CompoundButton
96.     buttonView, boolean isChecked) {
```

```
97.             if (isChecked == true){
98.                     cont++;
99.             }else{
100.                    cont--;
101.            }
102.            exibirSelecionados();
103.    }
104.    }
```

O resultado da execução do aplicativo no emulador é ilustrado na figura 3.1. No resultado é possível notar a seleção dos elementos no controle *CheckBox*.

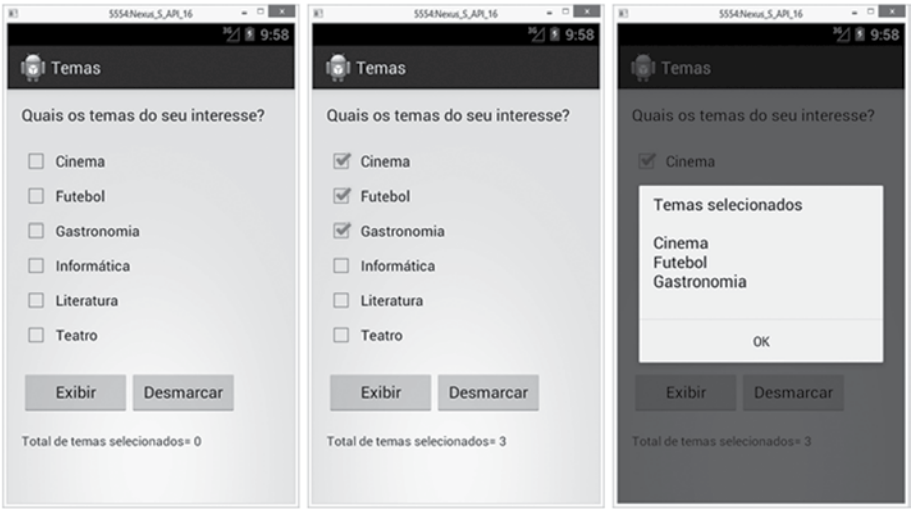

Figura 3.1 – Resultado da Execução do Aplicativo no Emulador.

CONCEITO

A classe *RadioGroup* é responsável por definir o comportamento mutuamente exclusivo nos controles *RadioButton*. Assim, é possível criar um mecanismo em que apenas um elemento (*RadioButton*) é selecionado de cada vez. Para retornar o elemento selecionado é necessário utilizar o método *getCheckecRadioButtonId()*

O controle Radio Button permite a seleção única de elementos a partir de um conjunto. Para demonstrar a utilização deste controle será criada um novo aplicativo nomeado **App04**. A atividade principal do projeto foi nomeada como *Principal*. A listagem Código 4 define as constantes textuais utilizadas no exemplo, as quais devem ser especificadas no arquivo *strings.xml*. A listagem Código 5 apresenta a interface gráfica da aplicação, em que os controles são definidos no arquivo de layout com a linguagem XML e que foi nomeado *activity_principal.xml*.

Código 4

```
1.      <?xml version=”1.0” encoding=”utf-8”?>
2.      <resources>
3.
4.          <string name=”app_name”>Quiz</string>
5.          <string name=”txt_questao”>Quem descobriu o
6.      Brasil?</string>
7.          <string name=”rdb_a”>Américo Vespúcio</string>
8.          <string name=”rdb_b”>Cristóvão Colombo</string>
9.          <string name=”rdb_c”>Dom Pedro II</string>
10.         <string name=”rdb_d”>Marco Polo</string>
11.         <string name=”rdb_e”>Pedro Álvares Cabral</string>
12.         <string name=”btn_ok”>OK</string>
13.         <string
14.     name=”action_settings”>Configurações</string>
15.
16.     </resources>
```

Código 5

```
1.      <LinearLayout
2.      xmlns:android=”http://schemas.android.com/apk/res/and
3.      roid”
4.          xmlns:tools=”http://schemas.android.com/tools”
5.      android:layout_width=”match_parent”
6.          android:layout_height=”match_parent”
7.      android:paddingLeft=”@dimen/activity_horizontal_margi
8.      n”
```

```
9.
10.     android:paddingRight=”@dimen/activity_horizontal_marg
11.     in”
12.
13.     android:paddingTop=”@dimen/activity_vertical_margin”
14.
15.     android:paddingBottom=”@dimen/activity_vertical_margi
16.     n” tools:context=”.Principal”
17.         android:orientation=”vertical”>
18.
19.         <TextView
20.             android:text=”@string/txt_questao”
21.             android:layout_width=”match_parent”
22.             android:layout_height=”wrap_content”
23.             android:layout_marginBottom=”20sp”
24.             android:textSize=”24sp”
25.             android:textColor=”#0000FF”/>
26.
27.         <RadioGroup
28.             android:id=”@+id/rdgAlter”
29.             android:layout_width=”match_parent”
30.             android:layout_height=”wrap_content”
31.             android:orientation=”vertical”>
32.
33.             <RadioButton
34.                 android:id=”@+id/rdbA”
35.                 android:layout_width=”match_parent”
36.                 android:layout_height=”wrap_content”
37.                 android:text=”@string/rdb_a”
38.                 android:textSize=”22sp”
39.                 android:checked=”true”/>
40.             <RadioButton
41.                 android:id=”@+id/rdbB”
```

```
42.                 android:layout_width="match_parent"
43.                 android:layout_height="wrap_content"
44.                 android:text="@string/rdb_b"
45.                 android:textSize="22sp"/>
46.             <RadioButton
47.                 android:id="@+id/rdbC"
48.                 android:layout_width="match_parent"
49.                 android:layout_height="wrap_content"
50.                 android:text="@string/rdb_c"
51.                 android:textSize="22sp"/>
52.             <RadioButton
53.                 android:id="@+id/rdbD"
54.                 android:layout_width="match_parent"
55.                 android:layout_height="wrap_content"
56.                 android:text="@string/rdb_d"
57.                 android:textSize="22sp"/>
58.             <RadioButton
59.                 android:id="@+id/rdbE"
60.                 android:layout_width="match_parent"
61.                 android:layout_height="wrap_content"
62.                 android:text="@string/rdb_e"
63.                 android:textSize="22sp"/>
64.
65.         </RadioGroup>
66.
67.
68.         <Button
69.             android:id="@+id/btnOK"
70.             android:layout_width="match_parent"
71.             android:layout_height="wrap_content"
72.             android:text="@string/btn_ok"
73.             android:layout_marginTop="20sp"
74.             android:textSize="22sp" />
75.
76.     </LinearLayout>
```

Na listagem Código 6 é demonstrado o tratamento de eventos originários do controle *Radio Button*. Neste exemplo, é possível notar o uso da interface *RadioGroup.OnCheckedChangeListener*, que implementa o método *onChecked-Changed*, o qual é responsável por identificar o controle responsável pelo disparo do evento.

Código 6

```
1.      package br.android.app04;
2.
3.      import android.app.Activity;
4.      import android.app.AlertDialog;
5.      import android.os.Bundle;
6.      import android.view.Menu;
7.      import android.view.MenuItem;
8.      import android.view.View;
9.      import android.widget.Button;
10.     import android.widget.CompoundButton;
11.     import android.widget.RadioButton;
12.     import android.widget.RadioGroup;
13.     import android.widget.Toast;
14.
15.
16.     public class Principal extends Activity implements
17.     View.OnClickListener,
18.     RadioGroup.OnCheckedChangeListener {
19.
20.         private RadioGroup rdgAlter;
21.         private Button btnOK;
22.         private final String CORRETA=”E”;
23.
24.         @Override
25.         protected void onCreate(Bundle
26.     savedInstanceState) {
27.             super.onCreate(savedInstanceState);
28.             setContentView(R.layout.activity_principal);
29.
```

```
30.             rdgAlter =
31.     (RadioGroup)findViewById(R.id.rdgAlter);
32.             btnOK = (Button)findViewById(R.id.btnOK);
33.
34.             rdgAlter.setOnCheckedChangeListener(this);
35.             btnOK.setOnClickListener(this);
36.         }
37.
38.         @Override
39.         public void onClick(View v) {
40.             String txt = “”;
41.             if (rdgAlter.getCheckedRadioButtonId() ==
42.     R.id.rdbE){
43.                 txt = “Parabéns, alternativa correta!”;
44.             }else{
45.                 txt = “Você não acertou, tente
46.     novamente.”;
47.             }
48.
49.             AlertDialog.Builder dlg = new
50.     AlertDialog.Builder(this);
51.             dlg.setTitle(R.string.app_name);
52.             dlg.setMessage(txt);
53.             dlg.setPositiveButton(“OK”,null);
54.             dlg.show();
55.         }
56.
57.         @Override
58.         public void onCheckedChanged(RadioGroup group,
59.     int checkedId) {
60.             //Retornar o TEXTO do RadioButton selecionado
61.             RadioButton rdb =
62.     (RadioButton)findViewById(group.getCheckedRadioButton
63.     Id());
64.
65.     Toast.makeText(this,rdb.getText(),Toast.LENGTH_SHORT)
```

```
66.     .show();
67.         }
68.     }
```

A figura 3.2 apresenta o resultado da execução do aplicativo no emulador. O controle Radio Button permite apenas a seleção de um único elemento do conjunto.

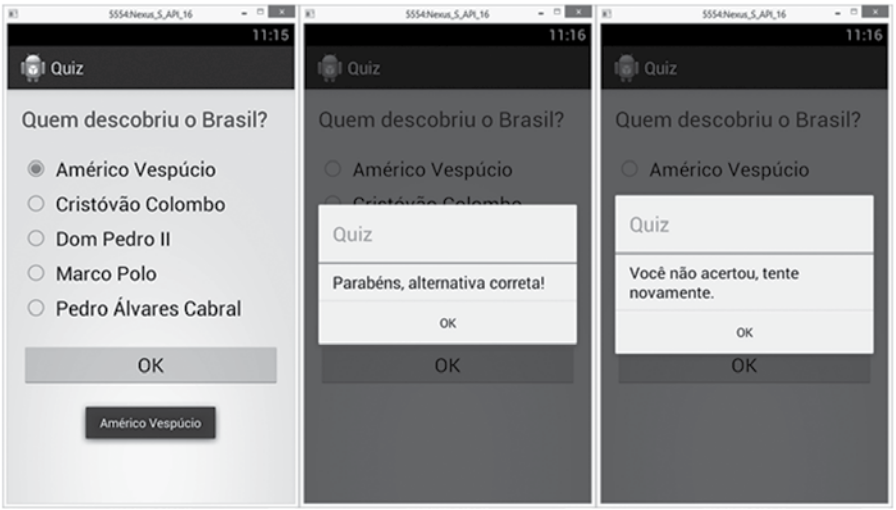

Figura 3.2 – Resultado da Execução do Aplicativo no Emulador.

Controle para Exibição de Conjuntos

Na plataforma Android, o controle *Spinner* é utilizado para a apresentação de um conjunto de elementos por meio de um menu suspenso. Este controle também é conhecido como *DropDown* (ou *ComboBox*), e permite que o usuário selecione um item a partir da exibição de um conjunto. Os elementos que fazem parte do *Spinner* podem ser definidos de maneira estática, no arquivo de valores *strings.xml*, ou podem ser obtidos dinamicamente, por meio de uma consulta a uma base de dados. A interface *AdapterView.OnItemSelectedListener* é utilizada para determinar qual elemento do conjunto foi selecionado. Assim, a cada interação do usuário é possível identificar e recuperar o elemento escolhido (SPINNER, 2014).

CONEXÃO

No desenvolvimento de uma aplicativo para dispositivo móvel é possível incluir o recurso de clicar com o botão direito do mouse, muito utilizado em aplicações para desktop. Em um aplicativo Android esta funcionalidade é acessada com o evento *OnLongClick*, quando o usuário toca e segura a tela touchscreen por mais de um segundo. Com isso, é possível, por exemplo, apresentar um menu suspenso para realização de determinadas tarefas. Você pode encontrar mais detalhes sobre o uso do *OnLongClick* em: <https://developer.android.com/eference/android/view/View.OnLongClickListener.html> .

Para demonstrar o funcionamento do controle *Spinner*, vamos criar um novo aplicativo no ambiente Android Studio, o qual deverá ser nomeado **App05**. A listagem Código 7 apresenta os valores definidos no arquivo strings.xml. Neste arquivo é importante notar a especificação do arranjo estático lista_nomes, o qual será utilizada para preencher o controle Spinner.

Código 7

```
1.      <?xml version=”1.0” encoding=”utf-8”?>
2.      <resources>
3.
4.          <string name=”app_name”>Cadastro</string>
5.          <string name=”txt_cadastro”>Cadastro de
6.      Pessoas</string>
7.          <string name=”txt_nome”>Nome</string>
8.          <string name=”txt_cidade”>Cidade</string>
9.          <string name=”btn_cadastrar”>Cadastrar</string>
10.         <string
11.     name=”action_settings”>Configurações</string>
12.
13.         <string-array name=”lista_nomes”>
14.             <item>Alexander Tanner</item>
15.             <item>Marsden Kirkland</item>
16.             <item>Brenden Swanson</item>
17.             <item>Caleb Odom</item>
```

```
18.             <item>Plato Baldwin</item>
19.             <item>Cadman Becker</item>
20.             <item>Byron Sharpe</item>
21.             <item>Nathaniel Bass</item>
22.             <item>Uriah Pate</item>
23.             <item>Wallace Day</item>
24.             <item>Elmo Woodard</item>
25.             <item>Lucius Evans</item>
26.             <item>Abdul Sexton</item>
27.             <item>Kelly Beach</item>
28.             <item>Richard Hardy</item>
29.             <item>Driscoll Adkins</item>
30.             <item>Tarik West</item>
31.             <item>Orson House</item>
32.             <item>Aaron Sparks</item>
33.         </string-array>
34.
35.     </resources>
```

Na listagem Código 8 são demonstrados os controles utilizados para definição da interface gráfica do aplicativo. Neste exemplo, dois menus suspensos foram utilizados para apresentação de conjuntos, em que os dados serão carregados programaticamente. Outro detalhe importante que merece destaque é o uso da propriedade *spinnerMode*, a qual modifica a maneira que os elementos do conjunto são apresentados. A definição *spinnerMode="dialog"* faz com que os elementos sejam exibidos em uma janela específica, a qual pode ocupar toda a tela do dispositivo dependendo da quantidade de elementos. Por outro lado, *spinnerMode="dropdown"* exibe os elementos no mesmo contexto dos demais controles que fazem parte da interface.

Código 8

```
1.      <LinearLayout
2.      xmlns:android="http://schemas.android.com/apk/res/and
3.      roid"
4.          xmlns:tools="http://schemas.android.com/tools"
5.      android:layout_width="match_parent"
```

```
6.          android:layout_height=”match_parent”
7.      android:paddingLeft=”@dimen/activity_horizontal_margi
8.      n”
9.
10.     android:paddingRight=”@dimen/activity_horizontal_marg
11.     in”
12.
13.     android:paddingTop=”@dimen/activity_vertical_margin”
14.
15.     android:paddingBottom=”@dimen/activity_vertical_margi]
16.     n” tools:context=”.Principal”
17.         android:orientation=”vertical”>
18.
19.         <TextView
20.             android:text=”@string/txt_cadastro”
21.             android:layout_width=”match_parent”
22.             android:layout_height=”wrap_content”
23.             android:textSize=”24sp”
24.             android:textStyle=”bold” />
25.
26.         <TextView
27.             android:text=”@string/txt_nome”
28.             android:layout_width=”match_parent”
29.             android:layout_height=”wrap_content”
30.             android:layout_marginTop=”20sp”
31.             android:textSize=”24sp”/>
32.
33.         <Spinner
34.             android:id=”@+id/spnNomes”
35.             android:layout_width=”match_parent”
36.             android:layout_height=”wrap_content”
37.             android:spinnerMode=”dialog”
38.             android:textSize=”24sp” />
39.
40.         <TextView
41.             android:text=”@string/txt_cidade”
```

```
42.             android:layout_width="match_parent"
43.             android:layout_height="wrap_content"
44.             android:layout_marginTop="20sp"
45.             android:textSize="24sp"/>
46.
47.         <Spinner
48.             android:id="@+id/spnCidades"
49.             android:layout_width="match_parent"
50.             android:layout_height="wrap_content"
51.             android:spinnerMode="dropdown"
52.             android:textSize="24sp" />
53.
54.         <Button
55.             android:id="@+id/btnCadastrar"
56.             android:layout_width="match_parent"
57.             android:layout_height="wrap_content"
58.             android:text="@string/btn_cadastrar"
59.             android:layout_marginTop="20sp"
60.             android:textSize="24sp"/>
61.
62.
63.
64.     </LinearLayout>
```

A listagem Código 9 apresenta a lógica de programação do aplicativo. Neste arquivo é possível notar os métodos para preenchimento dos controles *Spinner.* O método *carregarListaNomes* é responsável por preencher o *Spinner spnNomes* com os dados originários do arranjo estático definido no arquivo *strings.xml*. Neste método a chamada *createFromResource* realiza o preenchimento do controle, enquanto o argumento *android.R.layout.simple_spinner_item*, especifica como os elementos serão exibidos. O preenchimento do *Spinner spCidades,* a partir da lista dinâmica (*ArrayList*) denominada listaCidades, é realizado no método *carregarListaCidades*.

Código 9

```
1.      package br.android.app05;
2.
3.      import android.app.Activity;
4.      import android.app.AlertDialog;
5.      import android.os.Bundle;
6.      import android.view.Menu;
7.      import android.view.MenuItem;
8.      import android.view.View;
9.      import android.widget.AdapterView;
10.     import android.widget.ArrayAdapter;
11.     import android.widget.Button;
12.     import android.widget.Spinner;
13.     import android.widget.Toast;
14.
15.     import java.util.ArrayList;
16.     import java.util.List;
17.
18.     public class Principal extends Activity implements
19.     View.OnClickListener,
20.     AdapterView.OnItemSelectedListener{
21.
22.         private Spinner spnNomes;
23.         private Spinner spnCidades;
24.         private Button btnCadastrar;
25.         private List<String> listaCidadades;
26.
27.
28.         @Override
29.         protected void onCreate(Bundle
30.     savedInstanceState) {
31.             super.onCreate(savedInstanceState);
32.             setContentView(R.layout.activity_principal);
33.
34.       spnNomes = (Spinner) findViewById(R.id.spnNomes);
35.     spnCidades = (Spinner) findViewById(R.id.spnCidades);
```

```
36.      btnCadastrar = (Button)
37.    findViewById(R.id.btnCadastrar);
38.
39.            spnNomes.setOnItemSelectedListener(this);
40.            spnCidades.setOnItemSelectedListener(this);
41.            btnCadastrar.setOnClickListener(this);
42.
43.            carregarListaNomes();
44.            carregarListaCidades();
45.        }
46.
47.
48.         // CARREGAR lista de nomes do RESOURCE
49.         private void carregarListaNomes(){
50.            ArrayAdapter<CharSequence> adp =
51.    ArrayAdapter.createFromResource(
52.                    this,R.array.lista_nomes,
53.                   android.R.layout.simple_spinner_item);
54.
55.    adp.setDropDownViewResource(android.R.layout.simple_s
56.    pinner_dropdown_item);
57.            spnNomes.setAdapter(adp);
58.        }
59.        // CARREGAR lista de cidades do ARRAYLIST
60.         private void carregarListaCidades(){
61.            listaCidadades = new ArrayList<String>();
62.            listaCidadades.add(“Americana”);
63.            listaCidadades.add(“Araraquara”);
64.            listaCidadades.add(“Batatais”);
65.            listaCidadades.add(“Campinas”);
66.            listaCidadades.add(“Limeira”);
67.            listaCidadades.add(“Ribeirão Preto”);
68.            listaCidadades.add(“São Paulo”);
69.
70.            ArrayAdapter<String> adp = new
71.    ArrayAdapter<String>(this,
72.                    android.R.layout.simple_spinner_item,
```

```
73.     listaCidadades);
74.
75.
76.     adp.setDropDownViewResource(android.R.layout.simple_s
77.     pinner_dropdown_item);
78.             spnCidades.setAdapter(adp);
79.         }
80.
81.         @Override
82.         public void onClick(View v) {
83.             if(v.getId() == R.id.btnCadastrar){
84.                 AlertDialog.Builder dlg = new
85.     AlertDialog.Builder(this);
86.                 dlg.setTitle(R.string.app_name);
87.                 dlg.setMessage("");
88.                 dlg.setPositiveButton("OK",null);
89.                 dlg.show();
90.             }
91.         }
92.
93.         @Override
94.         public void onItemSelected(AdapterView<?> parent,
95.     View view, int position, long id) {
96.             //Exibir item selecionado
97.             String item =
98.     parent.getItemAtPosition(position).toString();
99.             Toast.makeText(this,item,
100.    Toast.LENGTH_SHORT).show();
101.        }
102.
103.        @Override
104.        public void onNothingSelected(AdapterView<?>
105.    parent) {
106.
107.        }
108.    }
109.
```

Na figura 3.3 são exibidos o resultado da execução do aplicativo no emulador. Na figura é possível notar a apresentação dos conjuntos de dados com o controle Spinner.

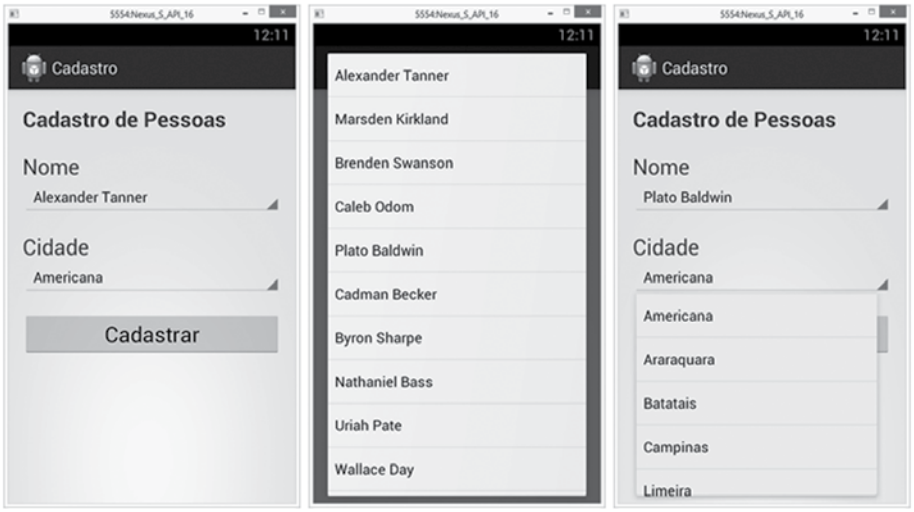

Figura 3.3 – Resultado da Execução do Aplicativo no Emulador.

Com isso, finalizamos o estudo a respeito dos controles para seleção e exibição de conjuntos.

ATIVIDADES

Com base no conteúdo apresentado neste capítulo, responda as seguintes questões objetivas:

01. Qual controle da plataforma Android permite a seleção mutuamente exclusiva de elementos?

a) Button
b) TextView
c) EditText
d) CheckBox
e) RadioButton

02. Qual interface deve ser implementada na classe da Atividade para permitir o tratamento de eventos originários do controle Spinner?

a) OnChecked
b) OnItemSelected
c) OnItemChecked
d) OnClick
e) OnSelected

03. Complemente a sentença: O controle _____ é utilizado para apresentação de um conjunto de elementos por meio de um menu suspenso.

a) CheckBox
b) EditText
c) RadioButton
d) Spinner
e) TextView

REFLEXÃO

Os controles para seleção de conjunto de elementos, como por exemplo o Spinner, permitem apresentar sequencias extensas de informações nos dispositivos móveis. Com as evoluções tecnológicas de *hardware* e *software* esta tarefa tem sido facilmente executada, porém, limitações relativas ao tamanho da tela podem dificultar a exibição de todos os elementos. Neste contexto, realize uma reflexão sobre a seguinte questão: Qual estratégia pode ser utilizada na construção de uma interface gráfica que necessita exibir uma quantidade grande de elementos? Imagine, por exemplo, que está interface precisa exibir uma listagem de 100.000 nomes de pessoas. Como você resolveria esta tarefa?

LEITURA

Neste momento, você já conseguiu um nível intermediário de aprendizado no desenvolvimento de aplicações para Android. Para aprimorar seus conhecimentos é sugerida a leitura da obra *Desenvolvimento de Aplicações Profissionais para Android*", do autor Ricardo Queirós, publicado em 2014 pela editora FCA.

REFERÊNCIAS BIBLIOGRÁFICAS

CHECKBOXES. Android Developer Guide: CheckBoxes. Disponível em <http://developer.android.com/guide/topics/ui/controls/checkbox.html>, acesso em novembro de 2014.

SPINNER. Android Developer Guide: Spinner. Disponível em <http://developer.android.com/guide/topics/ui/controls/spinner.html>, acesso em novembro de 2014.

RADIO BUTTONS. Android Developer Guide: Radio Buttons. Disponível em <http://developer.android.com/guide/topics/ui/controls/radiobutton.html>, acesso em novembro de 2014.

4

Construção de Aplicativos com Múltiplas Atividades

Construção de Aplicativos com Múltiplas Atividades

Neste capítulo você aprenderá sobre como desenvolver aplicativos para plataforma Android com várias atividades. Com isso, é possível construir aplicativos formados por várias telas, e ainda realizar a troca de dados entre as atividades. O entendimento desta metodologia é fundamental para os programadores de dispositivos móveis, uma vez, que quase a totalidade dos aplicativos existentes são confeccionados com múltiplas telas.

OBJETIVOS

Ao final deste capítulo você será capaz de:

- Desenvolver aplicativos para Android com múltiplas Atividades;
- Utilizar os recursos da classe Intent para chamar outras Atividades;
- Realizar a passagem de parâmetros entre Atividades;

Aplicativos com Múltiplas Atividades

Na plataforma Android o termo Atividade (*Activity*) é utilizado para designar uma tela do aplicativo. Na atividade é possível incluir elemento visuais, denominados controles, por meio dos quais o usuário realizará a interação. Dependendo da complexidade do aplicativo que será desenvolvido, será necessário utilizar um número significativo de controles para interação. Deste modo, a tela que será exibida para o usuário poderá ficar poluída em função da grande quantidade controles.

Uma das abordagens que pode ser utilizada para minimizar problemas no processo de construção de *interfaces* gráficas é confeccionar aplicativos com múltiplas telas. Com isso, é possível dividir o processo de interação entre o usuário e o aplicativo em múltiplas telas. Além disso, os controles de interação serão distribuídos nas diversas atividades que compõe o aplicativo.

Para construir aplicativos com múltiplas telas na plataforma Android, o primeiro passo é criar um novo projeto no ambiente de desenvolvimento. No ambiente Android Studio, ao final desse processo, o novo projeto criado é composto por uma atividade, a qual é denomina atividade principal. Conforme descrito

anteriormente, para cada atividade incluída no projeto dois arquivos são criados: um arquivo relativo a lógica de programação, o qual é representada por uma classe na linguagem de programação Java; e um arquivo em que o *layout* da interface gráfica é definido, em que os controles são especificados com a linguagem de marcação XML.

COMENTÁRIO

Uma alternativa interessante para construção de interface gráficas com múltiplas telas é utilizar um recurso da plataforma Android conhecido como Fragmentos (*Fragment*). Com isso, é possível desenvolver aplicações com uma única Atividade, em que são definidos fragmentos que possibilitam modificações partes da interface. Este recurso alterar porções do l*ayout* da Atividade, modificando, por exemplo, os controlesque são apresentados.

Quando uma nova atividade é acionada no projeto, outros dois arquivos serão criados, ou seja, um arquivo para lógica de programação e outro para definição do layout. Assim, para toda e qualquer atividade incluída em um aplicativo Android, dois arquivos precisam ser criados, tanto para lógica (Java), quanto para interface (XML).

No ambiente de desenvolvimento Android Studio, as etapas para adicionar uma nova Atividade ao projeto é exemplificada na figura 4.1. Descritivamente, esta tarefa é realizada acionando o menu *File* > *New ...* > *Activity* > *Blank Activity*. Em seguida é necessário especificar o nome da nova Atividade.

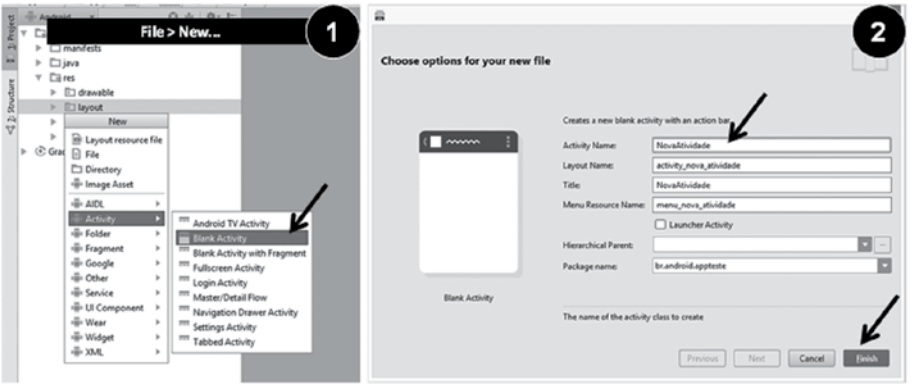

Figura 4.1 – Etapas para Criação de uma Nova Atividade ao Projeto.

Na figura 4.2 é possível visualizar o resultado da criação de uma nova atividade, em que dois arquivos foram criados:

- ***NovaAtividade.java*** – relativo a lógica de programação;
- ***activity_nova_atividade.xml*** – define o layout da interface gráfica com o usuário.

Ainda resultante do processo de criação de uma nova Atividade, alterações são realizadas no arquivo de manifesto do projeto. Estas modificações são necessárias para possibilitar que a nova atividade seja referenciada ao longo da programação do projeto. A listagem Código 1 demonstra as alterações do arquivo *AndroidManifest.xml* após a inserção de uma nova atividade ao projeto.

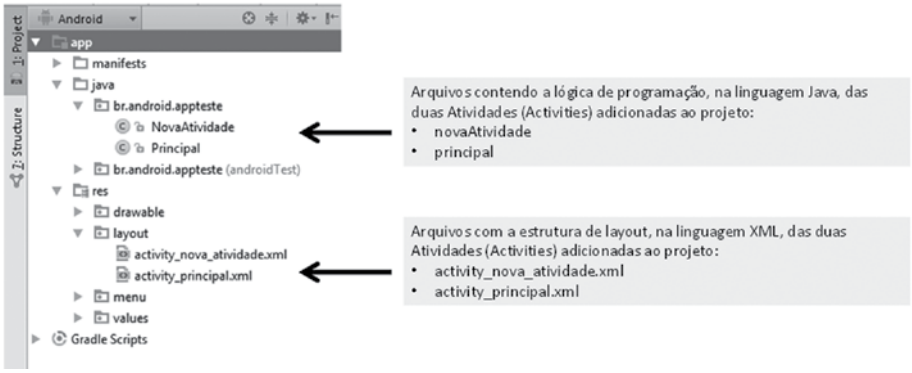

Figura 4.2 – Arquivos de lógica de programação e layout criados para a nova atividade.

Código 1

```
1.      <?xml version="1.0" encoding="utf-8"?>
2.      <manifest  xmlns:android="http://schemas.android.com/apk/res/
android"
3.          package="br.android.appteste" >
4.
5.          <application
6.              android:allowBackup="true"
7.              android:icon="@drawable/ic_launcher"
8.              android:label="@string/app_name"
9.              android:theme="@style/AppTheme" >
10.             <activity
11.                 android:name="br.android.appteste.Principal"
```

```
12.                android:label="@string/app_name" >
13.                <intent-filter>
14.                     <action android:name="android.intent.action.
MAIN" />
15.
16.                     <category android:name="android.intent.cate-
gory.LAUNCHER" />
17.                </intent-filter>
18.           </activity>
19.           <activity
20.                android:name="br.android.appteste.NovaAtividade"
21.                 android:label="@string/title_activity_nova_ativi-
dade" >
22.           </activity>
23.       </application>
24.
25.    </manifest>
```

CONCEITO

O arquivo de manifesto pode ser utilizado para especificar diversas propriedades das atividades que fazem parte do aplicativo. Um propriedade interessante, por exemplo, é denominada *android:screenOrientation* que permite indicar qual a orientação da atividade, entre paisagem (l*andscape*), retrato (*portrait*), entre outras. Com isso, é possível impedir que o aplicativo mude de orientação quando o usuário rotacionar o dispositivo.

Troca de Dados entre Atividades

O primeiro passo para desenvolver aplicativos que possibilitem a troca de dados entre Atividades, é necessário adicionar pelo menos duas atividades no projeto. Para exemplificar, considere um projeto contento as seguintes atividades: *Atividade1* – que representa a atividade principal do projeto, a qual é iniciada automaticamente quando o aplicativo é executado; e *Atividade2* – a qual é iniciada a partir de um botão na *Atividade1*, ou seja, a *Atividade2* é aberta quando o usuário acionar um botão na *Atividade1*.

Para realizar a ligação entre as duas atividades no projeto é preciso utilizar uma classe da plataforma Android conhecida como *Intent*, do pacote *android.content.intent.* Com um objeto do tipo Intent é possível fazer a conexão entre duas atividades, além disso, também existe a capacidade de trocar dados entre estas atividades. A criação de um objeto Intent, recebe no construtor dois parâmetros: o contexto da atividade; qual a atividade que será chamada (INTENT, 2014). Dessa forma, para instanciação do objeto a seguinte codificação é necessária:

```
Intent it = new Intent(this,Atividade2.class)
```

Veja na codificação que a *Atividade2*.class representa a atividade que será aberta quando o usuário interagir com o botão da *Atividade1*. O próximo passo, é especificar os parâmetros (ou dados) que serão enviados para a atividade que foi chamada. Neste caso, é preciso utilizar o método *putExtra* da classe *Intent*. Para este método são passados dois parâmetros, do tipo <*chave,valor*>. Em que a chave representa o identificador do parâmetro que será passado, enquanto, o valor define o conteúdo. Por exemplo,

```
it.putExtra("nome","João da Silva");
it.putExtra("idade", 23);
it.putExtra("altura", 1.85);
```

Conforme demonstrado, três informações serão passadas para a segunda atividade, as quais foram identificadas como *nome, idade e altura*, e receberam respectivamente os valores, "*João da Silva", 23 e 1.85*. A instrução que exibe na tela a nova atividade é denominada *startActivity*, da seguinte maneira:

```
startActivity(it);
```

Na codificação da *Atividade2* é desejável recuperar as informações que foram enviadas pela *Atividade1*. Para isso, é necessário utilizar a chamada do método getIntent, o qual retornará todas as informações recebidas.

```
Intent it = getIntent();
```

Finalmente, para extrair cada valor passado como parâmetro é utilizado o método get, seguido do tipo de dados que foi enviado, por exemplo:

```
String nome = it.getStringExtra("nome");
int idade = it.getIntExtra("idade",);
double altura = it.getDoubleExtra("altura",0);
```

Note na codificação que para os campos numéricos um segundo valor é especificado na chamada do método *get*. Este parâmetro representa o valor padrão, assim, caso algum problema ocorra durante a recuperação do parâmetro, o valor padrão é atribuído a variável.

CONEXÃO

Você pode encontrar mais informações a respeito da classe Intent na documentação oficial da plataforma Android. Na referência é apresentada outras utilizações da classes como, por exemplo, na inicialização de serviços no dispositivo. Mais detalhes em: <https://developer.android.com/reference/android/content/Intent.html>.

Exemplo: Iniciando uma nova Atividade com passagem de parâmetros

Para demonstrar o desenvolvimento de aplicativos com múltiplas atividades na plataforma Android, vamos criar um novo projeto denominado App06. Este projeto será composto por duas atividades: uma atividade principal, nomeada Principal; e por uma atividade secundária, a qual deve ser denominada Pedido. O conteúdo do arquivo de manifesto, em que as atividades são identificadas, é apresentado na listagem Código 2.

Código 2

```
1.      <?xml version="1.0" encoding="utf-8"?>
2.      <manifest
3.      xmlns:android="http://schemas.android.com/apk/res/andr
4.      oid"
5.          package="br.android.app06" >
6.          <application
7.              android:allowBackup="true"
8.              android:icon="@drawable/ic_launcher"
9.              android:label="@string/app_name"
10.             android:theme="@style/AppTheme" >
11.             <activity
12.                 android:name="br.android.app06.Principal"
13.                 android:label="@string/app_name" >
```

```
14.                 <intent-filter>
15.                     <action
16.     android:name="android.intent.action.MAIN" />
17.                     <category
18.     android:name="android.intent.category.LAUNCHER" />
19.                 </intent-filter>
20.             </activity>
21.             <activity
22.                 android:name="br.android.app06.Pedido"
23.                 android:label="@string/ app_name " >
24.             </activity>
25.         </application>
26.     </manifest>
```

A listagem Código 3 demonstra todas as constantes textuais que foram utilizadas no projeto, as quais são especificadas no arquivostrings.xml.

Código 3

```
1.      <?xml version="1.0" encoding="utf-8"?>
2.      <resources>
3.          <string name="app_name">Sorveteria do
4.      João</string>
5.          <string
6.      name="action_settings">Configurações</string>
7.
8.          <string name="txt_tipo">Tipo</string>
9.          <string name="txt_sabor">Sabor</string>
10.         <string name="txt_quantidade">Quantidade</string>
11.         <string name="txt_pedido">Pedido</string>
12.
13.         <string name="ckb_chocolate">Chocolate</string>
14.         <string name="ckb_creme">Creme</string>
15.         <string name="ckb_flocos">Flocos</string>
16.         <string name="ckb_morango">Morango</string>
17.         <string name="ckb_napolitano">Napolitano</string>
18.
```

```
19.         <string name=”btn_finalizar”>Finalizar</string>
20.         <string name=”btn_voltar”>Voltar</string>
21.     </resources>
```

O *layout* da tela principal do projeto, codificada no *arquivoactivity_principal.xml*, é apresentada na listagem Código 4.

Código 4

```
1.      <LinearLayout
2.      xmlns:android=”http://schemas.android.com/apk/res/andr
3.      oid”
4.          xmlns:tools=”http://schemas.android.com/tools”
5.      android:layout_width=”match_parent”
6.          android:layout_height=”match_parent”
7.      android:background=”#FFFFFF”
8.          android:orientation=”vertical”
9.
10.     android:paddingBottom=”@dimen/activity_vertical_margin
11.     ”
12.
13.     android:paddingLeft=”@dimen/activity_horizontal_margin
14.     ”
15.
16.     android:paddingRight=”@dimen/activity_horizontal_margi
17.     n”
18.
19.     android:paddingTop=”@dimen/activity_vertical_margin”
20.         tools:context=”.Principal”
21.         >
22.
23.         <TextView
24.             android:layout_width=”match_parent”
25.             android:layout_height=”wrap_content”
26.             android:text=”@string/txt_tipo”
27.             android:textSize=”20sp”
28.             android:textStyle=”bold”/>
```

```
29.
30.         <Spinner
31.             android:id=”@+id/spTipo”
32.             android:layout_width=”match_parent”
33.             android:layout_height=”wrap_content”
34.             android:textColor=”#000000”/>
35.
36.         <TextView
37.             android:layout_width=”match_parent”
38.             android:layout_height=”wrap_content”
39.             android:layout_marginTop=”5sp”
40.             android:text=”@string/txt_sabor”
41.             android:textSize=”20sp” />
42.
43.         <RadioGroup
44.             android:id=”@+id/rdgSabor”
45.             android:layout_width=”match_parent”
46.             android:layout_height=”wrap_content”>
47.
48.             <RadioButton
49.                 android:layout_width=”wrap_content”
50.                 android:layout_height=”wrap_content”
51.                 android:text=”@string/ckb_chocolate”/>
52.             <RadioButton
53.                 android:layout_width=”wrap_content”
54.                 android:layout_height=”wrap_content”
55.                 android:text=”@string/ckb_creme”/>
56.             <RadioButton
57.                 android:layout_width=”wrap_content”
58.                 android:layout_height=”wrap_content”
59.                 android:text=”@string/ckb_flocos”/>
60.             <RadioButton
61.                 android:layout_width=”wrap_content”
62.                 android:layout_height=”wrap_content”
63.                 android:text=”@string/ckb_morango”/>
64.             <RadioButton
```

```
65.                 android:layout_width="wrap_content"
66.                 android:layout_height="wrap_content"
67.                 android:text="@string/ckb_napolitano"/>
68.          </RadioGroup>
69.
70.
71.         <TextView
72.             android:layout_width="match_parent"
73.             android:layout_height="wrap_content"
74.             android:layout_marginTop="5sp"
75.             android:text="@string/txt_quantidade"
76.             android:textSize="20sp" />
77.
78.         <EditText
79.             android:id="@+id/edtQuantidade"
80.             android:layout_width="match_parent"
81.             android:layout_height="wrap_content"
82.             android:inputType="number"
83.             android:textSize="20sp"/>
84.
85.         <Button
86.             android:id="@+id/btnFinalizar"
87.             android:layout_width="match_parent"
88.             android:layout_height="wrap_content"
89.             android:layout_marginTop="5sp"
90.             android:width="150sp"
91.             android:text="@string/btn_finalizar"
92.             android:textSize="20sp"
93.              />
94.     </LinearLayout>
```

O conteúdo da lógica de programação da tela principal foi codificado na classe *Principal.java*. Neste arquivo você pode observar as instruções para chamada da atividade *Pedido*. Assim, neste momento, você já precisa ter adicionado a atividade ao projeto. A listagem Código 5 apresenta a codificação da classe *Principal.java*.

Código 5

```
1.      package br.android.app06;
2.
3.      import android.app.Activity;
4.      import android.app.AlertDialog;
5.      import android.content.DialogInterface;
6.      import android.content.Intent;
7.      import android.os.Bundle;
8.      import android.view.Menu;
9.      import android.view.MenuItem;
10.     import android.view.View;
11.     import android.widget.ArrayAdapter;
12.     import android.widget.Button;
13.     import android.widget.EditText;
14.     import android.widget.RadioButton;
15.     import android.widget.RadioGroup;
16.     import android.widget.Spinner;
17.
18.
19.     public class Principal extends Activity implements
20.     View.OnClickListener {
21.
22.         private Spinner spTipo;
23.         private RadioGroup rdgSabor;
24.         private EditText edtQuantidade;
25.         private Button btnFinalizar;
26.
27.         @Override
28.         protected void onCreate(Bundle savedInstanceState) {
29.             super.onCreate(savedInstanceState);
30.             setContentView(R.layout.activity_principal);
31.
32.
33.     spTipo = (Spinner)findViewById(R.id.spTipo);
34.             rdgSabor =
35.     RadioGroup)findViewById(R.id.rdgSabor);
36.             edtQuantidade =
```

```
37.     (EditText)findViewById(R.id.edtQuantidade);
38.             btnFinalizar =
39.     (Button)findViewById(R.id.btnFinalizar);
40.
41.             btnFinalizar.setOnClickListener(this);
42.             carregarListaTipos();
43.         }
44.
45.         private void carregarListaTipos(){
46.             String tipos[] = {“Cone”,”Cone
47.     Duplo”,”Picolé”,”Sundae”};
48.             ArrayAdapter<String> adp = new
49.     ArrayAdapter<String>(this,
50.                     android.R.layout.simple_spinner_item,
51.     tipos);
52.
53.
54.     adp.setDropDownViewResource(android.R.layout.simple_sp
55.     inner_dropdown_item);
56.             spTipo.setAdapter(adp);
57.         }
58.
59.
60.         @Override
61.         public void onClick(View v) {
62.             if(v.getId() == R.id.btnFinalizar){
63.
64.                 //Caixa de Diálogo
65.                 AlertDialog.Builder dlg = new
66.     AlertDialog.Builder(this);
67.                 dlg.setTitle(R.string.app_name);
68.                 dlg.setMessage(“Tem certeza que deseja
69.     finalizar o pedido?”);
70.                 dlg.setNegativeButton(“Sim”,new
71.     DialogInterface.OnClickListener() {
72.                     @Override
```

```
73.                     public void onClick(DialogInterface
74.    dialog, int which) {
75.                         abrirAtividade();
76.                     }
77.                 });
78.                 dlg.setPositiveButton(“Não”,null);
79.                 dlg.show();
80.             }
81.         }
82.
83.         private void abrirAtividade(){
84.
85.             //Definir Parâmetros
86.             String tipo =
87.    spTipo.getSelectedItem().toString();
88.
89.             int sabor_id =
90.    rdgSabor.getCheckedRadioButtonId();
91.             RadioButton rdb =
92.    RadioButton)findViewById(sabor_id);
93.             String sabor = rdb.getText().toString();
94.
95.             int qtde = 0;
96.       if (edtQuantidade.getText().length()> 0) {
97.        qtde =
98.    Integer.parseInt(edtQuantidade.getText().toString());
99.           }
100.
101.            //Criar Intent
102.            Intent it = new Intent(this,Pedido.class);
103.            it.putExtra(“tipo”,tipo);
104.            it.putExtra(“sabor”,sabor);
105.            it.putExtra(“quantidade”,qtde);
106.            startActivity(it);
107.        }
108.    }
109.
```

No Código 5 é importante observar nas linhas 103 até 107 a criação do objeto *Intent* para comunicação com a Atividade *Pedido*. Além disso, o método *putExtra* é utilizado para passagem de parâmetros.

O próximo passo é a codificação dos arquivos relativos a atividade secundária, a qual foi nomeada como *Pedido.* Para isso, inicialmente você deverá construir o *layout* da interface da atividade pedido, denominada *activity_pedido.xml*. O conteúdo do arquivo de marcação XML, referente a especificação dos controles que fazem parte da tela, é apresentado na listagem Código 6.

CONEXÃO

O método startActivity é utilizado para iniciar uma nova atividade na aplicação, no entanto, em muitos aplicativos após o fechamento da atividade desejamos retornar algum tipo de valor que foi manipulado. Para isso, é possível usar o método denominado startActivityForResult, o qual é detalhado em: <https://developer.android.com/reference/android/app/Activity.html>.

Código 6

```
1.      <LinearLayout
2.      xmlns:android=”http://schemas.android.com/apk/res/andr
3.      oid”
4.          xmlns:tools=”http://schemas.android.com/tools”
5.      android:layout_width=”match_parent”
6.          android:layout_height=”match_parent”
7.      android:paddingLeft=”@dimen/activity_horizontal_margin
8.      ”
9.
10.     android:paddingRight=”@dimen/activity_horizontal_margi
11.     n”
12.
13.     android:paddingTop=”@dimen/activity_vertical_margin”
14.
15.     android:paddingBottom=”@dimen/activity_vertical_margin
16.     ”
```

```
17.         tools:context=”br.android.app06.Pedido”
18.         android:orientation=”vertical”>
19.
20.         <TextView
21.             android:layout_width=”match_parent”
22.             android:layout_height=”wrap_content”
23.             android:text=”@string/txt_pedido”
24.             android:textSize=”20sp”
25.             android:textStyle=”bold”/>
26.
27.         <TextView
28.             android:id=”@+id/txtPedido”
29.             android:layout_width=”match_parent”
30.             android:layout_height=”wrap_content”
31.             android:maxLines=”10”
32.             android:singleLine=”false” />
33.
34.         <Button
35.             android:id=”@+id/btnVoltar”
36.             android:layout_width=”match_parent”
37.             android:layout_height=”wrap_content”
38.             android:layout_marginTop=”5sp”
39.             android:width=”150sp”
40.             android:text=”@string/btn_voltar”
41.             android:textSize=”20sp”
42.             />
43.
44.     </LinearLayout>
```

A listagem Código 7 apresenta a lógica de programação da atividade Pedido. Na codificação é importante notar o recebimento dos dados originários da atividade principal, além disso, um controle do tipo *TextView* foi empregado para exibição dos valores.

Código 7

```
1.      package br.android.app06;
2.
3.      import android.app.Activity;
```

```
4.      import android.content.Intent;
5.      import android.os.Bundle;
6.      import android.view.Menu;
7.      import android.view.MenuItem;
8.      import android.view.View;
9.      import android.widget.Button;
10.     import android.widget.EditText;
11.     import android.widget.TextView;
12.
13.
14.     public class Pedido extends Activity implements
15.     View.OnClickListener{
16.
17.         private Button btnVoltar;
18.         private TextView txtPedido;
19.
20.         @Override
21.         protected void onCreate(Bundle savedInstanceState)
22.     {
23.             super.onCreate(savedInstanceState);
24.             setContentView(R.layout.activity_pedido);
25.
26.             btnVoltar =
27.     Button)findViewById(R.id.btnVoltar);
28.             txtPedido =
29.     (TextView)findViewById(R.id.txtPedido);
30.             btnVoltar.setOnClickListener(this);
31.
32.             //RECEBER dados da Atividade Principal
33.             Intent it = getIntent();
34.             String tipo = it.getStringExtra(“tipo”);
35.             String sabor = it.getStringExtra(“sabor”);
36.             int quantidade = it.getIntExtra(“quantidade”,
37.     0);
38.
39.             double vltotal = 0;
```

```
40.             if (tipo.equals(“Cone”)){
41.                 vltotal = quantidade * 2.50;
42.             }else if (tipo.equals(“Cone Duplo”)) {
43.                 vltotal = quantidade * 4.50;
44.             }else if (tipo.equals(“Picolé”)) {
45.                 vltotal = quantidade * 1.50;
46.             }else{
47.                 vltotal = quantidade * 8.00;
48.             }
49.
50.             //Exibir o resultado
51.             StringBuilder sb = new StringBuilder();
52.             sb.append(“Tipo =
53.     “).append(tipo).append(“\n”);
54.             sb.append(“Sabor=
55.     “).append(sabor).append(“\n”);
56.             sb.append(“Qtde.=
57.     “).append(quantidade).append(“\n”);
58.           sb.append(“\n”).append(“\n”);
59.            sb.append(“Vl.Total=
60.     “).append(String.format(“%.2f”,vltotal)).append(“\n”);
61.             txtPedido.setText(sb.toString());
62.         }
63.
64.         @Override
65.         public void onClick(View v) {
66.             if(v.getId() == R.id.btnVoltar){
67.                 onBackPressed();
68.             }
69.         }
70.     }
```

No Código 7 o método *onBackPressed* foi empregado para possibilitar o retorna a atividade principal. Com isso, quando o usuário acionar o botão voltar, a atividade pedido será fechada e, a atividade principal será reexibida. A chamada do método o*nBackPressed* é equivalente ao pressionamento do botão voltar no dispositivo. A figura 4.3 apresenta o resultado da execução da aplicação, em que

é visualizada a tela principal denominada *Principal.* Além disso, é possível notar o preenchimento dos campos, bem como, o acionamento do botão finalizar.

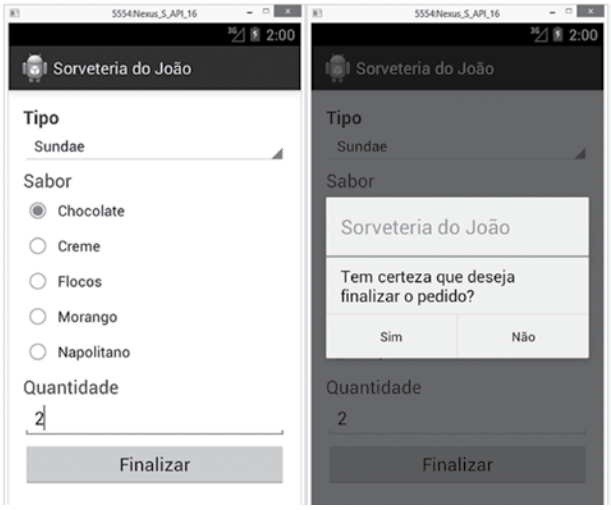

Figura 4.3 – Resultado da Execução do Aplicativo no Emulador.

A figura 4.4 demonstra o resultado da exibição da atividade Pedido, a qual será exibida quando o usuário pressionar o botão finalizar na atividade Principal. Nesta atividade, o pressionamento do botão Voltar finaliza a execução da atividade Pedido e apresenta novamente a atividade Principal.

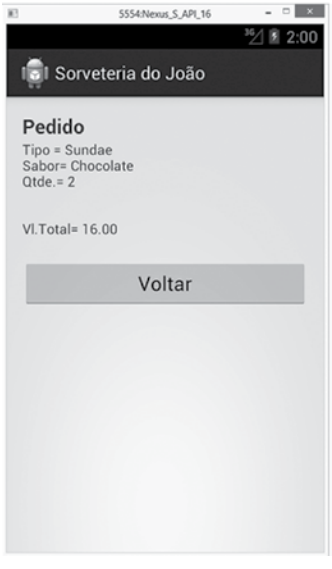

Figura 4.4 – Resultado da Execução do Aplicativo no Emulador.

Uma alternativa para o campo de entrada de dados quantidade (*edtQuantidade*) é usar o controle chamado de *NumberPicker*, que faz parte do pacote android.widget.*NumberPicker*. Com esse controle o usuário precisa selecionar um valor a partir de um intervalo pré-definido, em que são especificados o limite mínimo e máximo. Esta é uma ótima escolha para entrada de dados numéricos, uma vez, que os valores possível são apresentados na tela do usuário.

Para utilização do controle *NumberPicker* inicialmente é necessário incluir a codificação no arquivo de *layout* da interface. Na aplicação **App06** você pode, por exemplo, substituir o controle *EditText* para entrada da quantidade, por um controle do tipo *NumberPicker*. O trecho de código que apresenta esta mudança é demonstrado na sequência:

```
<NumberPicker
 android:id=”@+id/npkQuantidade”
 android:layout_width=”match_parent”
 android:layout_height=”wrap_content”
/>
```

O próximo passo é incluir na declaração nos atributos da classe Principal.java do controle *NumberPicker*, da seguinte maneira:

```
private NumberPicker npkQuantidade;
```

No método *onCreate* é preciso referenciar o novo controle adicionado ao layout da aplicação. Além disso, os métodos *setMinValue* e *setMaxValue* definem o valor inferior e superior do intervalo que será exibido para o usuário. Finalmente, o método *setWrapSelectorWheel* pode ser utilizado para definir qual o comportamento do intervalo numérico. Por exemplo, quando a propriedade é especificado como *setWrapSelectorWheel*(*true*), um intervalo circular é apresentado para o usuário. Assim, para um intervalo entre 0 e 100, quando o usuário atingir o valor 100 o próximo valor apresentado será o valor 0 e, vice-versa. O trecho para especificação do controle no método onCreate é apresentado na sequência:

```
npkQuantidade =
(NumberPicker)findViewById(R.id.npkQuantidade);
npkQuantidade.setMinValue(0);
npkQuantidade.setMaxValue(100);
npkQuantidade.setWrapSelectorWheel(false);
npkQuantidade.setValue(0);
```

Finalmente, para recuperar o valor selecionado pelo usuário é necessário utilizar o método getValue, por exemplo:

```
int qtde = npkQuantidade.getValue();
```

O resultado da inclusão deste controle no aplicativo **App06** é demonstrado na Figura 4. Note que a escolha da quantidade é realizada por meio do controle *NumberPicker*.

Dessa forma, finalizamos o conteúdo deste capítulo, em que você aprendeu sobre a construção de aplicativos com múltiplas atividades.

ATIVIDADES

Considerando o conteúdo apresentando neste capítulo, utilize as seguintes questões para avaliar o seu conhecimento:

01. Qual método da classe Intent é utilizado para passagem de parâmetros entre as atividades?

a) getIntent
b) getStringExtra
c) putExtra
d) putStringExtra
e) startActivity

02. Complete a sentença: O método _____ é utilizado para fechar a atividade atual e retornar para a atividade principal.

a) click
b) close
c) finish
d) onBackPressed
e) onClose

03. Qual instrução instancia adequadamente um objeto da classe Intent que será utilizado para abertura de uma atividade denominada Atividade2?

a) Atividade2 at = new Atividade2();
b) Intent it = new Intent();
c) Intent it = getIntent();
d) Intent it = new Intent(this,Atividade2.class)
e) Intent it = new Intent(Atividade2.class)

REFLEXÃO

A plataforma Android tem se tornado o principal framework para desenvolvimento de aplicativos para dispositivos móveis. A compatibilidade da plataforma com os inúmeros tipos de dispositivos é surpreendente, principalmente pelas diversas características de hardware. Neste contexto, faça uma reflexão sobre a seguinte questão: Como os desenvolvedores de aplicativos podem elaborar interfaces gráficas com o usuário para os mais variados tipos de telas e resoluções?

LEITURA

O desenvolvimento de aplicativos para Android oferece um universo repleto de possibilidades. Para aprender mais sobre a plataforma é sugerida a leitura do livro Android para Desenvolvedores, do autor Lúcio Camilo Oliva Pereira, que foi publicado pela editora Brasport em 2012.

REFERÊNCIAS BIBLIOGRÁFICAS

INTENT. Intent. Disponível em <http://developer.android.com/reference/android/content/Intent.html>, acesso em novembro de 2014.

5

Armazenamento Persistente de Dados

Armazenamento Persistente de Dados

Neste capítulo você conhecerá os recursos da plataforma Android para armazenamento persistente de dados. Com isso, será possível construir aplicativos capazes de salvar dados na memória secundária do aparelho. Além disso, estes dados poderão ser recuperados mesmo depois do aplicativo ser finalizado. Você aprenderá utilizar o banco de dados SQLite que é um recurso nativo do Android.

OBJETIVOS

Neste capítulo os objetivos da sua aprendizagem são:

- Conhecer os fundamentos do armazenamento persistente de dados em aplicativos móveis;
- Estudar os recursos disponíveis na plataforma Android para armazenamento de dados;
- Aprender a construir aplicativos utilizando o banco de dados SQLite;
- Desenvolver aplicativos capazes de inserir, atualizar, deletar e recuperar dados no banco de dados SQLite.

Fundamentos do SQLite

A plataforma Android oferece suporte nativo ao armazenamento persistente de dados utilizando o SQLite. Este é um banco de dados de código fonte aberto, multiplataforma e tem sido utilizado principalmente em aplicativos embarcados. A arquitetura de funcionamento do SQLite é diferente de outros sistemas de gerenciamento de banco de dados tradicionais, uma vez, que todas as funcionalidades do sistema são oferecidas por meio de uma biblioteca escrita na linguagem C. Este fato torna o SQLite uma solução leve, e de fácil integração, pois não necessita de instalação, configuração e administração. O SQLite é compatível com um grande número de linguagens de programação, e as instruções SQL utilizadas no banco são baseadas no padrão SQL-92 (SQLITE, 2014).

No contexto da plataforma Android, o SQLite é encapsulado em um provedor de conteúdo (*content provider*). Dessa forma, os dados são manipulados de maneira abstrata e podem ser disponibilizados para os diversos serviços oferecidos nos dispositivos. As bases de dados criadas com o SQLite são privadas e acessíveisapenas pelo aplicativo responsável, ou, disponibilizada para outros aplicativos

desde que atribuídas as devidas permissões. O armazenamento persistente dos dados ocorre em um caminho pré-definido do dispositivo, da seguinte maneira:

```
/data/data/<nome_do_pacote>/databases
```

CONEXÃO

A linguagem de consulta estruturada SQL é utilizada para acessar informações armazenadas no banco de dados. Esta linguagem oferece uma série de instruções para manipulação dos dados. Caso você tenha dúvidas, ou deseja aprimorar seus conhecimentos, você pode consultar o tutorial de SQL disponível no site W3Schools.com. O endereço para acesso é:

<http://www.w3schools.com/sql/>.

Em que *<nome_do_pacote>* é o identificador definido durante a criação do aplicativo.

O SQLite é um banco de dados relacional, assim, para o armazenamento persistente de dados é necessário definir tabelas e seus respectivos campos. Os tipos de dados suportados pelo SQLite são (DATATYPES, 2014):

CONEXÃO

O banco de dados SQLite oferece aos desenvolvedores de aplicativos o recurso de armazenamento persistente de dados. Você pode encontrar informações detalhadas sobre esta biblioteca de software no site oficial, no seguinte endereço:

<http://www.sqlite.org/>.

- **NULL**: valores nulos.
- **INTEGER**: para armazenamento de números inteiros.
- **REAL**: para manipulação de valores ponto-flutuante (por exemplo, double, float).
- **TEXT**: para dados textuais
- **BLOB**: armazenamento de objetos binários (arquivos, imagens, vídeos, entre outros).

A listagem Código 1 apresenta um típico script na linguagem SQL para criação de uma tabela, denominada pessoa, com os campos *id, nome, endereço, peso e altura* no banco de dados SQLite.

Código 1

```
1.          CREATE TABLE pessoa (
2.          id INTEGER PRIMARY KEY AUTOINCREMENT,
3.          nome TEXT,
4.          endereco TEXT,
5.          peso REAL,
6.          altura REAL
7.      );
```

Manipulação de Dados

Na plataforma Android a manipulação de dados no banco SQLite é facilitada com o uso da classe *SQLiteOpenHelper*. Esta classe auxilia o desenvolvedor no processo de criação e gerenciamento das versões do banco de dados. Em termos de programação, a classe *SQLiteOpenHelper* é utilizada como superclasse em uma hierarquia e possui dois métodos abstratos que necessitam ser implementados: *OnCreate* e *OnUpgrade*. A listagem Código 2 apresenta a estrutura típica de uma classe para manipulação de dados com o SQLite que herda a classe *SQLiteOpenHelper*. No código é importante destacar a especificação dos atributos *DATABASE_NAME e DATABASE_VERSION*, os quais são utilizados respectivamente para: definir o nome do arquivo do banco de dados que será criado no dispositivos; e indicar qual a versão da estrutura do banco de dados que está sendo utilizada.

Código 2

```
1.      public class DatabaseHelper extends SQLiteOpenHelper{
2.
3.      //ATRIBUTOS
4.      private static final String DATABASE_NAME =
5.      “exemplo”;
6.      private static final int DATABASE_VERSION = 1;
```

```
7.     private final String CREATE_TABLE_PESSOA=
8.            “CREATE TABLE pessoa (
9.                   id INTEGER PRIMARY KEY AUTOINCREMENT,
10.                  nome TEXT,
11.                  endereco TEXT,
12.                  peso REAL,
13.                  altura REAL
14.                  );
15.           “;
16.
17.    //CONSTRUTOR
18.    public DatabaseHelper(Context context){
19.      super(
20.         context,               //contexto da App
21.           DATABASE_NAME,       //nome do banco de dados
22.           null,                       //cursor para manipulação
23.           DATABASE_VERSION     //versão do banco de dados
24.       );
25.    }
26.
27.    // MÉTODO onCreate
28.    @Override
29.    public void onCreate(SQLiteDatabase db) {
30.           db.execSQL(CREATE_TABLE_PESSOA);
31.    }
32.
33.    // MÉTODO onUpgrade
34.    @Override
35.    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
36.           db.execSQL(“DROP TABLE IF EXISTS pessoa”);
37.           onCreate(db);
38.    }
39.    }
```

No Código 2 o método *onCreate* é utilizado para criar um novo banco de dados no dispositivo. Este método realiza uma verificação prévia a respeito da existência, assim, o script de criação da tabela (ou das tabelas) será executado apenas quando o banco de dados não existir. O método *onUpgrade* é empregado na modificação da estrutura do banco de dados. Este método faz uma comparação entre a versão do banco de dados da aplicação (*DATABASE_VERSION*) e a versão do banco de dados que está armazenado no dispositivo. Caso as versões sejam diferentes o conteúdo o método *onUpgrade* é executado.

A classe *SQLiteOpenHelper* oferece ainda outros métodos para manipulação e gerenciamento do banco de dados. O método *getWritableDatabase*, por exemplo, é utilizado para criação e/ou abertura do banco de dados, permitindo que sejam realizadas leituras e escritas de dados. O método *getReadableDatabase*, realiza a abertura e/ou criação do banco de dados, permitindo apenas que sejam efetuadas leituras dos dados.

A comunicação entre o aplicativo e o banco de dados SQLite por meio de instruções SQL é realizada com a classe *SQLiteStatement*. O objetivo desta classe é permitir a execução de instruções SQL e o seu funcionamento é semelhante a classe PreparedStatement do JDBC (*Java Database Connectivity*). A listagem Código 3 demonstra o uso da classe *SQLiteStatement* para inserção de um novo registro no banco de dados. Para isso, foi considerada a tabela Pessoa especificada no Código 1 e a classe *DatabaseHelper* do Código 2.

Código 3

```
1.     public int inserir(){
2.
3.            String SQL = “INSERT INTO pessoa
4.     (nome,endereco,peso,altura) VALUES (?,?,?,?)”;
5.
6.         SQLiteDatabase db = this.getWritableDatabase();
7.         SQLiteStatement cmd = db.compileStatement(sql);
8.
9.            //Parâmetros
10.           cmd.bindString(1, “João da Silva”);
11.           cmd.bindString(2, “Rua Niteroi, 1234”);
12.           cmd.bindDouble(3, 92.5);
13.           cmd.bindDouble(4, 1.95);
```

```
14.
15.            //Insere o registro e retorna o ID gerado
16.            return cmd.executeInsert();
17.    }
```

A classe *SQLiteDatabase* oferece uma maneira alternativa para manipulação dos dados com o SQLite. Esta classe possui métodos encapsulados para as operações de inserção (*insert*), atualização (*update*) e deleção (*delete*), com isso, não é necessário utilizar instruções SQL durante a comunicação com o SGBD. Os dados passados para os métodos devem ser armazenados em objetos da classe *ContentValues*.

A classe *ContentValue* permite a instanciação de objetos capazes de armazenar uma série de valores, os quais podem ser enviados para o SGBD. Em uma operação de inserção, por exemplo, os valores que serão adicionados a tabela são definidos no objeto *ContentValue*. A sintaxe a seguir demonstra a criação de um objeto *ContentValue*:

```
ContentValues obj = new ContentValues();
```

Para atribuir valores ao objeto é necessário o uso do método put, o qual deve receber dois parâmetros <chave,valor>, tal que, chave –indica o nome do campo da tabela; valor –representa o valor associado ao campo. Por exemplo, considerando a tabela Pessoa com nome, endereço, peso e altura, a definição do objeto ContentValue é realizada da seguinte maneira:

```
obj.put("nome","João da Silva");
obj.put("endereco","Rua Niteroi, 1234");
obj.put("peso", 92.5);
obj.put("altura,1.95);
```

Este objeto pode ser enviado como parâmetro nas operações de inserção, atualização e deleção da classe SQLiteDatabase.

Para retornar dados a partir de consultas as bases de dados do }SQLite, a plataforma Android disponibiliza uma classe denominada *Cursors*. Esta classe cria um objeto capaz de referenciar as linhas resultantes da consulta, além disso, possui funções de navegação, tais como: *moveToFirst* – para mover o cursor para primeira linha do resultado; *moveToNext* – para movimentar o cursor para próxima linha; *moveToPrevious* – para retroceder o cursor, entre outros.

Exemplo: Aplicativo para Catálogo de Livros

Neste exemplo é demonstrado o uso do armazenamento persistente de dados no SQLite a partir de um aplicativo que controla um Catálogo de Livros. Este aplicativo possibilita cadastrar novos livros, bem como realizar pesquisar por título e ano de publicação. Além disso, é possível listar todos os registros cadastrados nos banco de dados. Para iniciar os trabalhos crie um novo aplicativo no ambiente Android Studio intitulado **App07**. O aplicativo é composto por três telas:

- TelaPrincipal (activity_tela_principal)
- TelaCadastrar (activity_tela_cadastrar)
- TelaPesquisar (activity_tela_pesquisar)

O resultado da execução do aplicativo no emulador é demonstrado na figura 5.1.

Figura 5.1 –

O primeiro passo para codificação do aplicativo é a especificação do conteúdo textual estático no arquivo strings.xml. A listagem Código 4 apresenta as constantes do arquivo strings.xml.

Código 4

```
1.      <?xml version="1.0" encoding="utf-8"?>
2.      <resources>
3.
```

```
4.          <string name=”app_name”>Catálogo Livros</string>
5.          <string name=”txt_cadastrar”>Cadastrar</string>
6.          <string name=”txt_pesquisar”>Pesquisar</string>
7.          <string name=”txt_pesquisarpor”>Pesquisa
8.      por</string>
9.
10.         <string name=”txt_pesquisar_todos”>Todos</string>
11.         <string name=”txt_id”>Id</string>
12.         <string name=”txt_titulo”>Título</string>
13.         <string name=”txt_autor”>Autor</string>
14.         <string name=”txt_ano”>Ano</string>
15.
16.         <string name=”btn_salvar”>Salvar</string>
17.         <string name=”btn_voltar”>Voltar</string>
18.
19.         <string ]
20.     name=”title_activity_tela_cadastrar”>Catálogo
21.     Livros</string>
22.         <string
23.     name=”title_activity_tela_pesquisar”>Catálogo
24.     Livros</string>
25.
26.     </resources>
27.
```

Para manipulação do banco de dados SQLite, uma nova classe Java foi adicionada ao projeto com o nome de *DataBaseHelper*. Esta classe herda características e comportamentos da classe *SQLiteOpenHelper*, além disso, incluir métodos para inserção, pesquisa e listagem dos dados. Os dados do aplicativo são armazenados em uma tabela denominada catalogo e possui os campos *id, titulo, autor e ano*. A listagem Código 5 apresenta o conteúdo da classe *DatabaseHelper.java*.

Código 5

```
1.      package br.android.app07;
2.
3.      import android.content.ContentValues;
```

```
4.      import android.content.Context;
5.      import android.database.Cursor;
6.      import android.database.sqlite.SQLiteDatabase;
7.      import android.database.sqlite.SQLiteOpenHelper;
8.
9.      import java.util.ArrayList;
10.     import java.util.List;
11.
12.     public class DatabaseHelper extends SQLiteOpenHelper {
13.
14.         //ATRIBUTOS
15.         private static final String DATABASE_NAME =
16.     “catalogo”;
17.         private static final int DATABASE_VERSION = 3;
18.         private final String CREATE_TABLE_CATALOGO=
19.                 “CREATE TABLE catalogo (“
20.                 + “id INTEGER PRIMARY KEY AUTOINCREMENT,”
21.                 + “titulo TEXT, autor TEXT,”
22.                 + “ano INTEGER);”;
23.
24.         public  DatabaseHelper(Context context){
25.            super(context, DATABASE_NAME,null,
26.     DATABASE_VERSION);
27.        }
28.
29.         @Override
30.         public void onCreate(SQLiteDatabase db) {
31.             db.execSQL(CREATE_TABLE_CATALOGO);
32.         }
33.
34.         @Override
35.         public void onUpgrade(SQLiteDatabase db, int
36.     oldVersion, int newVersion) {
37.             db.execSQL(“DROP TABLE IF EXISTS catalogo”);
38.             onCreate(db);
39.         }
```

```
40.
41.         public long inserir(ContentValues cv){
42.             SQLiteDatabase db =
43.     this.getWritableDatabase();
44.             long id = db.insert(“catalogo”,null,cv);
45.             return id;
46.         }
47.
48.
49.         public
50.     List<ContentValues> pesquisarPorTitulo(String titulo){
51.             String sql = “SELECT * FROM catalogo WHERE
52.     titulo LIKE ?”;
53.             String where[] = new String[]{“%”+titulo+”%”};
54.             return pesquisar(sql,where);
55.         }
56.
57.         public List<ContentValues> pesquisarPorAno(int
58.     ano){
59.             String sql = “SELECT * FROM catalogo WHERE
60.     ano=?”;
61.             String where[] = new
62.     String[]{String.valueOf(ano)};
63.             return pesquisar(sql,where);
64.         }
65.
66.         public List<ContentValues> pesquisarPorTodos(){
67.             String sql = “SELECT * FROM catalogo ORDER BY
68.     id”;
69.             String where[] = null;
70.             return pesquisar(sql,where);
71.         }
72.
73.         private List<ContentValues> pesquisar(String
74.     sql,String where[]){
```

```
75.            List<ContentValues> lista = new ArrayList<>();
76.
77.            SQLiteDatabase db =
78.    this.getReadableDatabase();
79.            Cursor c = db.rawQuery(sql, where);
80.
81.            if (c.moveToFirst()){
82.                do{
83.                 ContentValues cv = new ContentValues();
84.          cv.put("id",c.getInt(c.getColumnIndex("id")));
85.
86.    cv.put("titulo",c.getString(c.getColumnIndex("titulo")
87.    ));
88.                    cv.put("autor",
89.    c.getString(c.getColumnIndex("autor")));
90.                    cv.put("ano",
91.    c.getInt(c.getColumnIndex("ano")));
92.                    lista.add(cv);
93.                }while(c.moveToNext());
94.            }
95.            return lista;
96.        }
97.
98.    }
```

CONCEITO

A classe *SQLiteDatabase* permite realizar operações com a base de dados sem a necessidade de instruções SQL. Para realizar a atualização de um registro é preciso, apenas, chamar o método update, por exemplo, *SQLiteDatabase* db.update informando o nome da tabela, os valores e a condição (*where*). Caso seja necessário apagar um registro é possível chamar o método delete, também da classe *SQLiteDatabase*.

No Código 5 é importante destacar a definição do método inserir que realiza a inclusão de um novo registro ao banco de dados. O método retornar ainda o id do

novo registro adicionado. Na listagem é também merece observação os métodos de pesquisa (*pesquisarPorTitulo, pesquisarPorAno e pesquisarPorTodos*), os quais realizam uma chamada do método *pesquisar*. Esse método é responsável por efetuar a consulta na base de dados e retornar uma lista dinâmica (*ArrayList*) contendo todos os registros encontrados na forma de objetos da classe *ContentValues*.

A listagem Código 6 apresenta o conteúdo do arquivo de layout da tela principal do aplicativo. Este arquivo foi nomeado na estrutura do projeto como *activity_tela_principal*. No código é possível notar a utilização do controle *ImageView* que permite a inserção de imagens nas tela. Para visualizar corretamente o resultado é necessário incluir um arquivo de imagem na paste *drawable*. No exemplo, foi utilizado um arquivo denominado *img_livros.png*.

COMENTÁRIO

Para trocar o ícone de um aplicativo Android basta alterar o arquivo *ic_laucher.png* disponível na pasta *drawable* do projeto. A resolução do ícone está relacionada com as configurações da tela do dispositivo. As resoluções recomendadas para os ícones são: MDPI 48x48 pixels; HDPI 72x72 pixels; XHDPI 96x96 pixels; e XXHDPI 144x144 pixels.

Código 6

```
1.      <LinearLayout
2.      xmlns:android=”http://schemas.android.com/apk/res/and
3.      roid”
4.          xmlns:tools=”http://schemas.android.com/tools”
5.      android:layout_width=”match_parent”
6.          android:layout_height=”match_parent”
7.      android:paddingLeft=”@dimen/activity_horizontal_margi
8.      n”
9.
10.     android:paddingRight=”@dimen/activity_horizontal_marg
11.     in”
12.
13.     android:paddingTop=”@dimen/activity_vertical_margin”
14.
15.     android:paddingBottom=”@dimen/activity_vertical_margi
16.     n”
```

```
17.         tools:context=".TelaPrincipal"
18.         android:orientation="vertical"
19.         android:background="#FFFFFF">
20.
21.         <LinearLayout
22.             android:orientation="horizontal"
23.             android:layout_width="match_parent"
24.             android:layout_height="wrap_content"
25.             android:layout_marginBottom="20sp">
26.
27.             <ImageView
28.                 android:src="@drawable/img_livros"
29.                 android:layout_width="wrap_content"
30.                 android:layout_height="wrap_content" />
31.
32.             <TextView
33.                 android:text="@string/app_name"
34.                 android:layout_gravity="center_vertical"
35.                 android:textSize="32sp"
36.                 android:layout_width="wrap_content"
37.                 android:layout_height="wrap_content" />
38.
39.         </LinearLayout>
40.
41.         <Button
42.             android:id="@+id/btnCadastrar"
43.             android:text="@string/txt_cadastrar"
44.             android:layout_width="match_parent"
45.             android:layout_height="wrap_content"
46.             android:textSize="20sp"
47.             android:layout_marginBottom="20sp"/>
48.
49.         <TextView
50.             android:text="@string/txt_pesquisarpor"
51.             android:layout_gravity="center_vertical"
52.             android:textSize="20sp"
```

```
53.             android:textColor=”#000000”
54.             android:layout_width=”wrap_content”
55.             android:layout_height=”wrap_content” />
56.
57.         <RadioGroup
58.             android:id=”@+id/rdgPesquisarPor”
59.             android:orientation=”horizontal”
60.             android:layout_width=”match_parent”
61.             android:layout_height=”wrap_content”>
62.
63.             <RadioButton
64.                 android:id=”@+id/rbPesquisarPorTitulo”
65.                 android:text=”@string/txt_titulo”
66.                 android:checked=”true”
67.                 android:layout_width=”wrap_content”
68.                 android:layout_height=”wrap_content”
69.                 android:textSize=”20sp” />
70.
71.             <RadioButton
72.                 android:id=”@+id/rbPesquisarPorAno”
73.                 android:text=”@string/txt_ano”
74.                 android:layout_width=”wrap_content”
75.                 android:layout_height=”wrap_content”
76.                 android:textSize=”20sp” />
77.
78.             <RadioButton
79.                 android:id=”@+id/rbPesquisarPorTodos”
80.
81.     android:text=”@string/txt_pesquisar_todos”
82.                 android:layout_width=”wrap_content”
83.                 android:layout_height=”wrap_content”
84.                 android:textSize=”20sp” />
85.
86.          </RadioGroup>
87.
88.
```

```
89.         <EditText
90.             android:id=”@+id/edtPesquisar”
91.             android:layout_width=”match_parent”
92.             android:layout_height=”wrap_content” />
93.
94.         <Button
95.             android:id=”@+id/btnPesquisar”
96.             android:text=”@string/txt_pesquisar”
97.             android:layout_width=”match_parent”
98.             android:layout_height=”wrap_content”
99.             android:textSize=”20sp”
100.            android:layout_marginBottom=”10sp”/>
101.
102.    </LinearLayout>
```

A lógica de programação da tela principal foi codificada na classe TelaPrincipal.java. A listagem Código 7 apresenta o conteúdo deste arquivo. No Código é importante notar o uso do objeto *Intent* para iniciar as demais atividades do projeto.

Código 7

```
1.      package br.android.app07;
2.
3.      import android.content.ContentValues;
4.      import android.content.Intent;
5.      import android.support.v7.app.ActionBarActivity;
6.      import android.os.Bundle;
7.      import android.view.Menu;
8.      import android.view.MenuItem;
9.      import android.view.View;
10.     import android.view.Window;
11.     import android.view.WindowManager;
12.     import android.widget.Button;
13.     import android.widget.EditText;
14.     import android.widget.RadioGroup;
15.
16.
```

```
17.    public class TelaPrincipal extends ActionBarActivity
18.    implements View.OnClickListener {
19.
20.        private Button btnCadastrar;
21.        private Button btnPesquisar;
22.        private RadioGroup rdgPesquisarPor;
23.        private EditText edtPesquisar;
24.
25.        @Override
26.        protected void onCreate(Bundle
27.    savedInstanceState) {
28.            super.onCreate(savedInstanceState);
29.    setContentView(R.layout.activity_tela_principal);
30.
31.            btnCadastrar =
32.    (Button)findViewById(R.id.btnCadastrar);
33.            btnPesquisar =
34.    (Button)findViewById(R.id.btnPesquisar);
35.            rdgPesquisarPor =
36.    (RadioGroup)findViewById(R.id.rdgPesquisarPor);
37.            edtPesquisar =
38.    (EditText)findViewById(R.id.edtPesquisar);
39.
40.            btnCadastrar.setOnClickListener(this);
41.            btnPesquisar.setOnClickListener(this);
42.        }
43.
44.        @Override
45.        public void onClick(View v) {
46.            Intent it = null;
47.            switch (v.getId()){
48.                case R.id.btnCadastrar:
49.                    it = new
50.    Intent(this,TelaCadastrar.class);
51.                    break;
52.                case R.id.btnPesquisar:
```

```
53.                     it = new
54.    Intent(this,TelaPesquisar.class);
55.
56.    it.putExtra(“tipo”,rdgPesquisarPor.getCheckedRadioBut
57.    tonId());
58.                     it.putExtra(“chave”,
59.    edtPesquisar.getText().toString());
60.                     break;
61.            }
62.
63.            if (it!= null){
64.                startActivity(it);
65.            }
66.        }
67.    }
```

A listagem Código 8 apresenta o conteúdo do arquivo de layout da tela de cadastro. Nesta tela foram definidos os campos de entrada para o título, autor e ano do livro, bem como, os botões salvar e voltar.

Código 8

```
1.     <LinearLayout
2.     xmlns:android=”http://schemas.android.com/apk/res/and
3.     roid”
4.         xmlns:tools=”http://schemas.android.com/tools”
5.     android:layout_width=”match_parent”
6.         android:layout_height=”match_parent”
7.     android:paddingLeft=”@dimen/activity_horizontal_margi
8.     n”
9.
10.    android:paddingRight=”@dimen/activity_horizontal_marg
11.    in”
12.
13.    android:paddingTop=”@dimen/activity_vertical_margin”
14.
15.    android:paddingBottom=”@dimen/activity_vertical_margi
```

```
16.     n”
17.         tools:context=”br.android.app07.TelaCadastrar”
18.         android:orientation=”vertical”
19.         android:background=”#FFFFFF”>
20.
21.         <LinearLayout
22.             android:orientation=”horizontal”
23.             android:layout_width=”match_parent”
24.             android:layout_height=”wrap_content”
25.             android:layout_marginBottom=”10sp”>
26.
27.             <ImageView
28.                 android:src=”@drawable/img_livros”
29.                 android:layout_width=”wrap_content”
30.                 android:layout_height=”wrap_content” />
31.
32.             <TextView
33.                 android:text=”@string/app_name”
34.                 android:layout_gravity=”center_vertical”
35.                 android:textSize=”32sp”
36.                 android:layout_width=”wrap_content”
37.                 android:layout_height=”wrap_content” />
38.
39.         </LinearLayout>
40.
41.         <TextView
42.             android:text=”@string/txt_cadastrar”
43.             android:textSize=”32sp”
44.             android:layout_width=”match_parent”
45.             android:layout_height=”wrap_content” />
46.
47.         <TextView
48.             android:layout_width=”match_parent”
49.             android:layout_height=”wrap_content”
50.             android:layout_marginTop=”5sp”
51.             android:text=”@string/txt_titulo”
```

```
52.             android:textSize="14sp" />
53.
54.         <EditText
55.             android:id="@+id/edtTitulo"
56.             android:layout_width="match_parent"
57.             android:layout_height="wrap_content"
58.             android:background="#CECECE"
59.             android:textSize="24sp" />
60.
61.         <TextView
62.             android:layout_width="match_parent"
63.             android:layout_height="wrap_content"
64.             android:layout_marginTop="5sp"
65.             android:text="@string/txt_autor"
66.             android:textSize="14sp" />
67.
68.         <EditText
69.             android:id="@+id/edtAutor"
70.             android:layout_width="match_parent"
71.             android:layout_height="wrap_content"
72.             android:background="#CECECE"
73.             android:textSize="24sp" />
74.
75.         <TextView
76.             android:layout_width="match_parent"
77.             android:layout_height="wrap_content"
78.             android:layout_marginTop="5sp"
79.             android:text="@string/txt_ano"
80.             android:textSize="14sp" />
81.
82.         <EditText
83.             android:id="@+id/edtAno"
84.             android:layout_width="wrap_content"
85.             android:layout_height="wrap_content"
86.             android:width="200sp"
87.             android:inputType="number"
```

```
88.             android:background=”#CECECE”
89.             android:textSize=”24sp” />
90.
91.
92.         <Button
93.             android:id=”@+id/btnSalvar”
94.             android:text=”@string/btn_salvar”
95.             android:layout_width=”match_parent”
96.             android:layout_height=”wrap_content”
97.             android:textSize=”20sp”
98.             android:layout_marginTop=”10sp”/>
99.
100.        <Button
101.            android:id=”@+id/btnVoltar”
102.            android:text=”@string/btn_voltar”
103.            android:layout_width=”match_parent”
104.            android:layout_height=”wrap_content”
105.            android:textSize=”20sp”       />
106.
107.    </LinearLayout>
```

A listagem Código 9 apresenta o conteúdo da classe TelaCadastrar.java em que pode ser observado a chamada do método inserir (linha 52) da classe DatabaseHelper, o qual efetua a inclusão de um novo livro no banco de dados.

Código 9

```
1.      package br.android.app07;
2.
3.      import android.content.ContentValues;
4.      import android.support.v7.app.ActionBarActivity;
5.      import android.os.Bundle;
6.      import android.view.Menu;
7.      import android.view.MenuItem;
8.      import android.view.View;
9.      import android.view.Window;
10.     import android.view.WindowManager;
```

```
11.    import android.widget.Button;
12.    import android.widget.EditText;
13.    import android.widget.Toast;
14.
15.
16.    public class TelaCadastrar extends ActionBarActivity
17.    implements View.OnClickListener {
18.
19.        private Button btnSalvar;
20.        private Button btnVoltar;
21.        private EditText edtTitulo;
22.        private EditText edtAutor;
23.        private EditText edtAno;
24.
25.        @Override
26.        protected void onCreate(Bundle
27.    savedInstanceState) {
28.            super.onCreate(savedInstanceState);
29.
30.    setContentView(R.layout.activity_tela_cadastrar);
31.
32.            btnSalvar =
33.    (Button)findViewById(R.id.btnSalvar);
34.            btnVoltar =
35.    (Button)findViewById(R.id.btnVoltar);
36.            edtTitulo =
37.    (EditText)findViewById(R.id.edtTitulo);
38.            edtAutor =
39.    (EditText)findViewById(R.id.edtAutor);
40.            edtAno = (EditText)findViewById(R.id.edtAno);
41.
42.            btnSalvar.setOnClickListener(this);
43.            btnVoltar.setOnClickListener(this);
44.        }
45.
46.        @Override
```

```
47.         public void onClick(View v) {
48.             if (v.getId() == R.id.btnVoltar){
49.                 onBackPressed();
50.             }else if (v.getId() == R.id.btnSalvar){
51.
52.                 ContentValues cv = new ContentValues();
53.
54.     cv.put(“titulo”,edtTitulo.getText().toString());
55.
56.     cv.put(“autor”,edtAutor.getText().toString());
57.
58.     cv.put(“ano”,edtAno.getText().toString());
59.
60.                 DatabaseHelper dh = new
61.     DatabaseHelper(this);
62.                 String msg = “”;
63.                 if ( dh.inserir(cv) > 0 ){
64.                     msg = “Operação realizada com
65.     sucesso!”;
66.                     edtTitulo.setText(“”);
67.                     edtAutor.setText(“”);
68.                     edtAno.setText(“”);
69.                     edtTitulo.requestFocus();
70.                 }else{
71.                     msg = “Ocorreu um erro durante a
72.     operação.”;
73.                 }
74.
75.     Toast.makeText(this,msg,Toast.LENGTH_LONG).show();
76.             }
77.         }
78.     }
79.
80.
```

Neste aplicativo, a tela de pesquisa é utilizada tanto para o proposito de apresentar os registros encontrados com as operações de pesquisa por título ou por ano, bem como, a listagem de todos os registros disponíveis no banco de dados. Para construção da tela foi o controle *TableLayout* que permite a exibição de dados na forma de tabela, assim, para cada registro disponível no banco foi criada uma nova linha TableRow. A listagem Código 10 demonstra o conteúdo do arquivo de layout da tela de pesquisa (*activity_tela_pesquisar.xml*).

Código 10

```
1.      <LinearLayout
2.      xmlns:android=”http://schemas.android.com/apk/res/and
3.      roid”
4.          xmlns:tools=”http://schemas.android.com/tools”
5.      android:layout_width=”match_parent”
6.          android:layout_height=”match_parent”
7.      android:paddingLeft=”@dimen/activity_horizontal_margi
8.      n”
9.
10.     android:paddingRight=”@dimen/activity_horizontal_marg
11.     in”
12.
13.     android:paddingTop=”@dimen/activity_vertical_margin”
14.
15.     android:paddingBottom=”@dimen/activity_vertical_margi
16.     n”
17.         tools:context=”br.android.app07.TelaPesquisar”
18.         android:orientation=”vertical”
19.         android:background=”#FFFFFF”>
20.
21.
22.         <LinearLayout
23.             android:orientation=”horizontal”
24.             android:layout_width=”match_parent”
25.             android:layout_height=”wrap_content”
26.             android:layout_marginBottom=”10sp”>
27.
```

```
28.             <ImageView
29.                 android:src=”@drawable/img_livros”
30.                 android:layout_width=”wrap_content”
31.                 android:layout_height=”wrap_content” />
32.
33.             <TextView
34.                 android:text=”@string/app_name”
35.                 android:layout_gravity=”center_vertical”
36.                 android:textSize=”32sp”
37.                 android:layout_width=”wrap_content”
38.                 android:layout_height=”wrap_content” />
39.
40.         </LinearLayout>
41.
42.         <TextView
43.             android:text=”@string/txt_pesquisar”
44.             android:textSize=”32sp”
45.             android:layout_width=”match_parent”
46.             android:layout_height=”wrap_content” />
47.
48.
49.         <TableLayout
50.             android:id=”@+id/tbPesquisa”
51.             android:layout_width=”match_parent”
52.             android:layout_height=”wrap_content”
53.             android:shrinkColumns=”*”
54.             android:stretchColumns=”*”>
55.
56.             <TableRow
57.                 android:id=”@+id/trPesquisa”
58.                 android:layout_width=”match_parent”
59.                 android:layout_height=”wrap_content”
60.                 android:layout_margin=”1dp” >
61.
62.                 <TextView
63.                     android:id=”@+id/txtId”
```

```
64.                 android:layout_width="match_parent"
65.                 android:layout_height="wrap_content"
66.                 android:text="@string/txt_id"
67.                 android:textSize="18sp"
68.                 android:textStyle="bold" />
69.
70.             <TextView
71.                 android:id="@+id/txtTitulo"
72.                 android:layout_width="match_parent"
73.                 android:layout_height="wrap_content"
74.                 android:text="@string/txt_titulo"
75.                 android:textSize="18sp"
76.                 android:textStyle="bold" />
77.             <TextView
78.                 android:id="@+id/txtAutor"
79.                 android:layout_width="match_parent"
80.                 android:layout_height="wrap_content"
81.                 android:text="@string/txt_autor"
82.                 android:textSize="18sp"
83.                 android:textStyle="bold" />
84.             <TextView
85.                 android:id="@+id/txtAno"
86.                 android:layout_width="match_parent"
87.                 android:layout_height="wrap_content"
88.                 android:text="@string/txt_ano"
89.                 android:textSize="18sp"
90.                 android:textStyle="bold" />
91.         </TableRow>
92.     </TableLayout>
93.
94.
95.     <Button
96.         android:id="@+id/btnVoltar"
97.         android:text="@string/btn_voltar"
98.         android:layout_width="match_parent"
99.         android:layout_height="wrap_content"
```

```
100.            android:textSize=”20sp”       />
101.
102.    </LinearLayout>
103.
```

Finalmente, a listagem Código 11 apresenta o conteúdo do arquivo TelaPesquisar.java, contendo a lógica de programação da tela de pesquisa. No Código é demonstrado a criação de linhas em tempo de execução, assim, a quantidade de linhas da tabela esta condicionada ao número de linhas retornado com a consulta na base de dados.

Código 11

```
1.      package br.android.app07;
2.
3.      import android.content.ContentValues;
4.      import android.content.Intent;
5.      import android.support.v7.app.ActionBarActivity;
6.      import android.os.Bundle;
7.      import android.view.Menu;
8.      import android.view.MenuItem;
9.      import android.view.View;
10.     import android.view.Window;
11.     import android.view.WindowManager;
12.     import android.widget.Button;
13.     import android.widget.TableLayout;
14.     import android.widget.TableRow;
15.     import android.widget.TextView;
16.
17.     import java.util.ArrayList;
18.     import java.util.List;
19.
20.
21.     public class TelaPesquisar extends ActionBarActivity
22.     implements View.OnClickListener {
23.
24.     private Button btnVoltar;
```

```
25.
26.     @Override
27.     protected void onCreate(Bundle savedInstanceState) {
28.             super.onCreate(savedInstanceState);
29.             setContentView(R.layout.activity_tela_pesquisar)
30.     ;
31.
32.             btnVoltar = (Button)
33.     findViewById(R.id.btnVoltar);
34.             btnVoltar.setOnClickListener(this);
35.
36.             Intent it = getIntent();
37.             if (it != null){
38.                     int tipo = it.getIntExtra(“tipo”,0);
39.                     String chave = it.getStringExtra(“chave”);
40.
41.                     List<ContentValues> lista = new
42.     ArrayList<>();
43.                     if(tipo == R.id.rbPesquisarPorTitulo) {
44.                             lista = new
45.     DatabaseHelper(this).pesquisarPorTitulo(chave);
46.                     }else if (tipo == R.id.rbPesquisarPorAno){
47.                             try{
48.                                     lista = new
49.     DatabaseHelper(this).pesquisarPorAno(Integer.parseInt
50.     (chave));
51.                             }catch(Exception e){
52.                                     lista = new
53.     DatabaseHelper(this).pesquisarPorTodos();
54.                             }
55.                     }else if (tipo ==R.id.rbPesquisarPorTodos){
56.                             lista = new
57.     DatabaseHelper(this).pesquisarPorTodos();
58.                     }
59.
60.
```

```
61.     if(lista!=null){
62.             if(lista.size()>0){
63.                     TableLayout tb =
64.     (TableLayout)findViewById(R.id.tbPesquisa);
65.                     for(ContentValues cv: lista){
66.                             TableRow tr = new TableRow(this);
67.
68.                             TextView col1 = new TextView(this);
69.
70.             col1.setText(String.valueOf(cv.getAsInteger(“id”
71.     )));
72.                             tr.addView(col1);
73.
74.                             TextView col2 = new TextView(this);
75.
76.             col2.setText(cv.getAsString(“titulo”));
77.                             tr.addView(col2);
78.
79.                             TextView col3 = new TextView(this);
80.
81.             col3.setText(cv.getAsString(“autor”));
82.                             tr.addView(col3);
83.
84.                             TextView col4 = new TextView(this);
85.             col4.setText(String.valueOf(cv.getAsInteger(“ano”
86.     )));
87.                             tr.addView(col4);
88.
89.                             tb.addView(tr);
90.                     }
91.             }
92.     }
93.             }
94.     }
95.
96.     @Override
```

```
97.     public void onClick(View v) {
98.             if (v.getId() == R.id.btnVoltar){
99.                     onBackPressed();
100.            }
101.    }
102.    }
```

Com isso, terminamos o conteúdo deste capítulo que demonstrou o uso da plataforma Android no armazenamento persistente de dados com o SQLite.

ATIVIDADES

Com base no conteúdo apresentado neste capítulo, responda as seguintes questões objetivas:

01. Qual tipo de dados do SQLite pode ser utilizado para armazenamento de valores ponto-flutuante?

a) NULL
b) INTEGER
c) REAL
d) TEXT
e) BLOB

02. Qual classe da plataforma Android possui métodos encapsulados para insert, update e delete?

a) SQLiteOpenHelper
b) SQLiteStatement
c) SQLiteDatabase
d) ContentValues
e) Cursor

03. Complete a sentença: O objeto _____ é capaz de referenciar as linhas resultantes de uma consulta em um banco de dados SQLite.

a) SQLiteOpenHelper
b) SQLiteStatement
c) SQLiteDatabase
d) ContentValues
e) Cursor

REFLEXÃO

O armazenamento persistente de informações é um recurso fundamental para os mais variados tipos de aplicativos. No cenário do desenvolvimento para dispositivos móveis, os aplicativos tem usado este tipo de armazenamento, porém podem enfrentar problemas em função das limitações de memória secundária dos aparelhos. Neste contexto, faça uma reflexão sobre a seguinte questão: Qual modelo de armazenamento é mais adequado para um dispositivo móvel: armazenamento local, utilizando o banco de dados SQLite, ou armazenamento na nuvem (cloud), em que os dados são transferidos para um servidor remoto?

LEITURA

Para complementar os conhecimentos apresentados até este momento é recomendada a leitura da obra Android em Ação. Este livro possui um conteúdo bastante detalhado a respeito do desenvolvimento para Android, e aborda conceitos como a comunicação via bluetooth e a escrita de aplicativos utilizando a linguagem C. A referência completa da obra é a seguinte:

ABLESON,W.F.; SEM, R.; KING, C.; ORTIZ, C.E. Android em Ação. Rio de Janeiro: Campus, 2012.

Referências

DATATYPES. SQLite 3 Data Types. Disponível em < http://www.sqlite.org/datatype3.html>, acesso em dezembro de 2014.

SQLITE. SQLite. Disponível em <http://sqlite.org/>, acesso em dezembro de 2014.

GABARITO

Capítulo 1

01. b) BlackBerry

02. d) Java ME

03. e) 4.4

Capítulo 2

01. b) EditText

02. d) Content Providers

03. e) Services

Capítulo 3

01. e) RadioButton

02. c) OnItemChecked

03. d) Spinner

Capítulo 4

01. c) putExtra

02. d) onBackPressed

03. d) Intent it = new Intent(this,Atividade2.class)

Capítulo 5

01. c) REAL

02. c) SQLiteDatabase

03. e) Cursor

ANOTAÇÕES

ANOTAÇÕES

ANOTAÇÕES

ANOTAÇÕES

ANOTAÇÕES

ANOTAÇÕES