

DESENVOLVIMENTO ANDROID

BRASÍLIA-DF.

Elaboração

Henrique Poyatos

Produção

Equipe Técnica de Avaliação, Revisão Linguística e Editoração

Sumário

APRESENTAÇÃO.....	5
ORGANIZAÇÃO DO CADERNO DE ESTUDOS E PESQUISA	6
INTRODUÇÃO.....	8
UNIDADE I	
INTRODUÇÃO	11
CAPÍTULO 1	
POR QUE ANDROID?.....	11
CAPÍTULO 2	
AMBIENTE DE DESENVOLVIMENTO.....	15
UNIDADE I	
MÃOS À OBRA!	20
CAPÍTULO 1	
EXEMPLO “ALÔ MUNDO”	20
CAPÍTULO 2	
ANDROIDMANIFEST	31
CAPÍTULO 3	
ACTIVITIES	33
CAPÍTULO 4	
SERVIÇOS	35
CAPÍTULO 5	
TAREFAS.....	38
CAPÍTULO 6	
VIEWS	41
CAPÍTULO 7	
LAYOUT	47
CAPÍTULO 8	
EXEMPLO MÉDIA ALUNO	50
CAPÍTULO 9	
EXEMPLO CÂMERA	68

CAPÍTULO 10	
EXEMPLO ALARME	74
UNIDADE III	
TESTANDO E INSTALANDO	88
CAPÍTULO 1	
TESTANDO EM UM DISPOSITIVO REAL	88
CAPÍTULO 2	
GERANDO UM PACOTE DE INSTALAÇÃO DA APP.....	90
PARA (NÃO) FINALIZAR.....	94
REFERÊNCIAS	95

Apresentação

Caro aluno

A proposta editorial deste Caderno de Estudos e Pesquisa reúne elementos que se entendem necessários para o desenvolvimento do estudo com segurança e qualidade. Caracteriza-se pela atualidade, dinâmica e pertinência de seu conteúdo, bem como pela interatividade e modernidade de sua estrutura formal, adequadas à metodologia da Educação a Distância – EaD.

Pretende-se, com este material, levá-lo à reflexão e à compreensão da pluralidade dos conhecimentos a serem oferecidos, possibilitando-lhe ampliar conceitos específicos da área e atuar de forma competente e consciente, como convém ao profissional que busca a formação continuada para vencer os desafios que a evolução científico-tecnológica impõe ao mundo contemporâneo.

Elaborou-se a presente publicação com a intenção de torná-la subsídio valioso, de modo a facilitar sua caminhada na trajetória a ser percorrida tanto na vida pessoal quanto na profissional. Utilize-a como instrumento para seu sucesso na carreira.

Conselho Editorial

Organização do Caderno de Estudos e Pesquisa

Para facilitar seu estudo, os conteúdos são organizados em unidades, subdivididas em capítulos, de forma didática, objetiva e coerente. Eles serão abordados por meio de textos básicos, com questões para reflexão, entre outros recursos editoriais que visam a tornar sua leitura mais agradável. Ao final, serão indicadas, também, fontes de consulta, para aprofundar os estudos com leituras e pesquisas complementares.

A seguir, uma breve descrição dos ícones utilizados na organização dos Cadernos de Estudos e Pesquisa.



Provocação

Textos que buscam instigar o aluno a refletir sobre determinado assunto antes mesmo de iniciar sua leitura ou após algum trecho pertinente para o autor conteudista.



Para refletir

Questões inseridas no decorrer do estudo a fim de que o aluno faça uma pausa e reflita sobre o conteúdo estudado ou temas que o ajudem em seu raciocínio. É importante que ele verifique seus conhecimentos, suas experiências e seus sentimentos. As reflexões são o ponto de partida para a construção de suas conclusões.



Sugestão de estudo complementar

Sugestões de leituras adicionais, filmes e sites para aprofundamento do estudo, discussões em fóruns ou encontros presenciais quando for o caso.



Praticando

Sugestão de atividades, no decorrer das leituras, com o objetivo didático de fortalecer o processo de aprendizagem do aluno.



Atenção

Chamadas para alertar detalhes/tópicos importantes que contribuem para a síntese/conclusão do assunto abordado.



Saiba mais

Informações complementares para elucidar a construção das sínteses/conclusões sobre o assunto abordado.



Sintetizando

Trecho que busca resumir informações relevantes do conteúdo, facilitando o entendimento pelo aluno sobre trechos mais complexos.



Exercício de fixação

Atividades que buscam reforçar a assimilação e fixação dos períodos que o autor/conteudista achar mais relevante em relação a aprendizagem de seu módulo (não há registro de menção).



Avaliação Final

Questionário com 10 questões objetivas, baseadas nos objetivos do curso, que visam verificar a aprendizagem do curso (há registro de menção). É a única atividade do curso que vale nota, ou seja, é a atividade que o aluno fará para saber se pode ou não receber a certificação.



Para (não) finalizar

Texto integrador, ao final do módulo, que motiva o aluno a continuar a aprendizagem ou estimula ponderações complementares sobre o módulo estudado.

Introdução

Olá, seja bem-vindo ao curso de introdução à programação em sistema operacional Android. Neste caderno de estudos, serão apresentadas as diversas razões, o número de participação de mercado da plataforma, além das vantagens e a facilidade de se desenvolver para Android. Por meio de um simples passo-a-passo, será abordado como instalar e configurar um ambiente completo para desenvolvimento, que inclui Frameworks, IDE e emuladores.

Além disso, será apresentado como criar um projeto do tipo Android e rodar um projeto pela primeira vez, com um exemplo mais simples possível. Será exposto, também do que é composto um projeto Android? O que é o manifesto e sua importância, o que são atividades e como elas funcionam? A importância das Notifications, Services e Tasks.

Compreender como um layout de aplicação pode ser feito e quais as possibilidades de Views: Existem vários tipos de layout para Android, cada um deles adequado à uma situação em específico. Saberemos quais são eles, quando usar e como serão exibidos;

Para facilitar a compreensão, serão apresentados três projetos passo a passo fáceis de se acompanhar, agora com uns exemplos mais elaborados, modificando o layout, criando um formulário, fazendo uma atividade se relacionar com outra, criando um alarme, gerando notificações, acionando a câmera. Além disso, será necessário testar sua aplicação em ambiente virtualizado ou ambiente real: serão abordados tópicos e possibilidades de testar sua aplicação em um dispositivo real.

Por fim, será abordado como é possível gerar um pacote de instalação e efetuar a mesma em qualquer dispositivo com Android que atende os pré-requisitos impostos por nós mesmos.

Objetivos

- » Compreender as razões e vantagens de se desenvolver na plataforma Android.
- » Configurar um ambiente completo para desenvolvimento na plataforma.
- » Testar o ambiente com o famoso exemplo “Alô Mundo”.

- » Compreender o que compõe a arquitetura, como AndroidManifest, Activities, Notifications, Services, Tasks.
- » Compreender como um layout de aplicação pode ser feito e quais as possibilidades de Views.
- » Criar um exemplo funcional de aplicação.
- » Testar sua aplicação em ambiente virtualizado ou ambiente real.
- » Gerar um pacote para instalação de seu aplicativo.

CAPÍTULO 1

Por que Android?

O que é Android?

É um sistema operacional concebido para ser utilizado em dispositivos móveis, como smartphones e tablets, baseado em Linux. Seu desenvolvimento é mantido pela OpenHandset Alliance, uma aliança de várias empresas com a intuito de criar padrões abertos para telefonia móvel. Entre as várias empresas parte da aliança temos Google, HTC, LG, Dell, Intel, Qualcomm, Texas Instruments, Motorola, Samsung, T-Mobile e Nvidia.

O sistema é baseado em licenças abertas de software, tendo boa parte do sistema operacional Android utilizando licença Apache 2.0, enquanto o Kernel Linux distribuído utiliza licença GPL 2.

Figura 1. Logo do Android.



Disponível em <http://pt.wikipedia.org/wiki/Ficheiro:Android_robot.svg>. Acesso em: 9 mar. 2014.

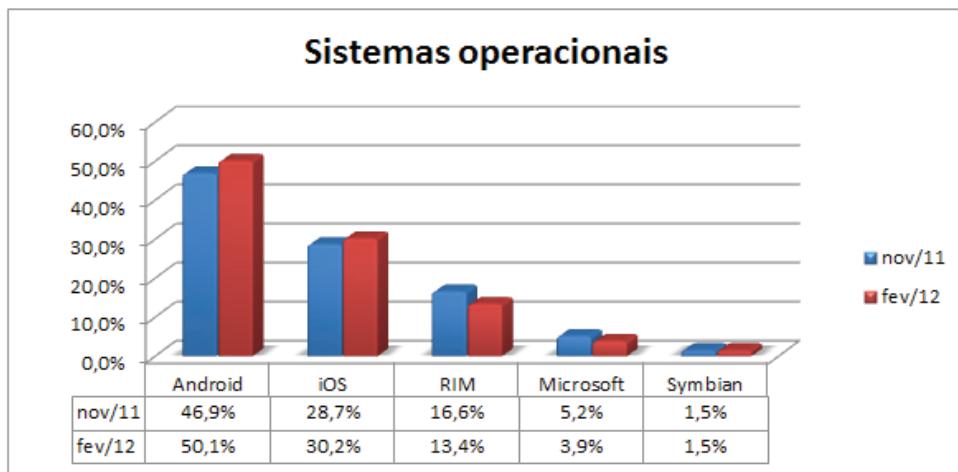
Por que Android?

Mercado

Justamente por ser baseadas em licenças e padrões abertos, o Sistema Operacional – SO é escolhido para dar suporte a centenas de dispositivos de diferentes fabricantes. Por

esta razão, estima-se que o Android é o sistema operacional para dispositivos móveis mais utilizado do mundo, possuindo 50,3% deste mercado, de acordo com números da comScore, em Fevereiro de 2012. Segundo pesquisa da StrategyAnalytics em 2013, a cada cinco smartphones comercializados, quatro possuem Android. Sua loja virtual de aplicativos, a Google Play, já ultrapassara de 1 milhão de aplicativos e jogos.

Figura 2. Marketshare dos sistemas operacionais para dispositivos móveis segundo comScore.



Fonte: Adaptado, originalmente em <<http://tecnoblog.net/wp-content/uploads/2012/04/market-share-android.png>>. Acesso em: 9 mar. 2014.

Desenvolvimento

Além de ter a maior fatia de mercado, existem outras boas razões para se escolher o Android como base para desenvolvimento de aplicações. Embora possam ser usadas outras linguagens para desenvolvimento, a mais utilizada é o Java, linguagem consagrada e uma das mais utilizadas no mundo todo. Java também é muito conhecida pela sua portabilidade, característica que se alinha completamente com os objetivos do SO Android.

As aplicações escritas em Java são compiladas em bytecode em um formato conhecido como Dalvik e são executadas usando a Máquina Virtual Dalvik – MVD, um dos componentes do Android. Trata-se de uma máquina virtual especializada, desenvolvida para uso em dispositivos móveis, permitindo que programas sejam em bytecode e possam ser executados em qualquer dispositivo Android, independentemente do processador utilizado, tornando os aplicativos mais portáveis, independentes do hardware e até mesmo versão de Android utilizada, bastando ao desenvolvedor se preocupar com a aderência à versão do MVD utilizada.

Outro fator que torna a plataforma atraente é a possibilidade de se adquirir todos os softwares para seu desenvolvimento de forma gratuita – Máquinas Virtuais, SDKs, IDEs

e até mesmo emuladores podem ser facilmente encontrados e baixados sem custo da Internet, enquanto SOs concorrentes, como iOS e Windows Phone, exigem a aquisição de licenças para suas IDEs.

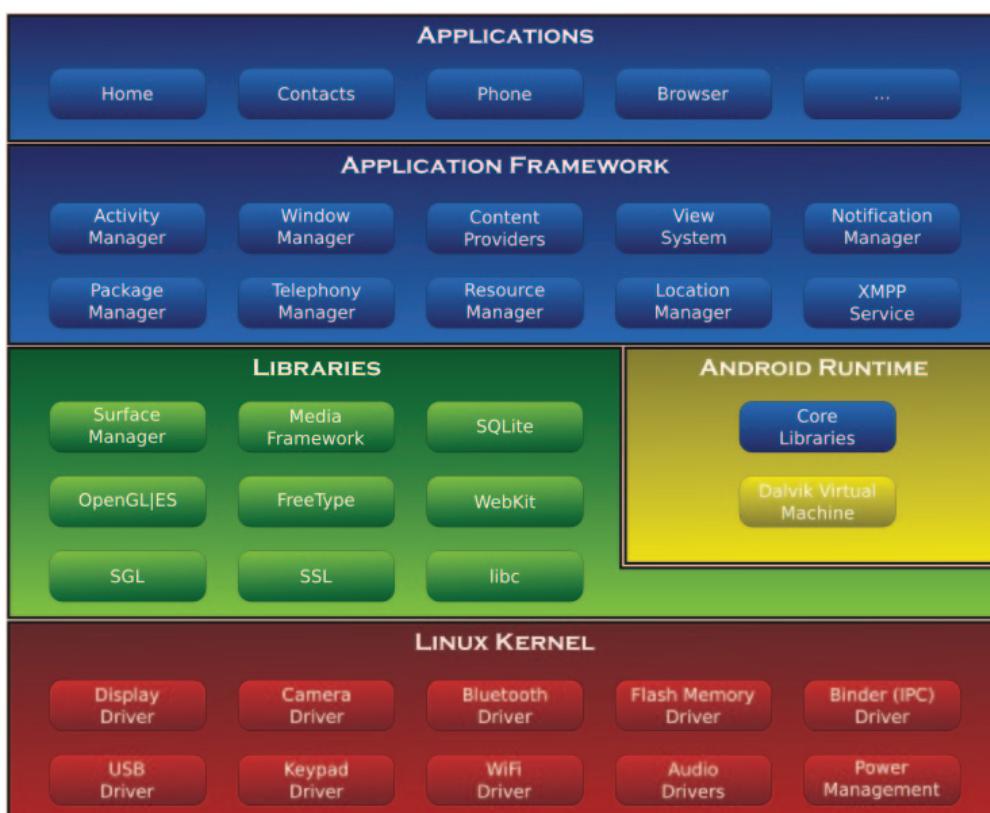
Frameworks e bibliotecas poderosas

O sistema operacional Android, aliado à Máquina Virtual Dalvik, traz bibliotecas em C/C++ e Frameworks Java para desenvolvimento de aplicados que, além de padronizar o acesso aos recursos do hardware, facilitam e automatizam uma série de procedimentos, tornando o desenvolvimento de aplicações mais produtivo e simples.

Arquitetura de sistema

Em qualquer projeto de desenvolvimento, é importante ter conhecimento da arquitetura do sistema operacional base. Desta maneira, podemos saber quais recursos estão disponíveis, em qual camada está e o que pode ser feito para acessá-lo. As camadas mais altas ficam mais longe do dispositivo físico, enquanto as camadas mais baixas são as mais próximas.

Figura 3. Arquitetura do sistema operacional Android.



Fonte: Originalmente em <<http://pt.wikipedia.org/wiki/Ficheiro:Android-System-Architecture.svg>>. Acesso em: 9 mar. 2014.

- » **Aplicações (Applications):** camada mais alta, onde ficam os softwares do telefone, contatos, navegador web, aplicações comerciais e jogos.
- » **Framework de Aplicação (Application Framework):** componentes de Framework que servem as aplicações, intermediando e gerenciando as requisições delas para as bibliotecas e AndroidRuntime.
- » **Bibliotecas (Libraries):** utilizadas pelo Framework de Aplicação, feitas em linguagens de níveis mais baixos (C e C++), possuem recursos de renderização 3D, banco de dados e multimídia, facilitando o desenvolvimento e intermediando os recursos disponibilizados pelo core do sistema (Linux Kernel).
- » **AndroidRuntime:** camada responsável por executar arquivos DEX (DalvikExecutable) das aplicações, traduzindo os bytecodesDalvik para instruções nativas compreendidas pelo Linux Kernel, tornando possível a portabilidade.
- » **Linux Kernel:** trata-se da última camada, a mais próxima do hardware. Trata-se de uma customização do kernel do Linux, na versão 2.6, e tem como responsabilidade o gerenciamento dos recursos físicos (hardware).

CAPÍTULO 2

Ambiente de desenvolvimento

Neste capítulo, veremos, passo a passo, como montar um ambiente para o desenvolvimento de seus aplicativos em Android.

Eclipse AndroidDevelopment – ADT

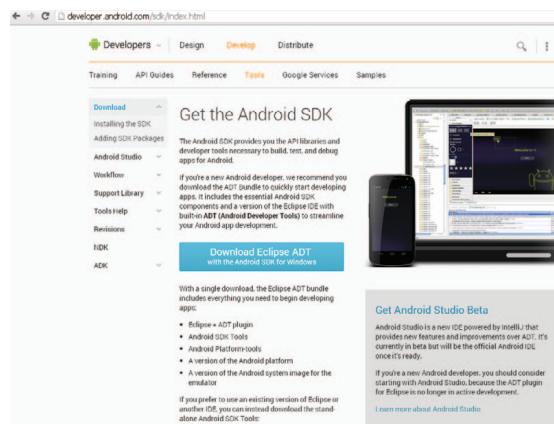
Existem vários ambientes de desenvolvimento que podem ser configurados e utilizados para desenvolvimento de aplicações em Android, como o MIT App Inventor (<http://appinventor.mit.edu/explore/>) ou o Phonegap/Apache Cordova (<http://phonegap.com/>), mas nenhum deles é tão popular quanto o uso da Integrated Development Environment – IDE, conhecida como Eclipse, extremamente popular para projetos utilizando Java como linguagem de desenvolvimento.

A IDE Eclipse pode ser expandida por meio de um recurso conhecido como plugins, que possibilitam que a ferramenta trabalhe com uma série de bibliotecas diferentes ou mesmo linguagens diferentes (como o PHP).

O plugin ADT (sigla para Android Development Tools ou ferramentas de desenvolvimento Android) cria um ambiente integrado, expandindo o Eclipse e o possibilitando a criar projetos em Android.

Embora uma versão atualizada do Eclipse possa receber o plugin ADT normalmente, neste tutorial faremos download e instalaremos uma versão da IDE que já vem com o ADT instalado, integrado com o Android SDK, que falaremos adiante.

Figura 4. Para baixar o Eclipse ADT.

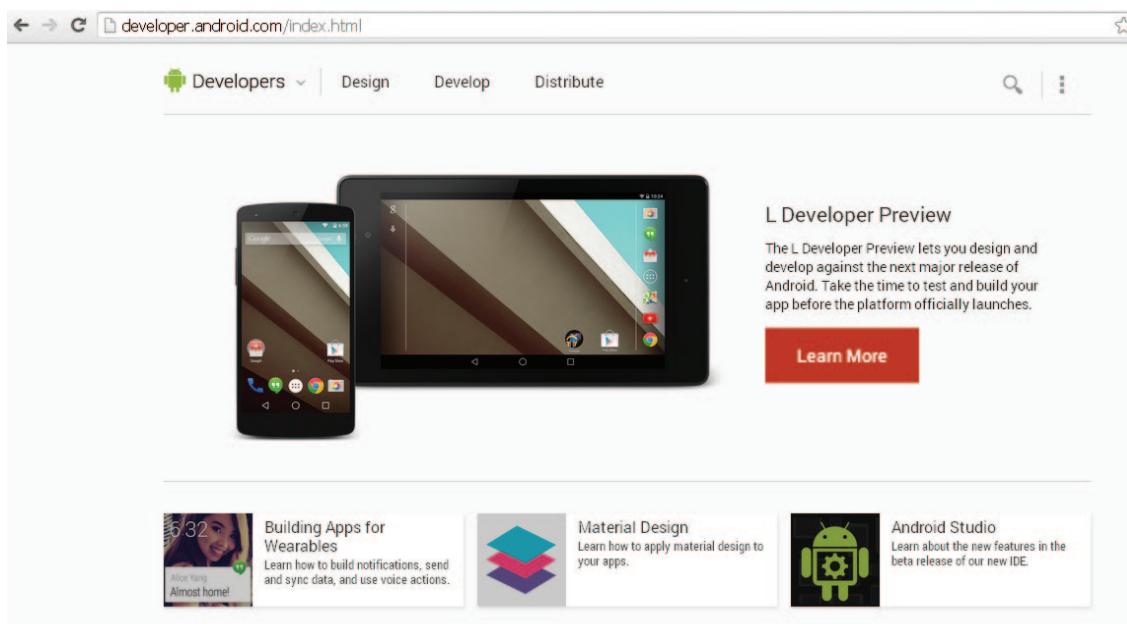


Fonte: <<http://developer.android.com/sdk/index.html>>. Acesso em: 17 ago. 2014.

Acesse o site oficial de desenvolvimento Android (<http://developer.android.com/>):

1. Clique no menu “Develop”, depois no submenu “Tools”, e será enviado para a página que possibilita baixar o Eclipse (Juno) com o plugin ADT e o Android SDK:

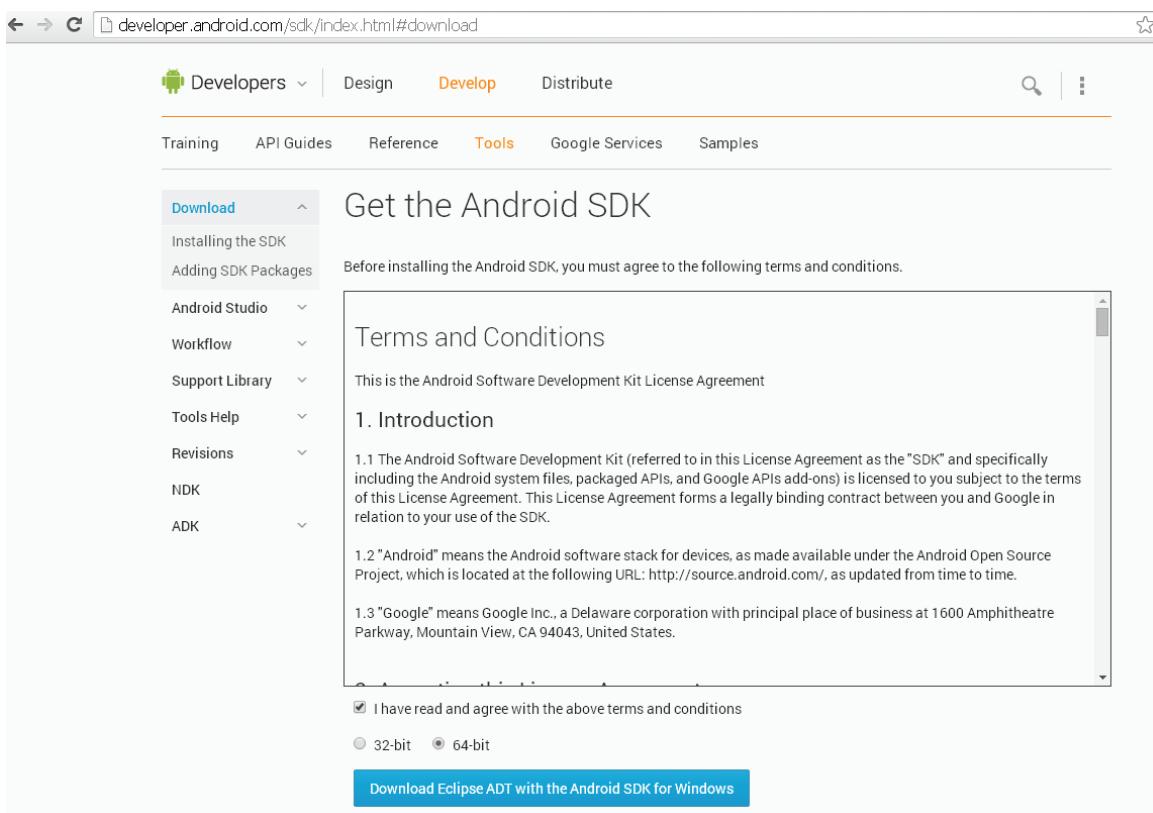
Figura 5. O site oficial de desenvolvimento Android.



Fonte: <<http://developer.android.com/>>. Acesso em: 17 ago. 2014.

2. Clique no botão “Download Eclipse ADT with Android SDK for Windows”. Caso esteja em outro sistema operacional como MacOS ou Linux, o site automaticamente o detecta, mudando a opção para download. Sinta-se à vontade para utilizar qualquer sistema operacional, embora alguns procedimentos sejam ligeiramente diferentes.
3. Aceite o contrato de termos de condições que aparece na sequência, marcando “I havereadandagreewiththeabovetermsandconditions” e escolha entre as versões de 32 e 64 bits. Caso seu sistema operacional seja uma versão recente, há uma grande chance que seja 64 bits. Clique no botão “Download the Eclipse ADT with the Android SDK for Windows” e o download começará.

Figura 6. Contrato de termos e condições que precede o download do Eclipse ADT.



Fonte: <<http://developer.android.com/sdk/index.html#download>>. Acessado em: 17 ago. 2014.

4. Após o download do arquivo em formato zip (de algumas centenas de megabytes), descompacte-a no diretório de sua preferência (Arquivos de Programas é uma sugestão). Embora seja opcional, recomendamos a criação de um atalho (no Windows, com o botão direito do mouse em “Criar Atalho”) do principal executável do pacote, eclipse.exe, presente em uma subpasta. Arraste o atalho para a área de trabalho para sua comodidade.

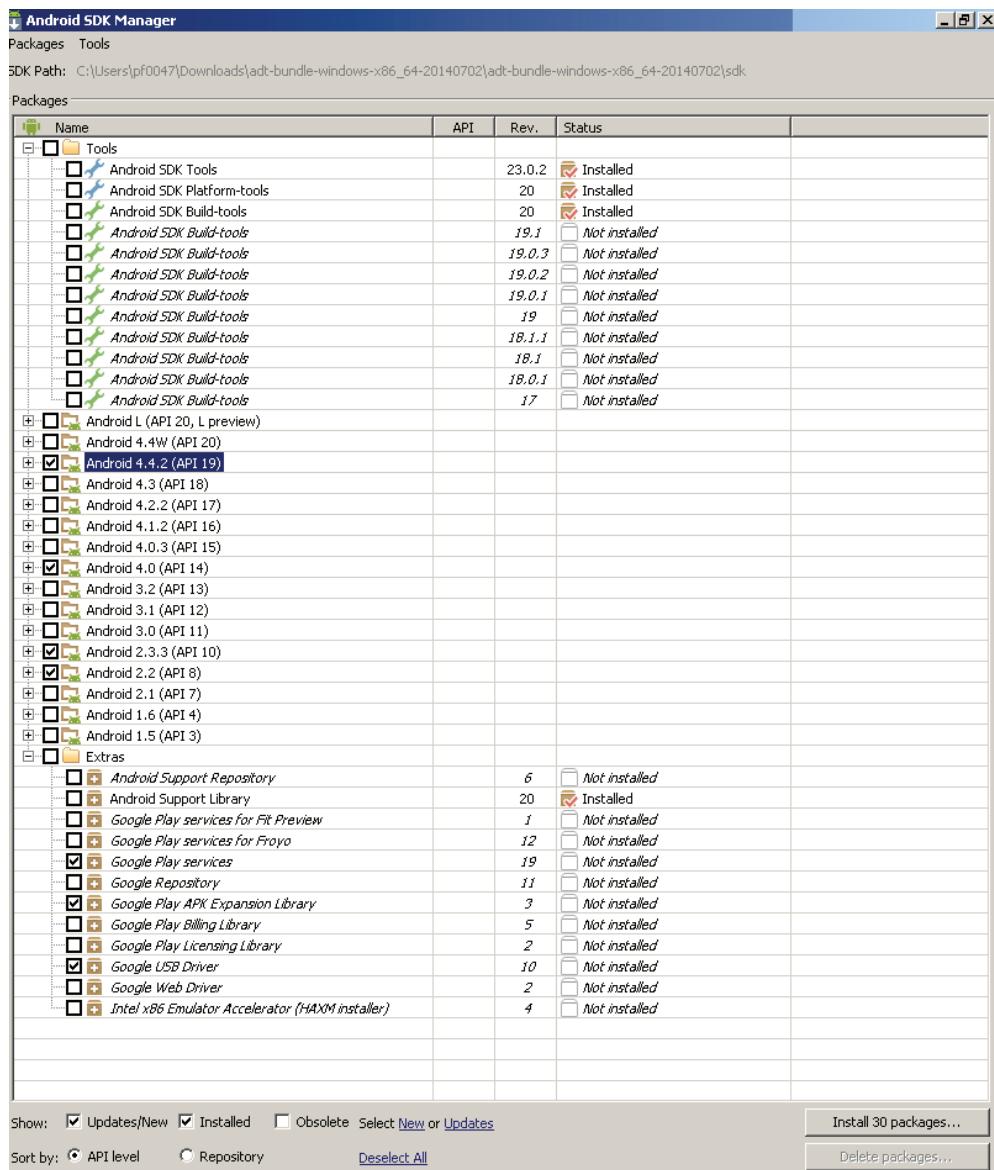
Android Software Development Kit – SDK

O Android SDK (ou Kit de Desenvolvimento de Software) contém uma série de bibliotecas de código, depurador, emulador do sistema operacional Android baseado no virtualizador QEMU, documentação, códigos de exemplo e tutoriais. Tudo isso presente na pasta “SDK”, que possui um executável chamado “Android SDK Manager”.

Por meio deste Manager, é possível visualizar as mais diversas versões de API, da versão 3 (Android 1.5) até a versão 20 (Android 4.4W). É recomendável a instalação de versões como Android 2.2 (Froyo) ou ao menos a versão 2.3.3 (Gingerbread), 4.0 (Ice Cream Sandwich) e a versão 4.1 (JellyBean), conforme explicaremos adiante.

Marque também a instalação do Google Play Services, Google Play APK Expansion Library e Google USB Driver, que serão requeridos em projetos mais robustos.

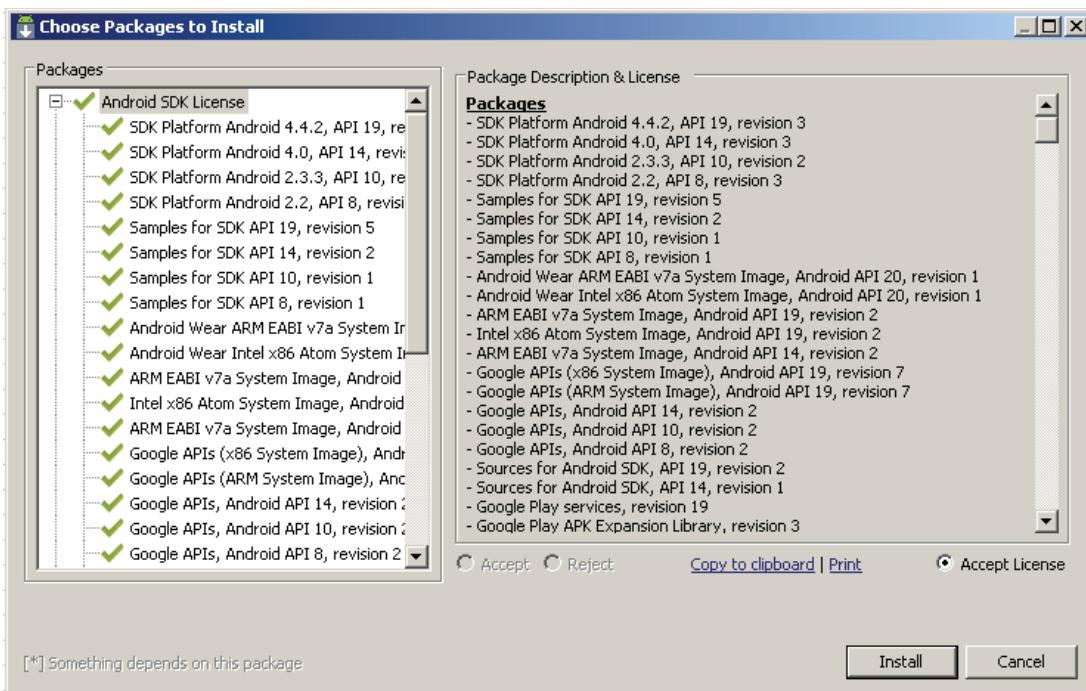
Figura 7. Tela do Android SDK Manager.



Fonte: próprio autor.

Pressione o botão de “Install x packages...” para realizar o download e instalação destes recursos. Aceite eventuais licenças para que possa prosseguir.

Figura 8. Aceitando os certificados na instalação da API.



Fonte: próprio autor.

Após os downloads, o ambiente estará pronto para desenvolvimento.

CAPÍTULO 1

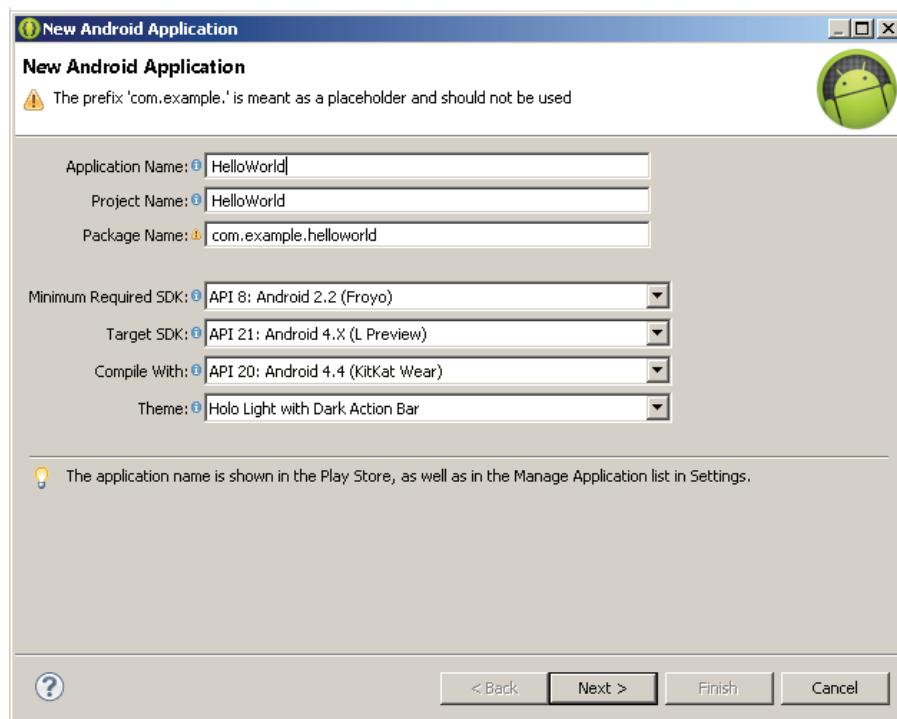
Exemplo “Alô Mundo”

Criando o projeto

Começaremos criando nosso primeiro projeto, o famoso exemplo de “Alô Mundo”, assim testamos as estruturas básicas de um projeto para Android.

- » Abra o Eclipse ADT (Juno). Informe a área de trabalho (workspace) que julgar mais adequado. Clique em “File”, “New” e “Android Application”. Um assistente como abaixo se fará disponível:

Figura 9. Tela de criação de uma aplicação Android.



Fonte: próprio autor.

- » Preencha com as informações do projeto.
 - › Em “Application Name”, insira o nome de sua aplicação, em nosso caso, “Hello World”.
 - › Em Project Name, complete com o nome do projeto, que geralmente segue o nome da aplicação.
 - › No campo “Minimum Required SDK”, deve ser informada qual é a API mínima para rodar a aplicação. Trata-se de uma decisão importante. Quanto mais antiga for a API, maior é a suportabilidade de sua aplicação. Se for informada aqui a versão 2.2 (Froyo), versão 8 da API, sua aplicação, com os devidos cuidados no desenvolvimento, rodará em Android versões 2.2 até as versões mais recentes, como a 4.4. Por outro lado, a cada nova versão da API, novos recursos e facilidades são implementadas, e estão abrindo mão em prol de uma maior abrangência de mercado.
 - › Para ajudar em sua decisão, segue no quadro 1 alguns números da participação de mercado mundial das versões de Android, números apurados em agosto de 2014.

Quadro 1. Participação de mercado mundial por versão de sistema operacional Android.

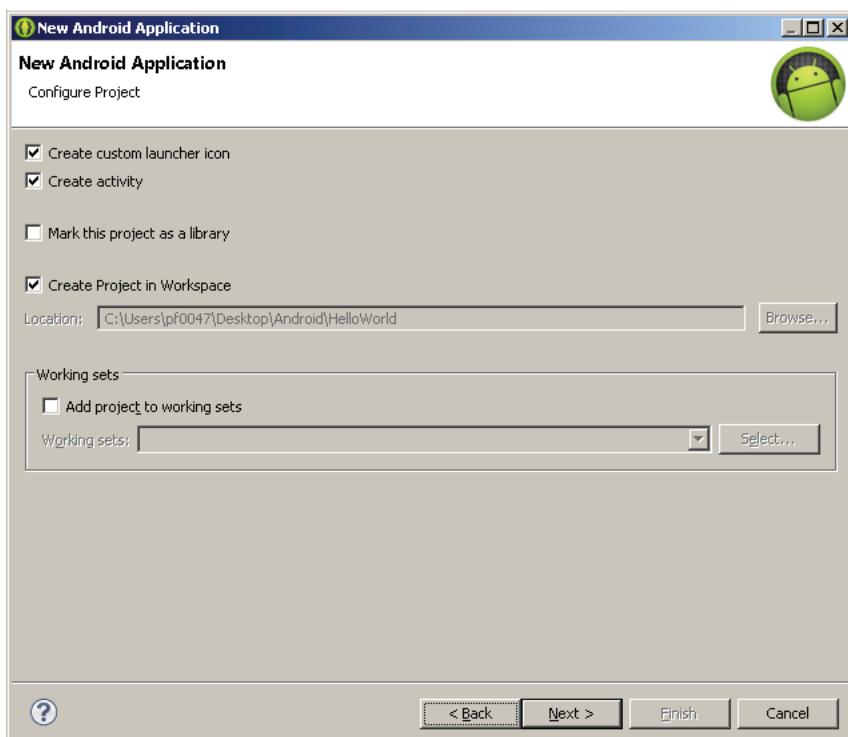
VERSÃO	CODINOME	API	PARTICIPAÇÃO
2.2	Froyo	8	0,7%
2.3.3 - 2.3.7	Gingerbread	10	13,6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	10,6%
4.1.x	JellyBean	16	26,5%
4.2.x		17	19,8%
4.3		18	7,9%
4.4	KitKat	19	20,9%

Fonte: <[site http://developer.android.com/](http://developer.android.com/)>. Acesso em: 11 set. 2014.

1. Para efeito de exercício, selecionaremos a API 8 (Android 2.2, codinome Froyo).
 - › Em Target SDK, informar a API mais alta que sua aplicação suportará. É ideal informar, como máximo, a última versão estável lançada.
 - › Em Compile With, trata-se da API que será utilizada na compilação.
 - › Em Theme, é possível escolher qual o tema que sua aplicação utilizará para seu visual.
2. Pressione o botão “Next”.

3. Na segunda tela de assistente, marque a opção “Create custom launcher icon”, que lhe permitirá personalizar o ícone que representa o aplicativo.
4. A opção “Create activity” deve permanecer marcada pois o aplicativo será configurado para apresentar uma atividade principal (Main Activity).

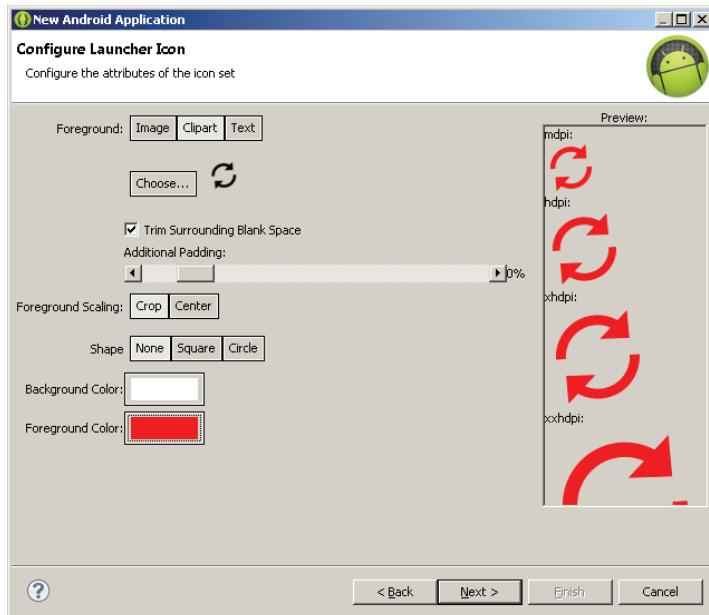
Figura 10. Segunda tela na criação do aplicativo Android.



Fonte: próprio autor.

5. Clique no botão “Next”.
6. A terceira tela do assistente permite a personalização o ícone do aplicativo. Pode ser utilizada uma imagem personalizada (botão “Image”), um texto (botão “Text”) ou é possível utilizar um dos ícones disponíveis no Clipart, pressionando seu botão, conforme mostrado no exemplo. Existe a possibilidade de cortar, centralizar, mudar sua cor, ou seja, algumas manipulações básicas de imagem são permitidas.
 - » Repare a necessidade de se criar quatro versões do ícone em vários tamanhos diferentes. Isso se deve a grande possibilidade de dispositivos suportados, que vão desde relógios de pulso até televisores. Os quatro tamanhos apresentados aqui garantirão uma boa visualização em maioria dos dispositivos suportados. Caso o dispositivo for maior ou menor do que os previstos por estas quatro versões, o ícone mais próximo será dimensionado, o que pode resultar em uma visualização ruim.

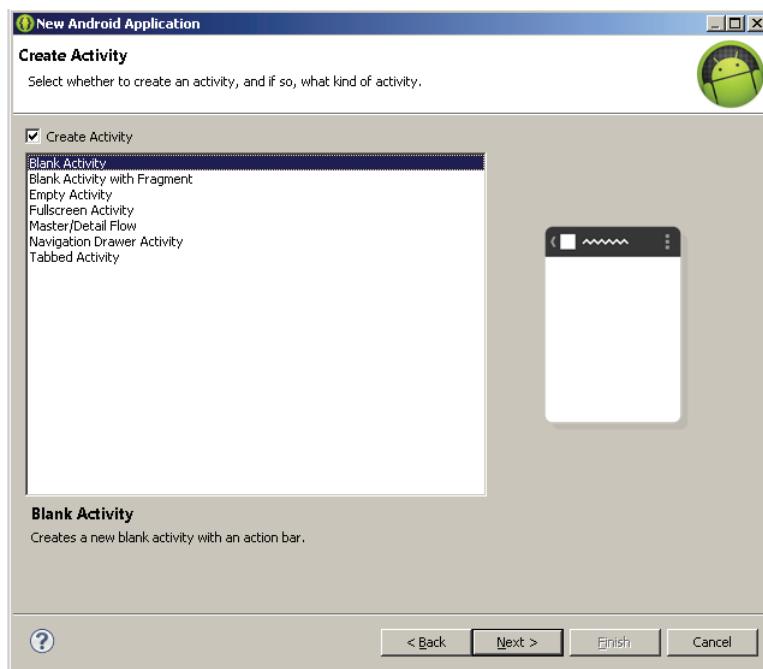
Figura 11. Personalizando o ícone do aplicativo.



Fonte: próprio autor.

7. Pressione o botão “Next”.
8. Neste trecho do assistente, podemos escolher qual o tipo da atividade principal queremos utilizar.

Figura 12. Escolha do tipo de Activity principal.

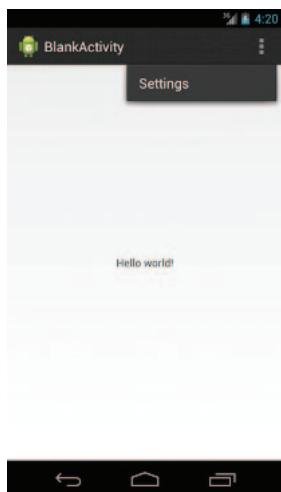


Fonte: próprio autor.

Activity ou atividade é cada camada visual da aplicação onde estão contidos os controles de interface (botões, caixas de texto etc.) chamados de Views. Os tipos disponíveis são:

1. **BlankActivity:** trata-se de um template em branco. Recomendado para criar um aplicativo básico ou como o ponto de início de um projeto. Possui Title Bar (conhecido como ActionBar a partir do Android 3.0), menu de opções (conhecido como Action Overflow a partir do Android 3.0) e um layout básico.

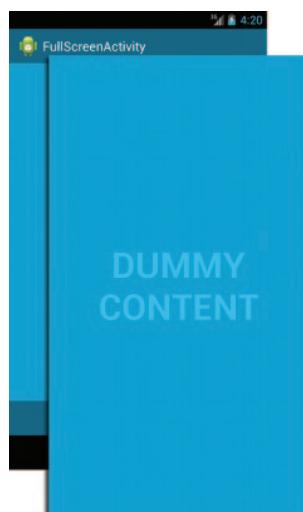
Figura 13. Template para BlankActivity.



Fonte: Originalmente em: <<https://developer.android.com/tools/projects/templates.html#blank-activity>>. Acesso em: 17 ago. 2014.

2. **BlankActivitywithFragment:** trata-se do template em branco padrão, mas com fragments.
3. **FullscreenActivity:** uma atividade que, por padrão, abre em tela cheia (ideal para jogos).

Figura 14. Template para FullScreenActivity



Fonte: Originalmente em: <<https://developer.android.com/tools/projects/templates.html#full-screen-activity>>. Acessado em: 17 ago. 2014.

- 4. Master/DetailFlow:** este template cria uma layout para uma lista de itens, podendo cada um destes itens possuir seus subitens.

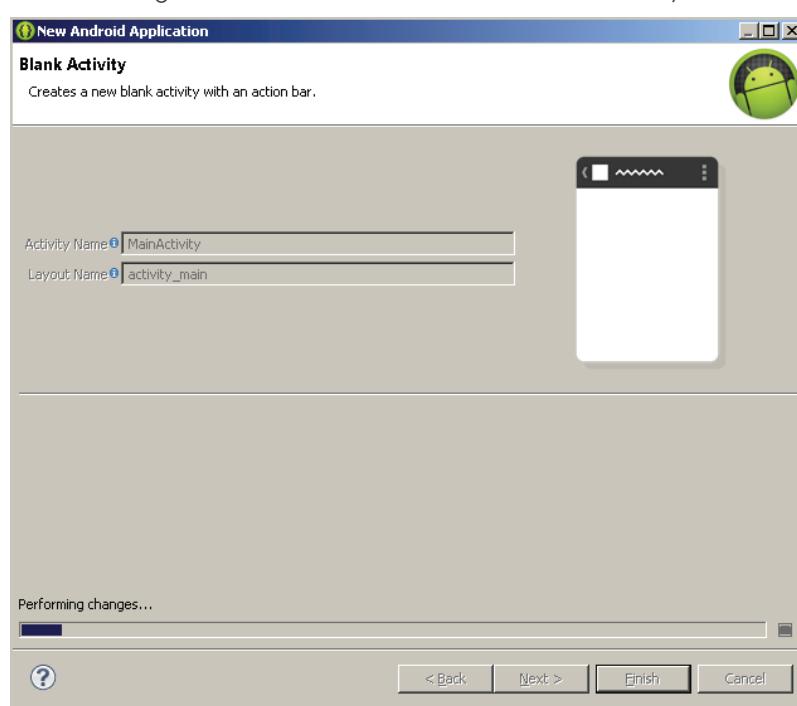
Figura 15. Template para Master/DetailFlow.



Fonte: Originalmente em <<https://developer.android.com/tools/projects/templates.html#master-detail-activity>>. Acessado em: 17 ago. 2014.

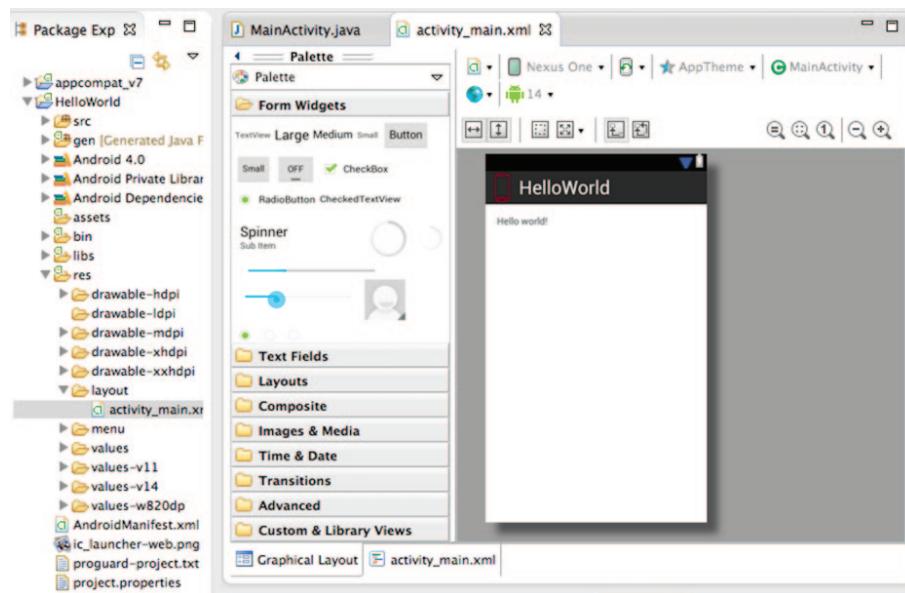
- » Pressione o botão “Next”.
- » Na última tela que abordaremos, é possível renomear a activity. Manteremos o mesmo nome, clicando em “Finish”. Desta maneira, o projeto será criado.

Figura 16. Possibilidade de renomear a MainActivity



E temos um projeto criado com sucesso.

Figura 17. Projeto HelloWorld criado.



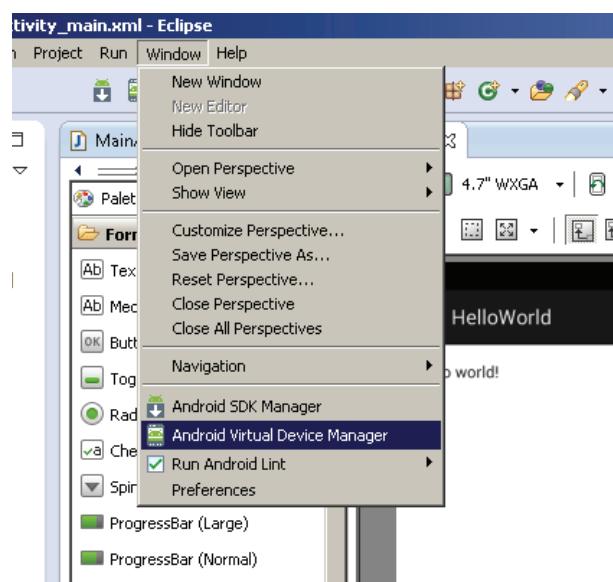
Fonte: próprio autor.

Criando um ambiente para testes

É necessário configurar um ambiente para testes, para que o aplicativo possa ser verificado durante desenvolvimento.

1. Clique no menu “Window”, opção “Android Virtual Device Manager”.

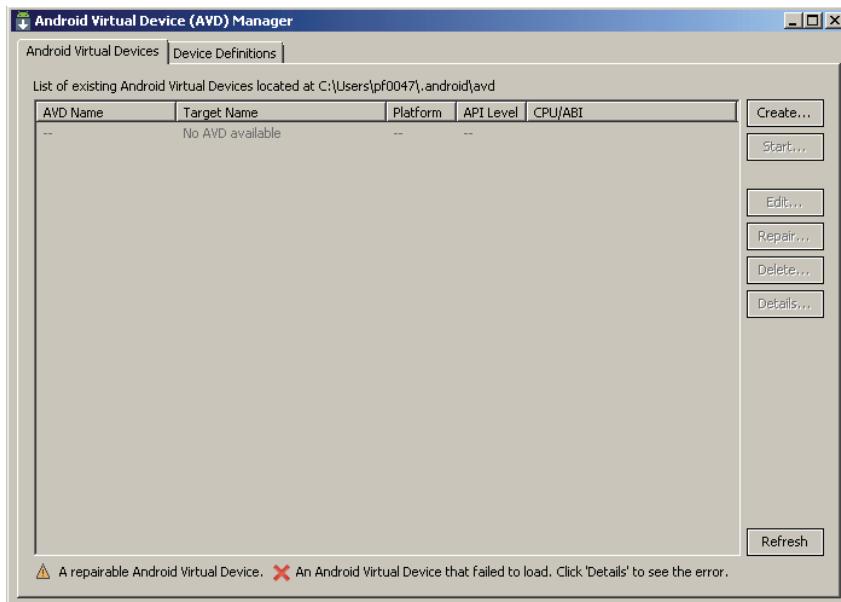
Figura 18. Acessando Android Virtual Device Manager.



Fonte: próprio autor.

2. Nenhuma AVD está criada inicialmente. Clique no botão “Create”.

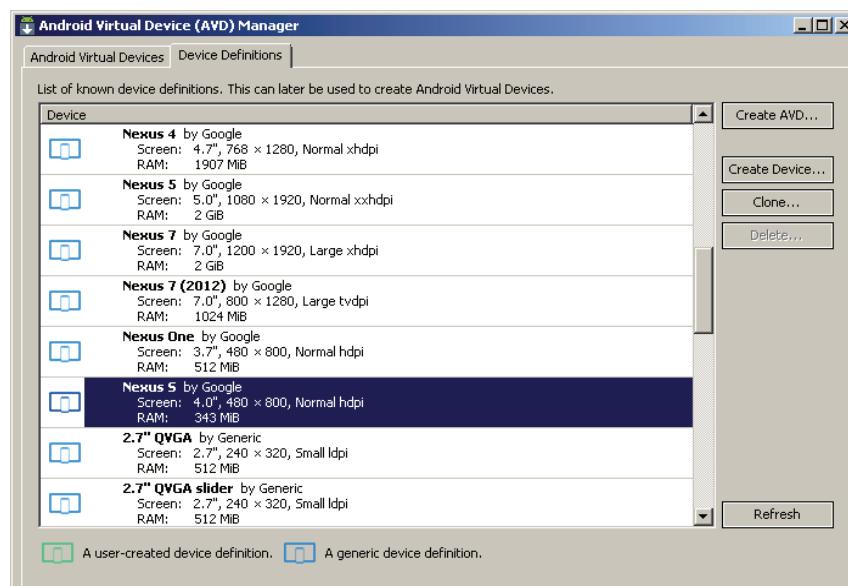
Figura 19. AVD Manager.



Fonte: Próprio autor.

3. Nenhuma AVD está criada inicialmente. Clique na aba “Device Definitions”.
4. Várias opções de dispositivos estão à disposição para criar um ambiente de testes. Escolha uma delas (usaremos o NexusOne, neste exemplo) e clique em “Create AVD”.

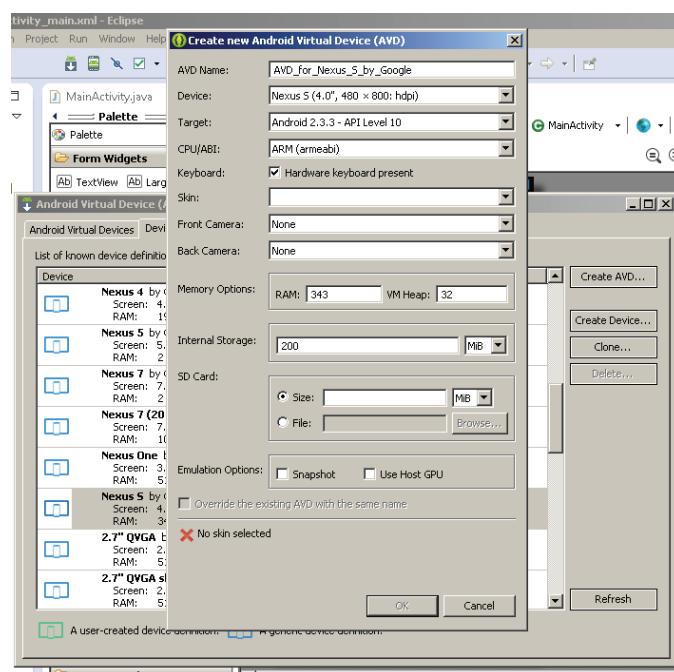
Figura 20. Escolhendo o dispositivo para emulação.



Fonte: Próprio autor.

- É possível personalizar o dispositivo a ser emulado, trocando o processador, memória, entre outros. Marque a opção “Use Host GPU” para uma melhor performance. Também é necessário escolher uma skin, mesmo que a opção seja “None” or “no skin”. Finalize clicando em “OK”.

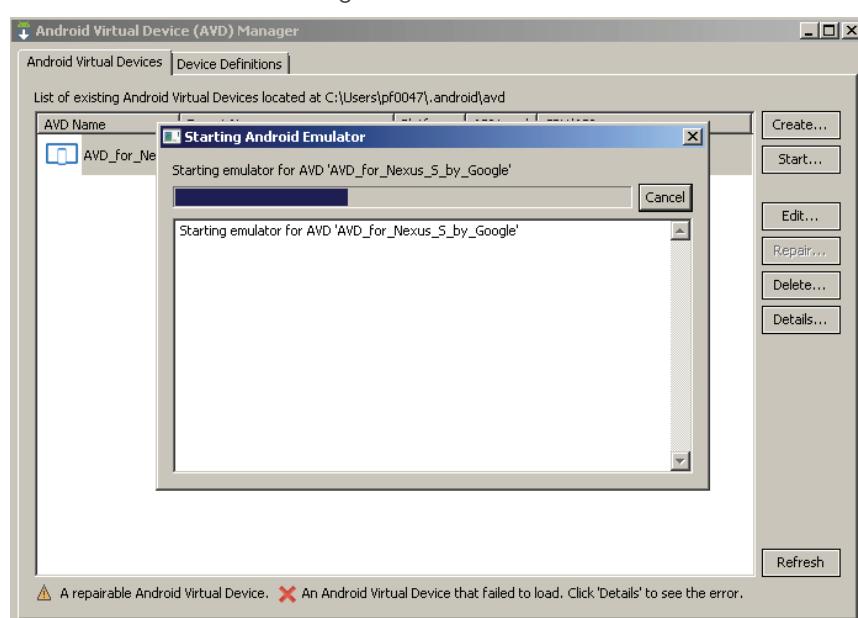
Figura 21. Personalizando AVD.



Fonte: Próprio autor.

- O AVD está criado. Clique no botão “Start”.

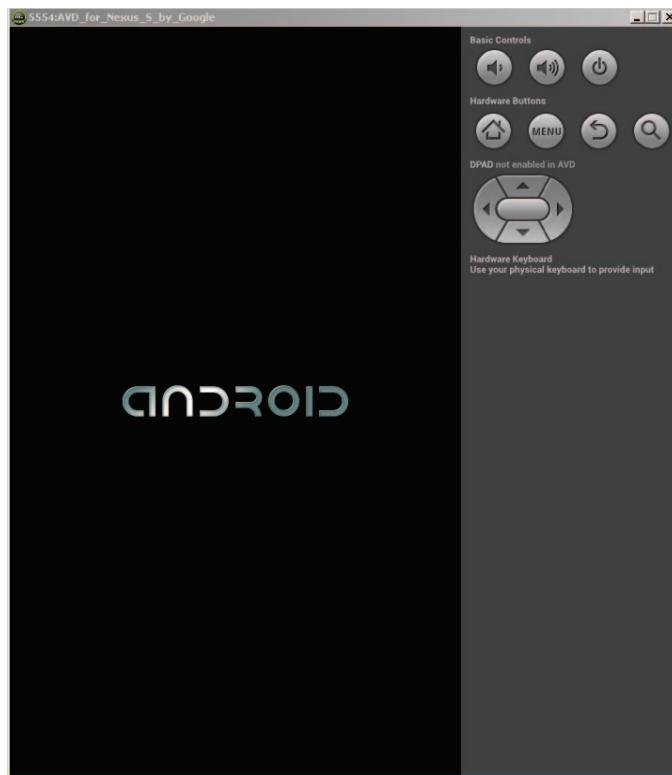
Figura 22. AVD criado.



Fonte: Próprio autor.

7. O emulador foi iniciado.

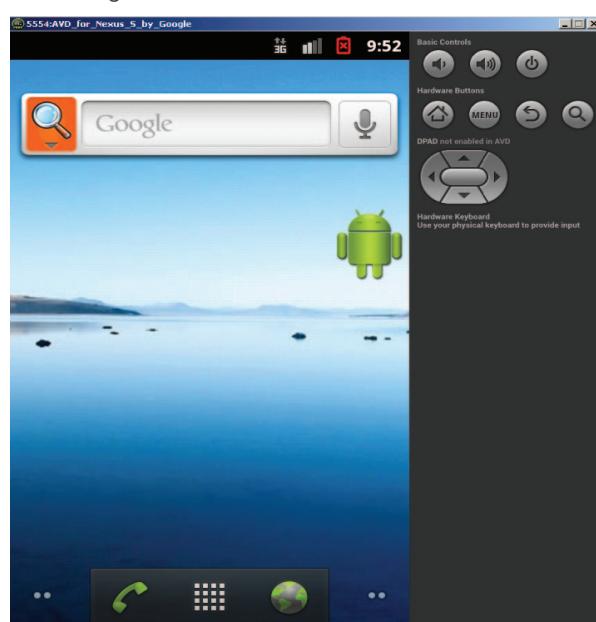
Figura 23. Emulador em inicializaçâo



Fonte: Próprio autor.

8. Ambiente pronto para testes.

Figura 24. Emulador em funcionamento.



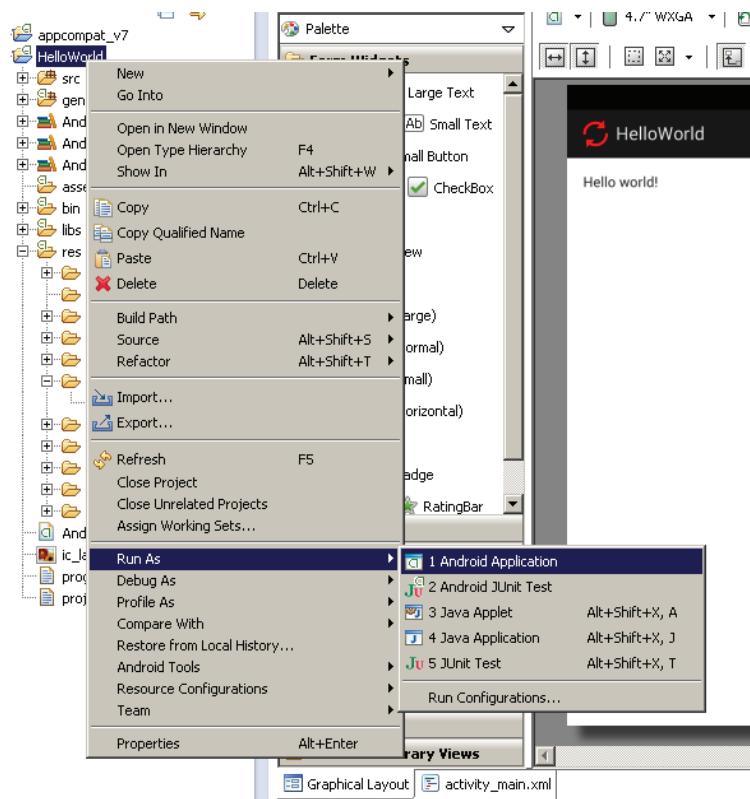
Fonte: Próprio autor.

Testando sua aplicação

Para testar sua aplicação:

1. Clique na pasta do projeto em “Project Explorer” (lado esquerdo) com o botão direito do mouse, selecionando a opção “Run As”, subopção “Android Application”.

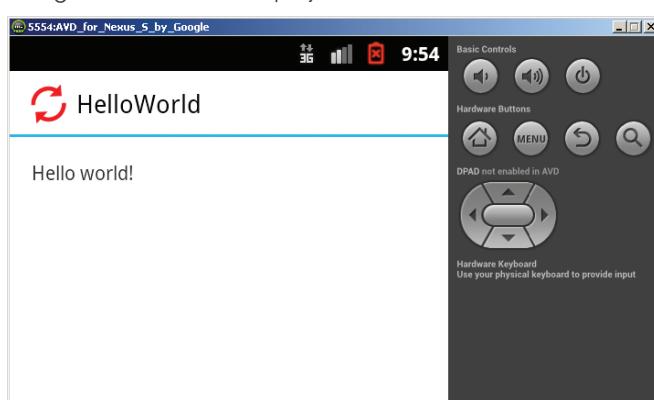
Figura 25. Enviando o projeto para o emulador.



Fonte: Próprio autor.

2. O projeto será enviado para o emulador para testes.

Figura 26. Rodando o projeto “HelloWorld” no emulador.



Fonte: Próprio autor.

CAPÍTULO 2

AndroidManifest

Entendo o manifesto

Toda aplicação para Android deve obrigatoriamente possuir um arquivo xml AndroidManifest.xml, exatamente com este nome, em sua pasta raiz. Ele provê informação essencial de sua aplicação ao sistema operacional Android, informações prévias que o sistema precisa obter antes de rodar quaisquer aplicações.

- » Nomeia o pacote Java da aplicação.
- » Declara qual é a API mínima para sua aplicação rodar.
- » Lista as bibliotecas que a aplicação precisa ser ligada.
- » Descreve os componentes de sua aplicação: suas atividades, serviços, provedores de conteúdo, enfim, tudo da qual sua aplicação é composta. É possível assim dizer ao sistema operacional sob quais condições a aplicação rodará.
- » Declara quais permissões a aplicação precisa ter para acessar partes protegidas da API do sistema possibilitando, por exemplo, que ela interaja com outras aplicações.
- » Declara, ainda, quais permissões as outras aplicações devem possuir para interagir com seus componentes.

Ao clicar no arquivo `AndroidManifest.xml`, temos como resultado:

Texto 1: Arquivo `AndroidManifest.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloworld"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19"/>
```

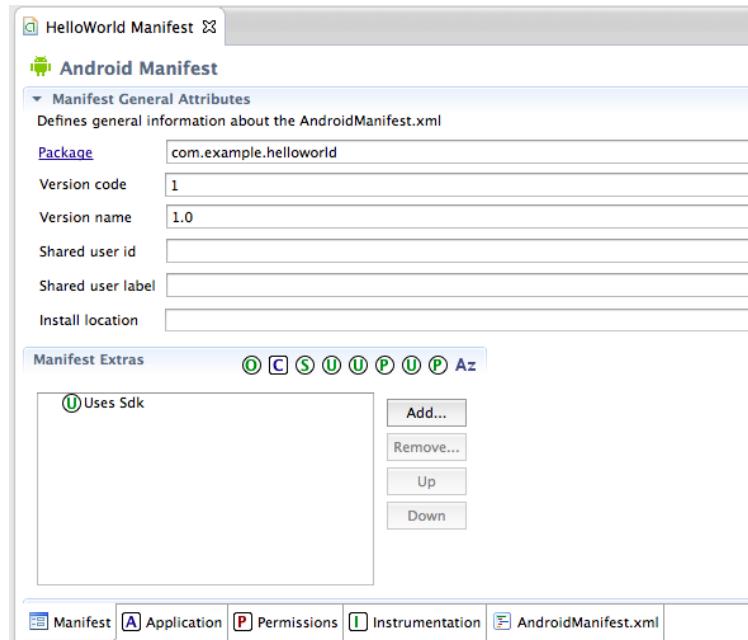
```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>

            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>
</manifest>
```

Podemos identificar, no arquivo XML, algumas tags importantes como `<uses-sdk android:minSdkVersion="8" />`, indicando a versão mínima da API para rodar a aplicação, ou a tag `<activity>`, definindo qual é a activity para iniciar.

O arquivo de manifesto pode ser manipulado por janelas de configuração presentes no plugin ADT, conforme pode ser visto a seguir:

Figura 27. Configurando o arquivo de Manifesto.



Fonte: Próprio autor.

CAPÍTULO 3

Activities

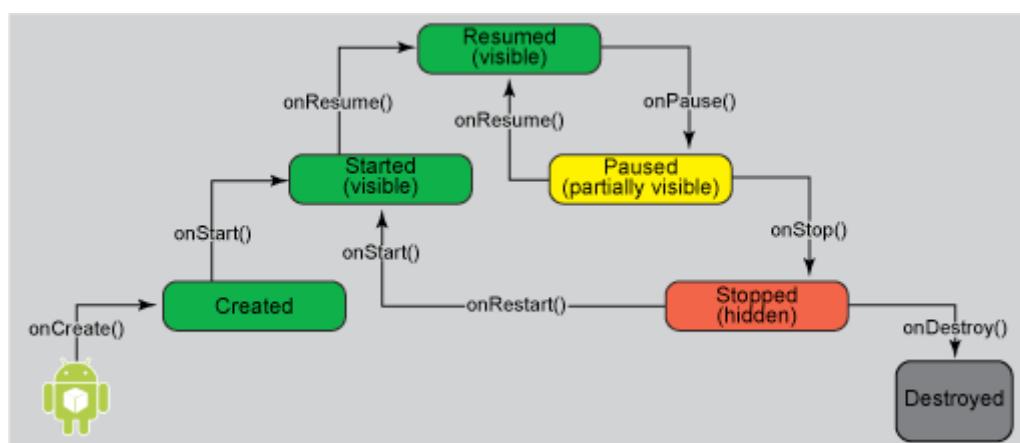
Uma atividade é um componente do aplicativo que fornece uma tela com o qual os usuários podem interagir com algum objetivo como tirar uma foto, discar um número de telefone, enviar um e-mail ou visualizar um mapa, por exemplo. Para cada atividade, é dada uma janela na qual é possível renderizar na interface de usuário.

A janela de uma activity pode preencher a tela parcialmente ou totalmente, podendo até mesmo flutuar em cima de outras janelas. Um aplicativo consiste em múltiplas atividades que são ligados umas nas outras por meio do que chamamos de Intent. Uma das atividades é especificada como “principal” e esta é apresentada ao usuário ao iniciar o aplicativo pela primeira vez. Cada atividade pode então disparar outra atividade, que, por sua vez, tem objetivos e ações diferentes. Para cada vez que uma nova atividade começa a atividade anterior é interrompida, e o sistema Android a preserva em uma pilha (a “pilha de volta”). É por essa razão que, quando clicamos no botão de “voltar” em um dispositivo Android, ele fecha a janela atual, retornando para a janela anteriormente aberta.

Este ciclo de vida da atividade, em que ela é criada, colocada em foco e pausada, retorna ao funcionamento e pode ser destruída. Métodos são disponibilizados como verdadeiros ganchos de programação da qual podemos realizar programações.

Veja o ciclo de vida com os métodos no diagrama no site AndroidDevelopers:

Figura 28. Ciclo de vida de uma Activity (do site AndroidDevelopers).



Fonte: Site AndroidDevelopers.

São os estados disponíveis:

- » **Created:** estado da atividade assim que o usuário clica no lançador (launcher) e executa o aplicativo.
- » **Started:** disparado assim que a atividade se torna visível.
- » **Resumed:** igual à Running. Activity/View em primeiro plano, usuário pode interagir.
- » **Paused:** encoberta por outra activity. Parcialmente visível e inacessível pelo usuário.
- » **Stopped:** totalmente encoberta, invisível para o usuário, considerada em segundo plano
- » **Destroyed:** destruído, assim que a atividade é fechada.

CAPÍTULO 4

Serviços

Os serviços são elementos de uma aplicação que executam em segundo plano e, portanto, não possuem uma interface gráfica com o usuário.

Um serviço pode se apresentar em duas formas:

- » **Unbound:** uma vez iniciado, é executado indefinidamente mesmo que o componente que o acionou (por exemplo, uma Activity) seja destruído.
- » **Bound:** dependente do componente que o iniciou e dos componentes associados a ele.

Para criar um serviço, basta estender a classe Service e oferecer a implementação de determinados métodos. A inicialização e comunicação com o serviço será sempre por meio de uma Intent.

Além da possibilidade de criação de serviços personalizados, existem também serviços prontos oferecidos pelo sistema operacional. Para obter acesso a um desses serviços do sistema, basta acionar o método:

`getSystemService(String P1)`, onde P1 corresponde ao tipo de serviço desejado (ex: ALARM_SERVICE, AUDIO_SERVICE, POWER_SERVICE, VIBRATOR_SERVICE, NOTIFICATION_SERVICE etc...), veja o exemplo a seguir:

Texto 2: Exemplo de acionamento de serviço padrão do Android.

```
Vibrator vs = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);  
vs.vibrate(5000);
```

A classe Service possui os seguintes métodos à disposição:

- » **onStart ()**: executado quando um componente inicia o serviço por meio do método `startService()` – serviço Started visto anteriormente. Existem na API < 2.0;
- » **onStartCommand ()**: idem ao anterior quando a API >= 2.0;

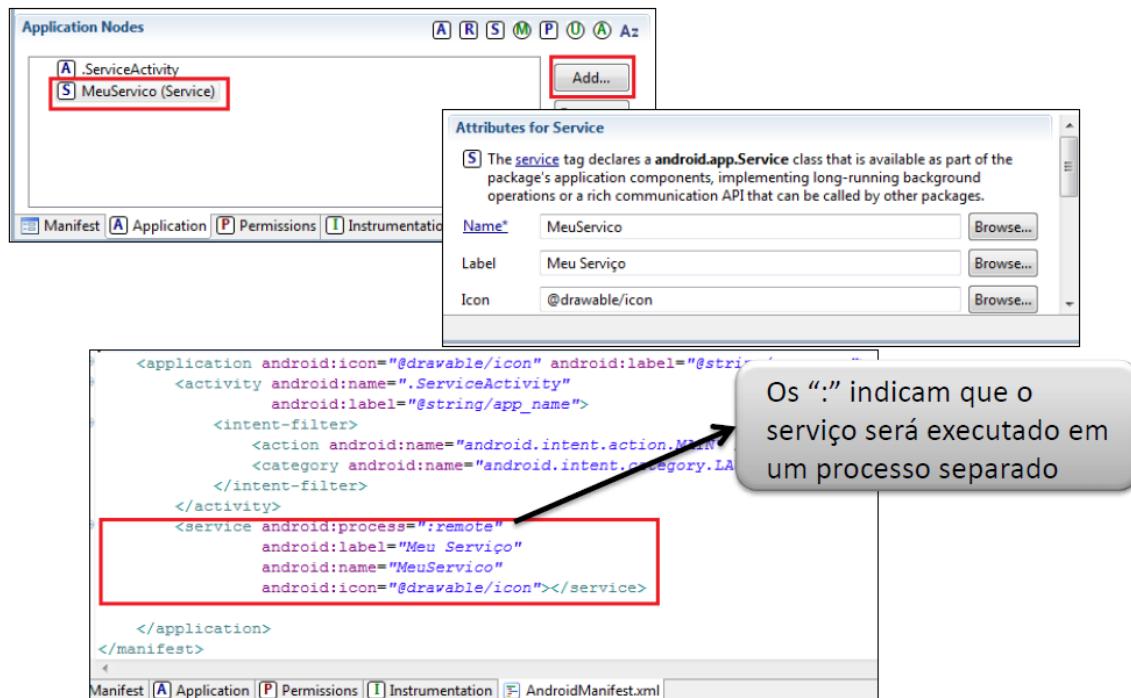
- » **onBind()**: executando quando um componente deseja associar-se ao serviço por meio do método bindService() – serviço Bound visto anteriormente;
- » **onCreate()**: executado quando o serviço é iniciado pela primeira vez;
- » **onDestroy()**: acionado quando o serviço é destruído.

Outros métodos importantes:

- » **stopSelf()**: para finalizar o serviço por ele próprio;
- » **stopService()**: quando outro componente deseja finalizar o serviço;

É necessário declarar o serviço no arquivo de manifesto `AndroidManifest.xml`:

Figura 29. Serviço declarado no arquivo de manifesto.



Fonte: próprio de autor.

Todo serviço deve estender da classe Service, presente no pacote `android.app`. Ela possui os métodos descritos acima à disposição. Veja a seguir um exemplo de Serviço personalizado:

Texto 3: Criando um serviço personalizado.

```
package br.com.wpos.androidtest.services;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MeuServiço extends Service {
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("MeuServiço", "Serviço Inicializado");
        return START_STICKY;
    }
}
```

Para interagir com o serviço, deverá se utilizar de um Intent. Por meio dele é possível iniciar ou parar um serviço:

Texto 4: Utilizando um Intent para interagir com o serviço personalizado através de uma Activity.

```
public class MainActivity extends ActionBarActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent i = new Intent(this, MeuServiço.class);
        // Iniciando o serviço
        startService(i);
        // Parando o serviço
        stopService(i);
    }
}
```

CAPÍTULO 5

Tarefas

Um serviço pode agendar tarefas que são executadas programaticamente de tempos em tempos. Para criar uma tarefa, basta estender a classe abstrata TimerTask e fornecer a implementação de seu método run(). Em seguida, basta agendar a sua execução por meio da classe Timer (pacote java.util) que possui o método: schedule (TimerTask P1, long P2, long p3), sendo:

- » **P1:** a tarefa a ser agendada;
- » **P2:** o tempo inicial a ser transcorrido para iniciar a tarefa novamente (em milissegundos);
- » **P3:** o tempo no qual a tarefa será executada novamente (em milissegundos);

Vejamos a seguir um exemplo de implementação de tarefa:

Texto 5: Criando uma tarefa personalizada.

```
package br.com.wpos.androidtest.tasks;  
import java.util.TimerTask;  
import android.util.Log;  
  
public class MinhaTarefa extends TimerTask {  
    @Override  
    public void run() {  
        Log.i("Minha Tarefa", "tarefa rodando!");  
    }  
}
```

E, na sequência, um agendamento da tarefa personalizada a partir de um serviço personalizado:

Texto 6: Agendando uma tarefa personalizada a partir de um serviço personalizado (Parte 1).

```
package br.com.wpos.androidtest.services;  
import java.util.Timer;  
import br.com.wpos.androidtest.tasks.MinhaTarefa;  
import android.app.Service;
```

```

import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MeuServiço extends Service {
    private Timer timer;
    private MinhaTarefa tarefa;
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("MeuServiço", "Serviço Inicializado");
        return START_STICKY;
    }
}

```

Texto 7: Agendando uma tarefa personalizada a partir de um serviço personalizado (Parte 2).

```

@Override
public void onCreate()
{
    timer = new Timer();
    tarefa = new MinhaTarefa();
    // Tarefa agendada para começar 1 segundo,
    // repetindo-se a cada 5 segundos
    timer.schedule(tarefa, 1000, 5000);
}
@Override
public void onDestroy()
{
    tarefa.cancel();
    timer.cancel();
    super.onDestroy();
}

```

Notificações

As notificações oferecem um recurso interessante para comunicar ao usuário sobre a execução ou resultado de determinada operação (principalmente quando associados a serviços em background). Estas notificações são gerenciadas por meio do NotificationManager que deve ser obtido do serviço padrão do sistema Android, conhecido como NOTIFICATION_SERVICE:

```
NotificationManager nm = (NotificationManager)  
getSystemService(Context.NOTIFICATION_SERVICE);
```

As notificações podem ser criadas por meio da instanciação da classe Notification:

```
Notification n = new Notification(int P1, CharSequence P2, long P3)
```

Sendo:

- » **P1:** ícone da notificação (deve estar presente nas pastas “drawable”, acessível via R.drawable...);
- » **P2:** Texto a ser exibido.
- » **P3:** Quando exibir. Para exibir imediatamente, utilizar a função System.currentTimeMillis()

Ao se clicar em uma notificação exibida, alguma ação deve ser executada. Para fazê-lo, é necessário associá-la a uma PendingIntent (como o nome já indica, é uma Intent que fica pendente). Pode ser associado por meio do método setLatestEventInfo():

Texto 8: Exemplo de Notificação

```
Intent notifyIntent = new Intent(this, OutraActivity.class);  
PendingIntent i = PendingIntent.getActivity(this, 0, notifyIntent, 0);  
notifyDetails.setLatestEventInfo(this, "Título", "Texto", i);  
// Exibe a notificação  
nm.notify(0, notifyDetails);
```

Existem outros métodos úteis provados por NotificationManager, entre eles:

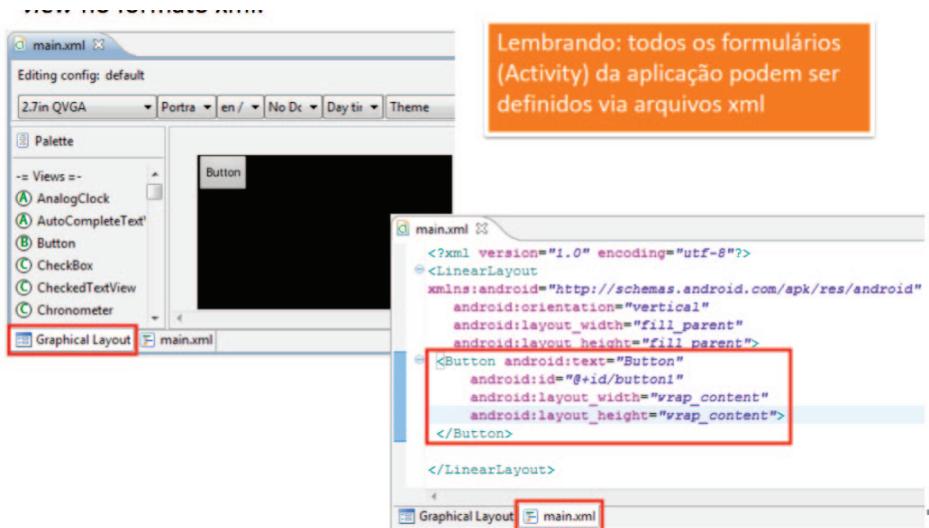
- » cancel(int P1) : cancela a notificação identificada em P1;
- » cancelAll() : cancela todas as notificações.

CAPÍTULO 6

Views

Chamam-se views os controles que podem ser adicionados a uma Activity. As views criadas pelo editor gráfico no ambiente Eclipse na realidade criam a view no formato xml:

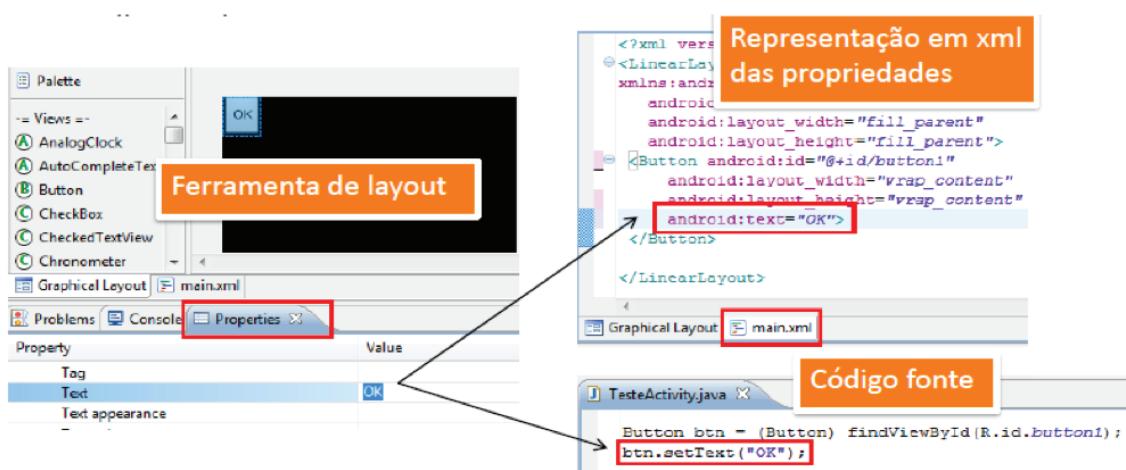
Figura 30. Views sendo geradas no editor gráfico que se tornam códigos XML.



Fonte: próprio autor.

Views possuem propriedades que permitem uma personalização de suas características. Algumas propriedades são mais genéricas, presentes em todas as views, outras mais específicas. Cada propriedade alterada na aba Properties gera um atributo XML correspondente e vice-versa:

Figura 31. Propriedades de uma View.



Fonte: próprio autor.

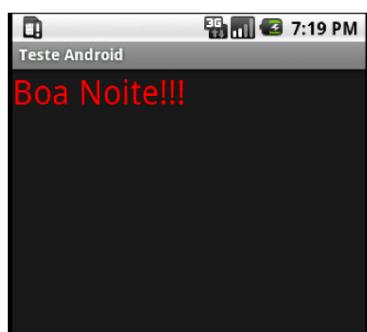
View do tipo TextView

Exibe um texto sem a possibilidade de edição. É o rótulo ou label.

Possui métodos como:

- » **setText(CharSequence p1)** : define o texto a ser exibido.
- » **setTextSize (float p1)** : define o tamanho do texto.
- » **setTextColor (int P1)** : define a cor do texto, em que P1 pode ser obtido das constantes da classe Color.
- » **setBackgroundColor (int P1)** : define a cor de fundo.

Figura 32. View do tipo TextView



Fonte: próprio autor.

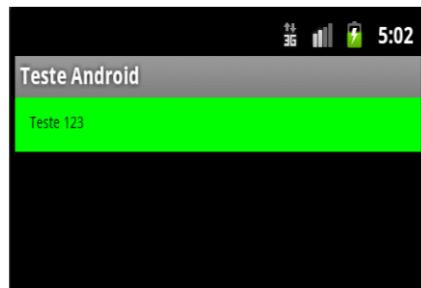
View do tipo EditText

Permite que o usuário insira informações por meio da digitação de um texto. Trata-se da tradicional caixa de texto.

Possui métodos como:

- » **setText(String p1)** : define o texto a ser exibido na caixa de texto.
- » **EditablegetText()** : retorna o texto digitado. Para obter a String informada no campo, deve-se adicionar o método **.toString()**.
- » **setEnabled(boolean p1)** : habilita (true) ou desabilita (false) a edição.

Figura 33. View do tipo EditText



Fonte: próprio autor.

View do tipo ImageView

Permite a exibição de imagens.

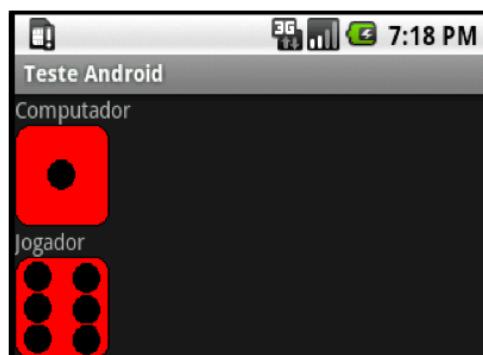
A imagem pode ser definida via propriedade src e as imagens devem estar salvas nas pastas res\drawable. Existem várias pastas, prevendo várias resoluções de tela diferentes. O dispositivo utilizará a melhor resolução de imagem para a resolução utilizada.

A imagem da ImageView pode ser trocada a qualquer momento por meio do método setImageResource que recebe como parâmetro a referência à imagem contida nos diretórios res\drawable. Exemplo:

Texto 9: Exemplo de código para ImageView.

```
ImageView img = (ImageView) findViewById(R.id.imgJogador);
imgDadoJogador.setImageResource(R.drawable.dado2);
```

Figura 34. View do tipo ImageView

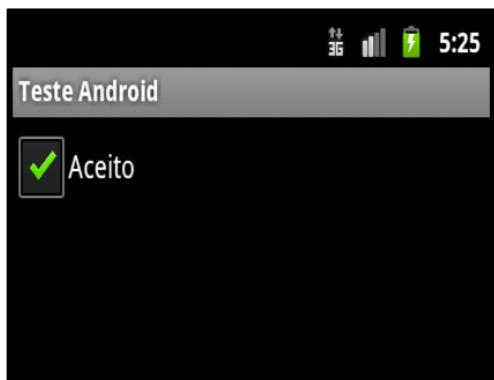


Fonte: próprio autor.

View do tipo Checkbox

Cria uma caixa de seleção.

Figura 35. View do tipo Checkbox.



Fonte: próprio autor.

Principais métodos:

- » **isChecked()** : verifica se o *checkbox* está selecionado (*true / false*);
- » **setChecked(boolean p1)** : seleciona ou não o *checkbox* (*true / false*);

View do tipo RadioButton

Cria itens agrupados de seleção exclusiva. As opções para seleção, são os **RadioButton**, que devem ser agrupadas em um **RadioGroup**; Para descobrir qual o item selecionado basta chamar o método do RadioGroup `int getCheckedRadioButtonId()`, que retorna o identificador do item selecionado no momento.

Por exemplo, para verificar qual a cor selecionada nos itens acima, supondo que o name dos itens sejam radAzul, radAmarelo e radVermelho:

Texto 10. Exemplo de código para RadioButton

```
RadioGrouprg = (RadioGroup) findViewById(R.id.radioGroupCores);
if (rg.getCheckedRadioButtonId() == R.id.radAzul) { ... }

elseif (rg.getCheckedRadioButtonId() == R.id.radAmarelo) { ... }
elseif (rg.getCheckedRadioButtonId() == R.id.radVermelho) {
... }
```

Figura 36. View do tipo RadioButton.



Fonte: próprio autor.

View do tipo Spinner

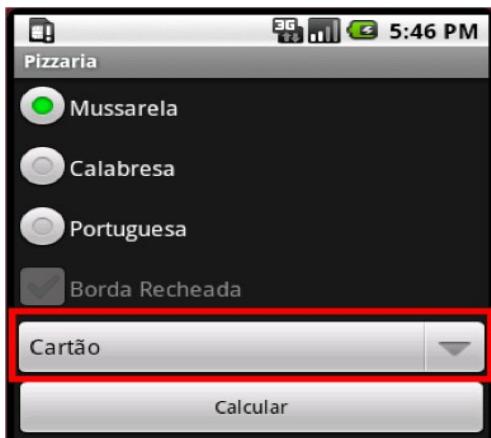
Cria uma lista de opções para seleção, parecido com um Combo Box;

Os itens da lista podem ser definidos externamente, por exemplo, no arquivo strings.xml (dentro da pasta res/values), como no exemplo a seguir:

Texto 11: Exemplo de lista de valores para Spinner em strings.xml.

```
<string-array name="forma_pagamento">
    <item>Cartão</item>
    <item>Cheque</item>
    <item>Dinheiro</item>
</string-array>
```

Figura 37. View do tipo Spinner



Fonte: próprio autor.

Para carregar os itens dentro do Spinner, o código abaixo é um exemplo:

Texto 12: Exemplo da carga do *Spinner* a partir da lista em *strings.xml*.

```
Spinnerpg = (Spinner) findViewById(R.id.spiFormaPagto);  
String[] pagto = getResources().getStringArray(R.array.forma_  
pagamento);  
ArrayAdapteraPagto = new ArrayAdapter<String>(this,  
        android.R.layout.simple_spinner_item, pagto);  
pg.setAdapter(aPagto);
```

Views do tipo DatePicker e TimePicker

Permitem a seleção de data e hora. Em caso de conversão das datas selecionadas em um objeto, basta utilizar o método construtor da classe GregorianCalendar.

Principais métodos:

- » **getDayOfMonth()** : retorna o dia do mês.
- » **getMonth()** : retorna o mês (janeiro = 0!).
- » **getYear()** : retorna o ano.
- » **updateDate(int ano, intmês, int dia)** : atualiza a data atual.
- » **getCurrentHour()** : retorna a hora.
- » **getCurrentMinute()** : retorna os minutos.

Para atualizar os valores da hora e minuto, basta utilizar os respectivos métodos set dos getters apresentados aqui.

Figura 38. View do tipo DatePicker e TimePicker.



Fonte: próprio autor

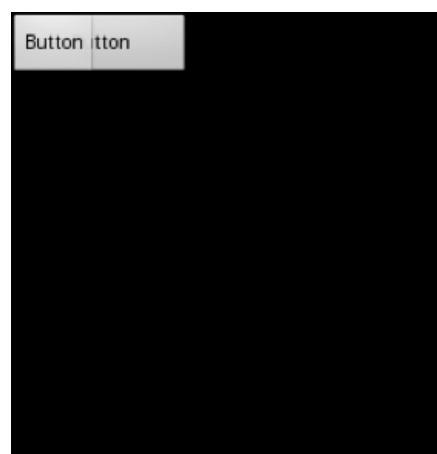
CAPÍTULO 7

Layout

Os elementos podem ser dispostos de diversas maneiras em uma atividade.

- » **FrameLayout:** é o tipo de layout mais limitado no qual as views são exibidas na mesma posição, isto é, no canto superior esquerdo do display.

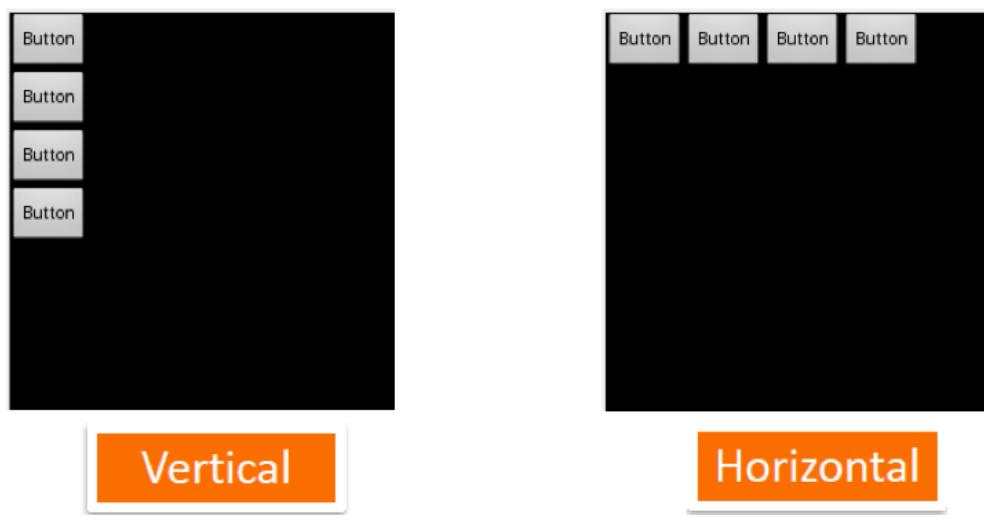
Figura 39. Layout do tipo FrameLayout.



Fonte: próprio autor.

- » **LinearLayout:** é um dos tipos de gerenciadores de layout mais utilizados nos projetos, pois organiza as views lado a lado em linhas (modo vertical) ou colunas (modo horizontal).

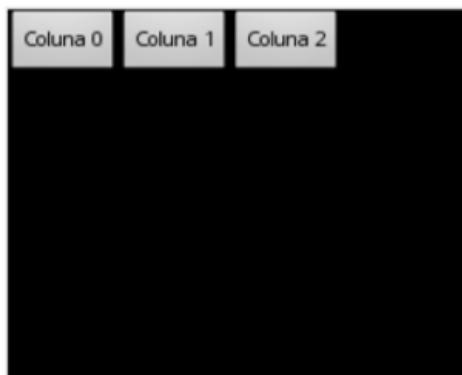
Figura 40. Layout do tipo LinearLayout.



Fonte: próprio autor.

- » **TableLayout:** organiza as views em uma tabela de modo semelhante às tabelas HTML: cada linha da tabela é representada por uma TableRow. Existe uma propriedade StretchColumns utilizada para indicar quando uma coluna deve ocupar do espaço restante não utilizado.

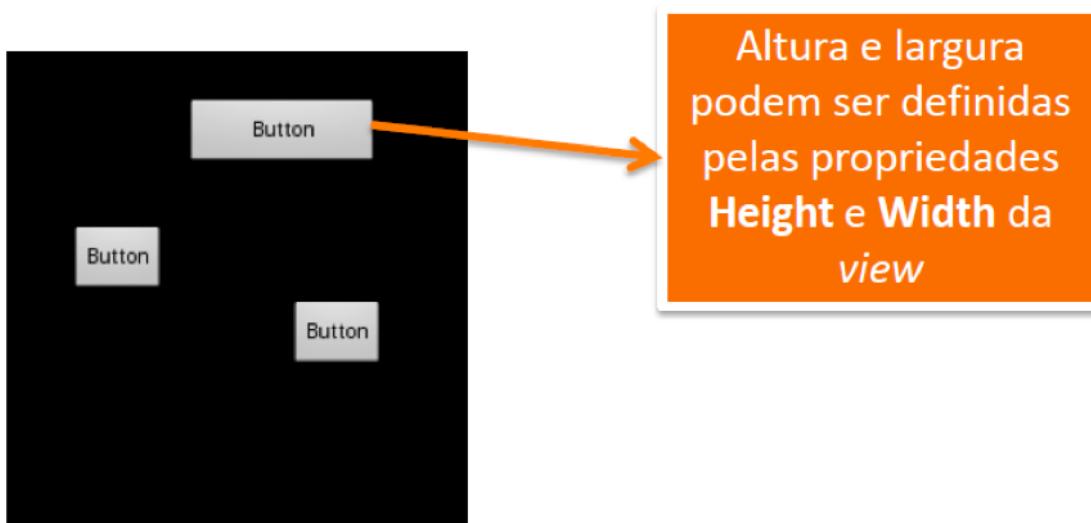
Figura 41. Layout do tipo TableLayout.



Fonte: próprio autor.

- » **AbsoluteLayout:** este gerenciador de layout respeita o tamanho e posição das views de acordo com a definição do usuário. Trata-se do que apresenta a maior flexibilidade no posicionamento das views, embora seu ajuste automático das views não seja dos melhores.

Figura 42. Layout do tipo AbsoluteLayout.

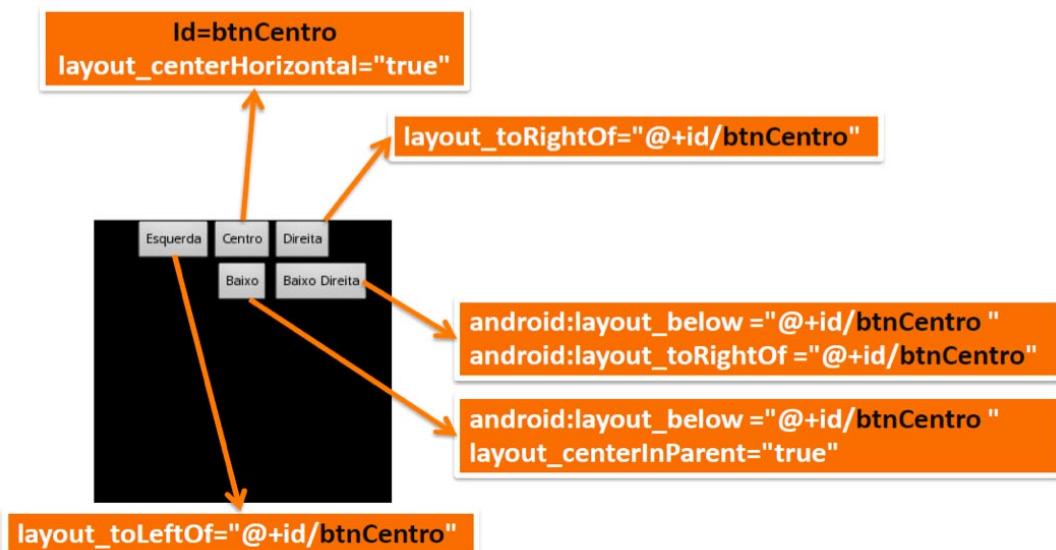


Fonte: próprio autor.

- » **RelativeLayout:** define a posição de uma view relativa à posição de outra view:

- » esquerda – layout_toLeftOf;
- » direita – layout_toRightOf;
- » acima – layout_above;
- » abaixo – layout_below.

Figura 43. Layout do tipo RelativeLayout.



Fonte: próprio autor.

Layouts podem ser combinados para um melhor resultado, como veremos adiante.

CAPÍTULO 8

Exemplo média aluno

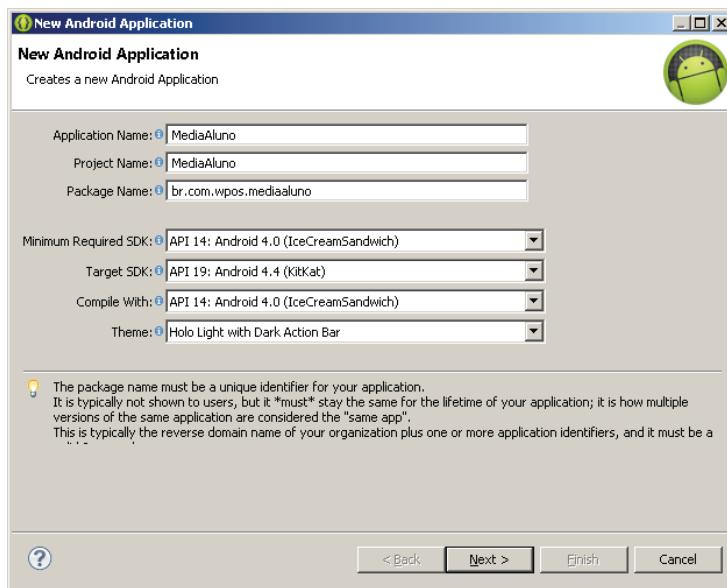
Depois de providenciar um ambiente de desenvolvimento adequado, testá-lo com o tradicional “Alô Mundo” e conhecer os conceitos importantes relacionados ao desenvolvimento para Android, chegou a hora de um exemplo mais elaborado, consolidando tudo aquilo visto até agora.

A aplicação a seguir é um formulário que receberá o nome do usuário e três notas acadêmicas na tradicional escala de zero a dez. Após uma média aritmética simples, verificaremos se o valor calculado é acima ou igual a 7, e o consideraremos aprovado, exibindo uma mensagem de sucesso. Caso contrário, exibiremos a mensagem de reprovando.

Criando o projeto MediaAluno

Criaremos um projeto chamado “MediaAluno”, seguindo os mesmos passos vistos antes. Uma vez que as versões 4.0 (Ice Cream Sandwich) e superiores contemplam mais de 85% do mercado, é seguro pensar nesta versão como SDK mínimo para rodar nossa aplicação. Entretanto, por se tratar de um exemplo relativamente simples, este exemplo poderia ser realizado com o FrozenYogurt (Froyo, versão 2.2) como SDK mínimo sem qualquer tipo de problema.

Figura 44. Criando o projeto MediaAluno.



Fonte: próprio autor.

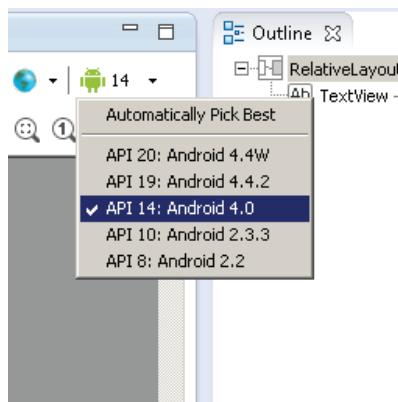
A AndroidVirtualizationDevice – AVD que temos à disposição é uma Froyo 2.2. Crie um novo AVD com a versão Ice Cream Sandwich 4.0 ou, se preferir, crie o projeto “MediaAluno” utilizando a versão 2.2 como SDK mínimo, possibilitando os testes na AVD que já está disponível.

Informando a API de referência

Após o projeto ser criado, o ADT abrirá o layout MainActivity em seu editor gráfico de tela. Por padrão, o ADT utiliza a API mais recente (“AutomaticallyPick Best”, em nosso caso 4.4W), portanto, é natural que algum erro se apresente nesta tela, a não ser que você tenha baixado a API do 4.4W previamente utilizando o SDK Manager.

Informaremos a API mínima do projeto (versão 14, do Android 4.0) para que seja utilizado o Toolkit gráfico esta versão. Sinta-se à vontade para informar a API 8 (Android 2.2) se for o caso.

Figura 45. Informando a API de referência ao plugin para manipulação gráfica.



Fonte: próprio autor.

Layout para MainActivity

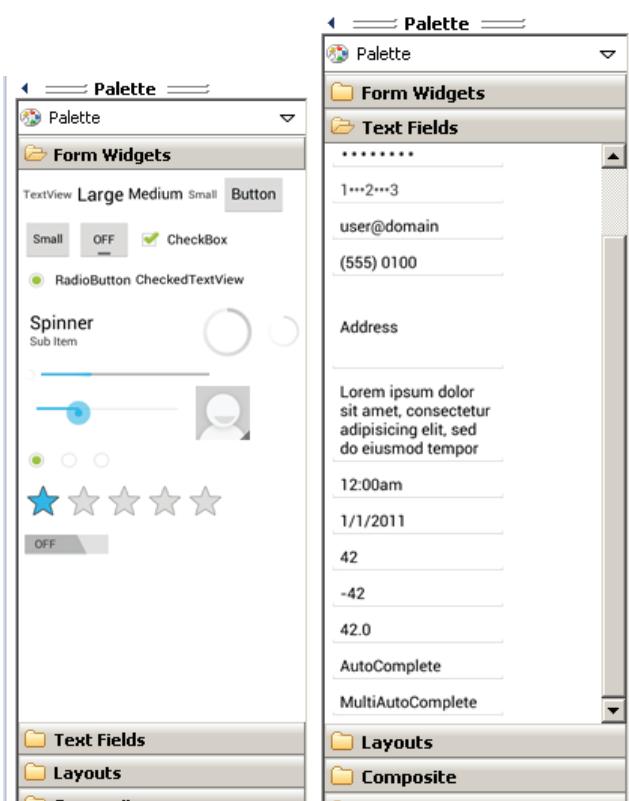
Conforme mencionado anteriormente, criaremos um formulário possibilitando ao usuário digitar seu nome e três notas acadêmicas. Para que a aplicação faça os cálculos apenas após a digitação dos valores, um botão será adicionado a este formulário.

Aprendemos vários tipos de layout para aplicações Android, como os layouts absolutos, relativos, tabulares, grid ou lineares. Para melhores resultados, utilizaremos neste um layout linear vertical (que dispõe cada elemento de tela automaticamente um embaixo do outro) mesclado com um layout linear horizontal (que dispõe os seus elementos automaticamente lado a lado). Desta forma, cada rótulo ou label de campo (elemento

conhecido como TextView) será acompanhado de uma caixa de texto (EditText), cada campo em uma linha.

Os layouts estão no grupo de elementos “Layouts” da Palette do toolkit gráfico, assim como labels (TextView) e botões (Button) podem ser encontrados em “FormWidgets” e várias versões de caixa de texto (EditText) estão presentes em “TextFields”. Após a definição do layout, recomendamos fazê-lo utilizando a versão XML do layout, estes elementos de palette pode ser aplicado ao layout com uma simples operação de arrastar e soltar (DragandDrop). Entretanto, é interessante se acostumar com o editor XML de layout que, embora não pareça amigável a princípio, possibilita uma personalização de tela rápida e robusta.

Figura 46. Palette contendo o toolkit gráfico (versão API 14).



Fonte: próprio autor.

O código-fonte em XML de mainActivity.xml deve ser semelhante a este:

Texto 13: MainActivity.xml (Parte 1 de 3).

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:orientation="vertical" <!-- Linear vertical geral -->
    tools:context=".MainActivity">

        <LinearLayout
            android:layout_width="match_parent" <!-- match_parent para
            o comprimento do layout, adequando-se ao pai (linear vertical),
            utilizando todo o espaço disponível -->
            android:layout_height="wrap_content" <!-- wrap_content na
            altura do layout, ou seja, altura do conteúdo = altura da linha
            -->
            android:orientation="horizontal" <!-- Linear horizontal -->
            android:paddingLeft="10dp" <!-- borda à esquerda -->
            <TextView
                android:id="@+id/lblNome" <!-- id do rótulo para nome
            -->
                android:layout_width="70dp"
                android:layout_height="wrap_content"
                android:text="@string/lblNome" <!-- string para in-
                ternacionalização -->
            />
<EditText
    android:id="@+id/txtNome" <!-- id do caixa de texto nome
-->
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="text" <!-- caixa do tipo texto -->
    <requestFocus/> <!-- foco automático no elemento -->
</EditText>
</LinearLayout>
```

Texto 14: MainActivity.xml (Parte 2 de 3).

```
<LinearLayout <!-- cada linear horizontal, um label e caixa
-->
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:paddingLeft="10dp">
```

```
<TextView
    android:id="@+id/lblNota1"
    android:layout_width="70dp"
    android:layout_height="wrap_content"
    android:text="@string/lblNota1"/>

<EditText
    android:id="@+id/txtNota1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="5"
    android:inputType="numberDecimal"/><!-- caixa específica
para números decimais (habilita o teclado numérico nos dispositivos) -->
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:paddingLeft="10dp">
    <TextView
        android:id="@+id/lblNota2"
        android:layout_width="70dp"
        android:layout_height="wrap_content"
        android:text="@string/lblNota2"/>

    <EditText
        android:id="@+id/txtNota2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="5"<!-- 5 caracteres numéricos = caixas menores -->
        android:inputType="numberDecimal"/>
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
```

```
    android:paddingLeft="10dp">
```

Texto15: MainActivity.xml (Parte 3 de 3).

```
<TextView  
    android:id="@+id/lblNota3"  
    android:layout_width="70dp"  
    android:layout_height="wrap_content"  
    android:text="@string/lblNota3"/>  
  
<EditText  
    android:id="@+id/txtNota3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:ems="5"  
    android:inputType="numberDecimal"/>  
</LinearLayout>  
  
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal"  
    android:gravity="center" <!-- gravity em center posiciona o botão no centro da linha horizontal -->  
    android:paddingTop="10dp" >  
  
<Button  
    android:id="@+id	btnCalcular"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/btnCalcular"  
    android:onClick="calcularMedia"/><!-- ação realizada ao clique do botão, programaremos posteriormente -->  
  
</LinearLayout>  
</LinearLayout>
```

Como prevê a boa prática, todos os elementos em tela foram internacionalizados, pois todos eles em seus `android:text` se referem a uma string presente no arquivo strings.xml (`@string/...`).

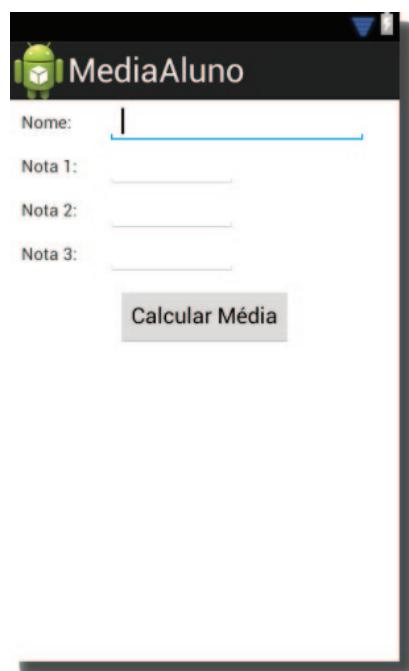
Abra o arquivo strings.xml presente na pasta **res/values** e preencha-o de forma similar a este:

Texto 16: Arquivo strings.xml na pasta res/values.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">MediaAluno</string>
<string name="action_settings">Settings</string>
<string name="lblNome">Nome: </string>
<string name="lblNota1">Nota 1: </string>
<string name="lblNota2">Nota 2: </string>
<string name="lblNota3">Nota 3: </string>
<string name="btnCalcular">Calcular Média</string>
<string name="title_activity_result">Resultado</string><!--
-será criado posteriormente -->
</resources>
```

Ao abrir o layout de MainActivity no editor gráfico, teremos como resultado:

Figura 47. O layout de MainActivity no editor gráfico.

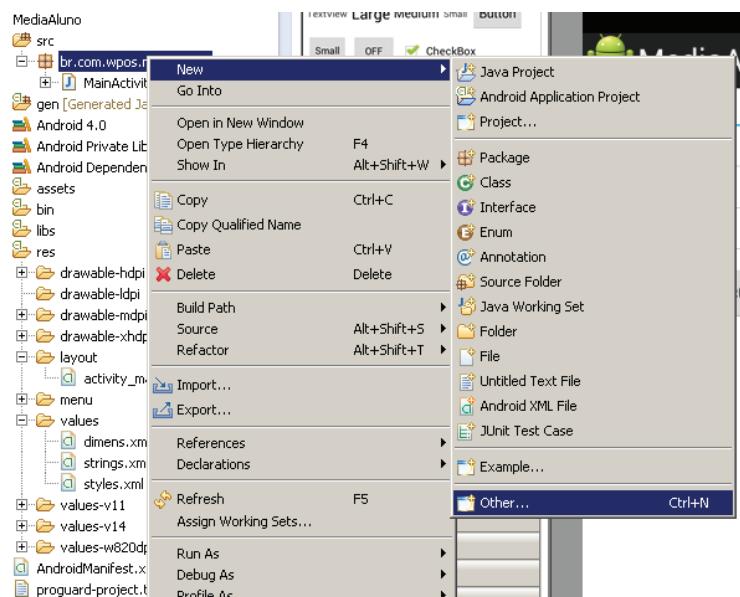


Fonte: próprio autor.

Adicionando uma nova activity (ResultActivity)

Para tornar o exemplo mais interessante, ao clicar no botão “Calcular Média”, o resultado será apresentado em uma segunda activity (chamaremos de ResultActivity), chamada pela principal (MainActivity). Para isso, clique com o botão direito do mouse no projeto, selecione New >Other:

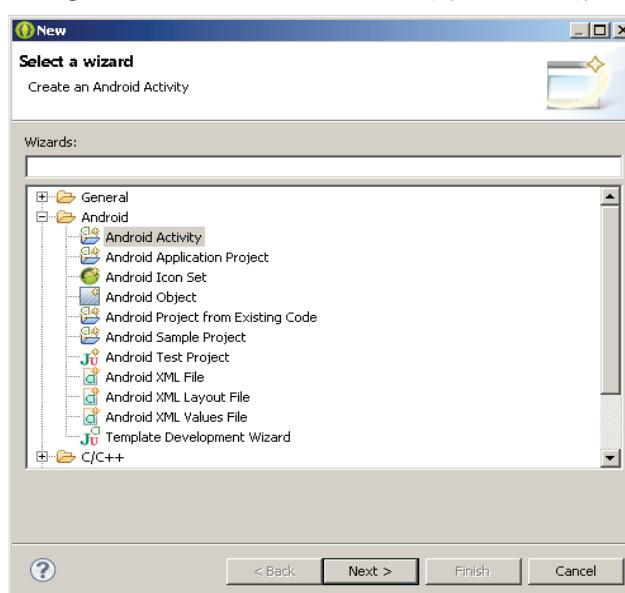
Figura 48. Criando uma nova activity (ResultActivity).



Fonte: próprio autor.

Selecione a opção “Android Activity”:

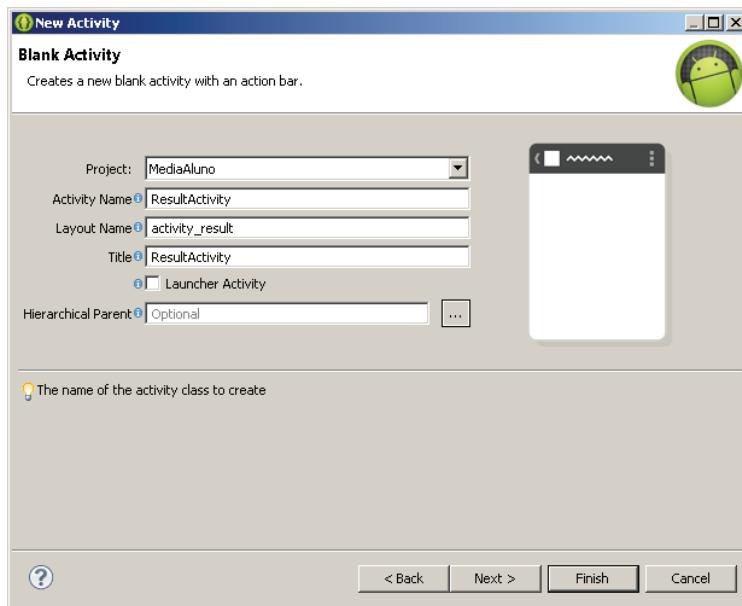
Figura 49. Criando uma nova activity (ResultActivity).



Fonte: próprio autor.

Repare que a criação de uma nova activity se assemelha muito à criação da primeira (MainActivity) selecione a opção “Black Activity” e na nomeação coloque como se apresenta abaixo:

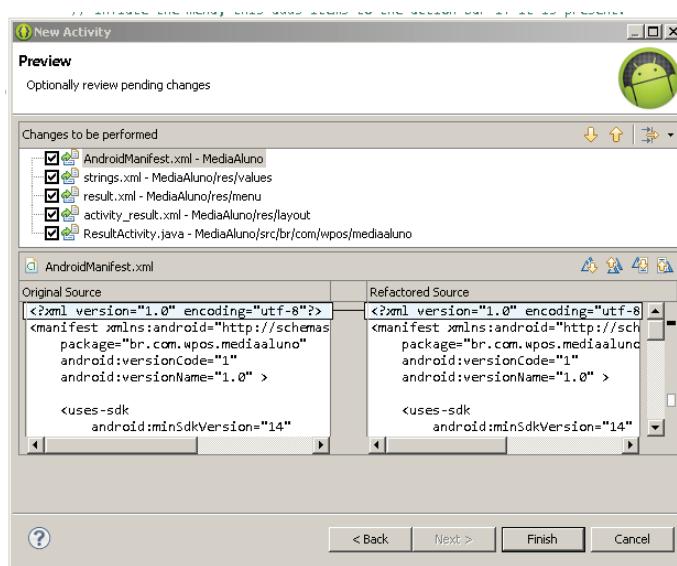
Figura 50. Criando uma nova activity (ResultActivity).



Fonte: próprio autor.

Na tela seguinte, é possível ver o impacto da criação da nova activity no projeto: AndroidManifest.xml, strings.xml, a criação o result.xml (para seu menu), o arquivo ResultActivity.java (para a programação desta activity) além de activity_result.xml, para seu layout.

Figura 51. Criando uma nova activity (ResultActivity).



Fonte: próprio autor.

Abra o arquivo activity_result.xml na pasta de layout e adicione um TextView para apresentar o resultado:

Texto17: O layout em activity_result.xml.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="br.com.wpos.mediaaluno.ResultActivity">

    <TextView<!-- para apresentar o resultado do cálculo -->
        android:id="@+id/lblResultado"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" />

</RelativeLayout>
```

Programando MainActivity.java

Deveremos agora programar a activity, utilizando a linguagem Java, para que se faça o esperado: calcular a média aritmética simples, verificar a situação acadêmica e apresentar o resultado.

O arquivo começará da seguinte maneira:

Texto 18: O arquivo MainActivity.java (Parte 1 de 4).

```
package br.com.wpos.mediaaluno;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent; /* importar */
import android.view.Menu;
```

```
import android.view.View;
import android.widget.EditText; /* importar */

public class MainActivity extends Activity {

    /* Adicionar estas linhas */
    private EditText txtNome, txtNota1, txtNota2, txtNota3;
    public final static String RESULTADO = "";

}
```

As bibliotecas importadas são para os comandos que seguirão. Para chamar uma nova activity, será necessário um Intent, o que explica a presença de `import android.content.Intent;`. Os elementos EditText na tela deverão ser referenciados no código Java, por esta razão a presença de `import android.widget.EditText;`.

As caixas de texto (EditText) serão representadas neste arquivo .java por atributos com o mesmo nome de seus ids, por isso a linha `private EditText txtNome, txtNota1, txtNota2, txtNota3;` se faz presente.

O atributo estático `public final static String RESULTADO = "";`; será nosso meio de transporte da frase de resultado a ser montado e exibido na activity posteiror.

Texto 19: O arquivo MainActivity.java (Parte 2 de 4).

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

    /* Adicionar estas linhas */
    txtNome = (EditText) findViewById(R.id.txtNome);
    txtNota1 = (EditText) findViewById(R.id.txtNota1);
    txtNota2 = (EditText) findViewById(R.id.txtNota2);
    txtNota3 = (EditText) findViewById(R.id.txtNota3);
}
```

Conforme visto anteriormente, o método `onCreate()` é disparado automaticamente no momento a criação da activity quando o aplicativo está sendo executado, ou seja, momentos antes do formulário ser renderizado na tela do portátil do usuário, estas quatro linhas serão executadas.

O comando `findViewById()` é responsável por localizar o elemento do tipo View pelo id. Cada um dos elementos criados no aplicativo foi automaticamente criado no arquivo

R.java, e é por essa razão que `R.id.txtNome` é um identificador válido para a caixa de texto do nome.

O retorno de `findViewById()` é um objeto do tipo `View`, e o casting (`EditText`) se faz necessário para que o objeto mais genérico (`View`) seja transformado em algo mais específico (`EditText`).

Texto 20: O arquivo `MainActivity.java` (Parte 3 de 4).

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is
    // present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

/* Novo método (a ser acionado ao clique do botão */
public void calcularMedia(View v)
{
    float floMedia;

    String strNome = txtNome.getText().toString();
    float floNota1 = Float.parseFloat(txtNota1.getText().to-
String());
    float floNota2 = Float.parseFloat(txtNota2.getText().to-
String());
    float floNota3 = Float.parseFloat(txtNota3.getText().to-
String());

    floMedia = (floNota1 + floNota2 + floNota3) / 3;
}

```

O novo método `calcularMedia()` será acionado quando o usuário digitar o nome, as notas e clicar no botão “Calcular Média”. O parâmetro de entrada recebendo uma `View` é obrigatório e se trata da `View` que disparou o evento (em nosso caso, o botão `btnCalcular`).

A variável `floMedia` foi criada para receber o resultado da conta que será feita posteriormente. Repare que a variável `strNome` recebe `txtNome.getText().toString();`, pois a variável `txtNome` representa o objeto `EditText` como um todo,

tornando-nos capazes de receber ou mesmo manipular quaisquer características do elemento (cor de fundo da caixa, fonte, cor da borda, se está ou não com foco etc.). O resultado de `getText()` não é uma String e sim um objeto do tipo Editable e, por esta razão, o método `.toString()` se faz necessário.

Procedimento similar foi utilizado para as notas, exceto pela necessidade de se converter **String** para **float**, realizado pelo método `Float.parseFloat()`.

Texto 21: O arquivo MainActivity.java (Parte 4 de 4).

```
String situacao="";  
  
if (floMedia>= 7)  
{  
    situacao = "aprovado";  
}  
else  
{  
    situacao = "reprovado";  
}  
  
// Criação do Intent que intermedia as activities  
Intent intent = new Intent(this, ResultActivity.class);  
  
// Armazenando o que será enviado a seguir em RESULTADO  
intent.putExtra(RESULTADO, "Olá, "+strNome+" você foi  
"+situacao+  
    " com a média "+floMedia);  
  
// Adicionar estas linhas */  
startActivity(intent);  
}  
  
}
```

O código se encerra com a condição que verifica a situação acadêmica do usuário, seguido pela instanciação do Intent `Intent intent = new Intent(this, ResultActivity.class);` que intermedia esta activity (`this`) e a seguinte (`ResultActivity.class`).

A informação que será exibida em ResultActivity deve ser transportada via intent, utilizando o atributo estático resultado (`intent.putExtra(RESULTADO, "Olá, "+strNome+" você foi "+situacao+" com a média "+floMedia);`).

A activityResultActivity é chamada por meio do método `startActivity(intent);`.

Programando ResultActivity.java

Para terminar o exemplo, tornando-o funcional, resta um pequeno trecho de código em ResultActivity:

Texto 22. O arquivo ResultActivity.java.

```
package br.com.wpos.mediaaluno;

import br.com.wpos.mediaaluno.MainActivity;
import br.com.wpos.mediaaluno.R;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class ResultActivity extends Activity {

    // Atributo para representar o rótulo lblResultado.
    private TextView lblResultado;

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

        // Adicionar as três linhas abaixo.
        lblResultado = (TextView) findViewById(R.
id.lblResultado);
        Intent intent = getIntent();
    }
}
```

lblRe -

```
sultado.setText(intent.getStringExtra(
    MainActivity.RESULTADO));
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    if it is present.
    getMenuInflater().inflate(R.menu.result, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar
    will
    // automatically handle clicks on the Home/Up button,
    so long
    // as you specify a parent activity in AndroidMani-
    fest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

A primeira linha adicionada é familiar. O atributo `lblResultado` é criado como um atributo do tipo `TextView` (`private TextView lblResultado;`).

Dentro do método `onCreate`, outro procedimento familiar, a captura do rótulo por meio de `findViewById()`, só que o casting desta vez é para um `TextView` (`lblResultado = (TextView) findViewById(R.id.lblResultado);`).

O método `getIntent()` retorna o Intent utilizado como intermediário (`Intent intent = getIntent();`).

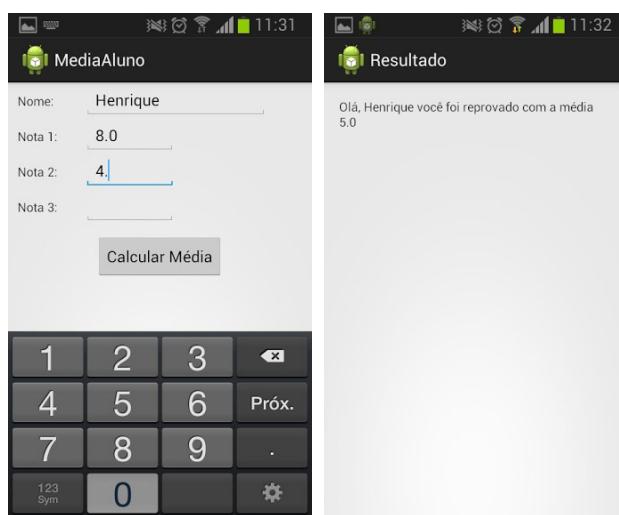
Terminando com o intent retornando a String enviada em caráter extraordinário `intent.getStringExtra()` pelo atributo da activity anterior `MainActivity`.

RESULTADO. A informação por sua vez é enviada ao rótulo manipulando sua propriedade texto `lblResultado.setText()`.

Testando a aplicação

O aplicativo pode ser testado no AVD criado ou mesmo em um smartphone real (veremos adiante):

Figura 52. O aplicativo MediaAluno sendo testado.



Fonte: próprio autor.

Adicionando uma Notificação Simples

Tornando a aplicação mais interessante e cheia de recursos, faremos uma pequena modificação: o clique de botão não irá mais iniciar a `activityResultActivity`; ao invés disso, disparará uma notificação e esta, por sua vez, permitirá o acesso à `activity` posterior. Abra `MainActivity.java` e, ao final dela:

Texto 23: O arquivo `MainActivity.java` modificado (Parte 4 de 4).

```
// Criação do Intent que intermedia as activities
Intent intent = newIntent(this, ResultActivity.class);

// Armazenando o que será enviado a seguir em RESULTADO
intent.putExtra(RESULTADO, "Olá, "+strNome+
"+situacao+
" com a média "+floMedia);
```

```
// Comentar ou remover a linha abaixo
// startActivity(intent);

// Como a notificação será construída
NotificationCompat.Builder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_launcher)
        .setContentTitle("Média Aluno")
        .setContentText("Situação Calculada!");

TaskStackBuilder stackBuilder = TaskStackBuilder.
create(this);
stackBuilder.addParentStack(ResultActivity.class);

// Adicionando a Intent queliga as duas actividades
stackBuilder.addNextIntent(intent);

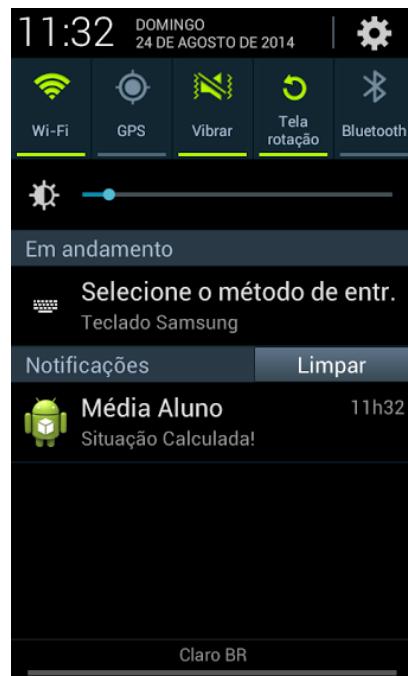
// Adicionando uma PendingIntent
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
/* Acessa o gerenciador de notificações do Android e dispara a notificação */
byte mId = 1;
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

mNotificationManager.notify(mId, mBuilder.build());
}

}

Ao testar, chegamos ao seguinte resultado clicando no botão “Calcular Média”:
```

Figura 53. Notificação da aplicação MediaAluno.



Fonte: próprio autor.

CAPÍTULO 9

Exemplo Câmera

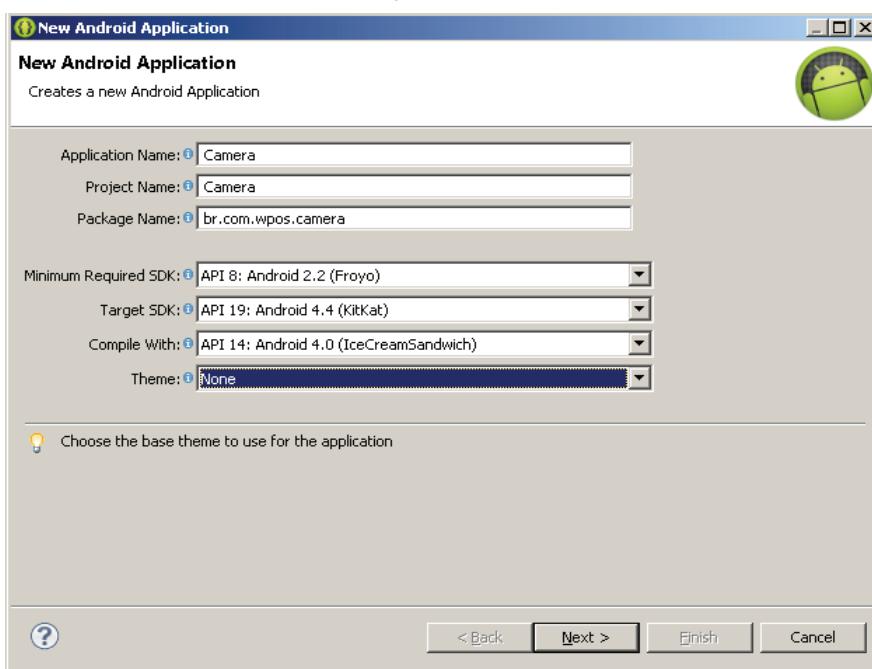
Vamos agora explorar outras possibilidades de nosso dispositivo móvel: o uso da câmera. O exemplo a seguir nos permite atirar o software de câmera, bater uma foto e exibi-la em nossa aplicação. Para tanto, faremos uso do elemento/widget conhecido como ImageView.

Criando um novo Projeto Android

Começamos criando um novo projeto do tipo aplicação Android, em nosso Eclipse ADT. Por não se tratar de recursos muito avançados, o requisito mínimo pode ser o FrozenYogurt (Froyo 2.2). Uma das razões a favor de um requisito mínimo tão baixo, além de atingir uma fatia de mercado maior, é a facilidade em se testar em emuladores com versões de Android que, embora sejam antigas, são extremamente mais leves, tornando os testes em um AVD mais fáceis. É recomendável, entretanto, uma vez que a aplicação esteja pronta, testar em várias versões de Android mais recentes e, se possível, testar também em alguns dispositivos físicos reais.

Crie um projeto no Eclipse ADT, clicando na opção New >Android Project:

Figura 54. Criação do projeto Câmera.



Fonte: próprio autor.

Criando um layout para Main Activity

É necessário fazer a diagramação visual da atividade. Seja no editor gráfico ou em XML, adicionaremos um rótulo e uma ImageView para receber a foto que será realizada pela câmera. Este widget terá o ícone padrão do Android inicialmente.

Texto 24: O arquivo activity_main.xml (Parte 1 de 2).

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

Texto 26: O arquivo activity_main.xml (Parte 2 de 2).

```
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="34dp"
    android:layout_marginTop="36dp"
    android:contentDescription="@string/hello_world"
    android:src="@drawable/ic_launcher" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_alignRight="@+id/imageView"
    android:text="@string/mensagem_do_clique"
    android:textAppearance="?android:attr/textAppearanceLarge"
/>
```

</RelativeLayout>

Não podemos nos esquecer da internacionalização da interface. Abriremos o arquivo strings.xml em res/values para traduzir os termos de interface necessários:

Texto 25: O arquivo strings.xml em res/values.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Camera</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="mensagem_do_clique">Clique na imagem para
    acionar a câmera</string>
</resources>
```

Obtendo o seguinte resultado:

Figura 55. Editor gráfico de Main Activity do Projeto Câmera.



Fonte: próprio autor.

A programação de MainActivity.java

A programação começa com as bibliotecas necessárias para as ações previstas, que é acionar a câmera, bater uma foto e trazer a foto de volta, para ser exibida no ImageView:

Texto 27: O arquivo MainActivity.java (Parte 1 de 3).

```
package br.com.wpos.camera;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageView;

public class MainActivity extends Activity {

    // Atributo que representará o ImageView da foto
    ImageView imagem;
```

Ao criar a atividade, programaremos as seguintes ações: relacionar o ImageView com o atributo que o representa e programar o evento de clique dele: ao clicar, chamará a função open() (apresentada adiante):

Texto 28: O arquivo MainActivity.java (Parte 2 de 3).

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

    //Captura o ImageView da tela e relaciona com o atributo.
    imagem = (ImageView) findViewById(R.id.imageView);
    imagem.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            //Chamada para a função que abrirá a camera
            open();
        }
    });
}
```

```
public void open() {  
    // Intent que chamara o aplicativo da camera  
    Intent intent = new Intent(android.provider.MediaStore.  
ACTION_IMAGE_CAPTURE);  
    startActivityForResult(intent, 0);  
}
```

A programação se encerra com o acionamento do software de câmera e a programação do seu retorno – ao salvar a foto, os dados da foto são devolvidos ao nosso aplicativo, que exibirá a foto no ImageView:

Texto 29: O arquivo MainActivity.java (Parte 3 de 3).

```
public void open() {  
    // Intent que chamará o aplicativo da camera  
    Intent intent = new Intent(android.provider.MediaStore.  
ACTION_IMAGE_CAPTURE);  
    startActivityForResult(intent, 0);  
}  
  
@Override  
protected void onActivityResult(int requestCode, int result-  
Code, Intent data) {  
    // Programando o retorno do aplicativo de câmera  
    super.onActivityResult(requestCode, resultCode, data);  
    // Pega a foto da camera e abre o ImageView como Bitmap  
    Bitmap bp = (Bitmap) data.getExtras().get("data");  
    imagem.setImageBitmap(bp);  
}  
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu; this adds items to the action bar if  
    it is present.  
    getMenuInflater().inflate(R.menu.main, menu);  
    return true;  
}  
}
```

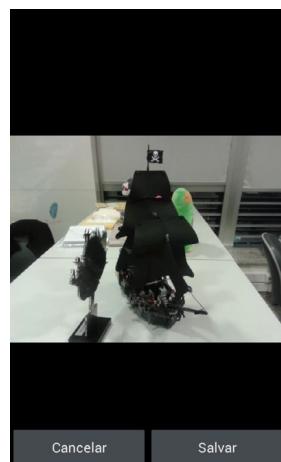
Testaremos o aplicativo aqui utilizando um dispositivo físico, ligado via cabo usb e configurado usando Samsung Kies (detalhes em um capítulo posterior):

Figura 56. Projeto Câmera em funcionamento (Parte 1 de 3).



Fonte: próprio autor.

Figura 57. Projeto Câmera em funcionamento (Parte 2 de 3).



Fonte: próprio autor.

Figura 58. Projeto Câmera em funcionamento (Parte 3 de 3).



Fonte: próprio autor.

CAPÍTULO 10

Exemplo Alarme

Agora que conhecemos um pouco mais sobre programação em ambiente Android, chegou a hora de um exemplo um pouco mais elaborado, explorando outros elementos de formulário e até mesmo outros recursos presentes nesta arquitetura.

O exemplo a seguir é um tradicional alarme de eventos. Criaremos um formulário da qual nosso usuário poderá informar o evento (“Tomar um Remédio”, “Reunião com o Marketing” etc.), assim como sua data e hora. Quando a data e hora do evento forem atingidas, emitiremos um som de alarme, acompanhada de uma notificação alertando para qual foi o evento disparado.

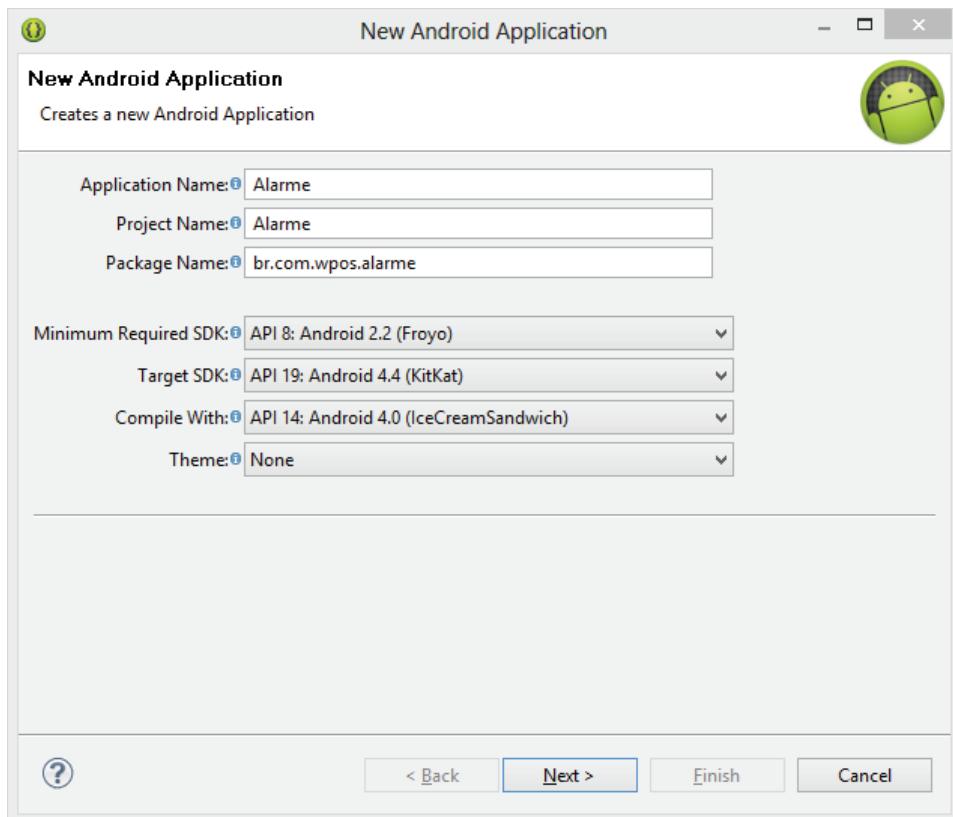
Para atingir este objetivo, usaremos elementos novos de formulário (como o DatePicker e o TimePicker, práticos e muito úteis em demandas como esta), assim como o Media Player que nos permitirá emitir um som no formato MP3.

Criando um novo Projeto Android

Começamos criando um novo projeto do tipo aplicação Android, em nosso Eclipse ADT. Por não se tratar de recursos muito avançados, o requisito mínimo pode ser o FrozenYogurt (Froyo 2.2). Uma das razões a favor de um requisito mínimo tão baixo, além de atingir uma fatia de mercado maior, é a facilidade em se testar em emuladores com versões de Android que, embora sejam antigas, são extremamente mais leves, tornando os testes em um AVD mais fáceis. É recomendável, entretanto, uma vez a aplicação esteja pronta, testar em várias versões de Android mais recentes e, se possível, testar também em alguns dispositivos físicos reais.

Crie um projeto no Eclipse ADT, clicando na opção New >Android Project:

Figura 59. Criação do projeto Alarme.



Fonte: próprio autor.

Recomendamos a personalização do ícone do projeto. Dentro dos cliparts, temos alguns ícones interessantes, entre eles o de um relógio no estilo alarme.

Figura 60. Personalizando o ícone da aplicação.



Fonte: próprio autor.

Criaremos a aplicação com uma atividade (activity) padrão, chamada de Main_Activity.

Trabalhando o layout da aplicação

É necessário fazer a diagramação visual da atividade. Seja no editor gráfico ou em XML, adicionaremos rótulos para os campos deste formulário, que deverá contar com:

- » uma caixa de texto (EditText) para receber o nome do Evento ou compromisso;
- » um DatePicker, para que o usuário possa informar facilmente a data do evento;
- » um TimePicker, para que ele também possa informar a hora e minuto deste alarme;
- » um botão para que o alarme seja armado com a data e hora acordadas.

O código-fonte resultante deve ser mais ou menos assim.

Texto 30: O arquivo de layout activity_main.xml (Parte 1 de 2)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    tools:context="br.com.wpos.alarme.MainActivity">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="10dp"><!-- uma borda superior é adequada -->

        <TextView
            android:id="@+id/lblEvent"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/lblEvent"
            android:paddingLeft="10dp"/>
```

```
<EditText  
    android:id="@+id/txtEvent"  
    android:inputType="text"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:ems="10"/><!-- caixa de texto para nome do evento -->  
  
</LinearLayout>
```

Texto 31: O arquivo de layout activity_main.xml (Parte 2 de 2)

```
<LinearLayout  
    android:orientation="horizontal"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:paddingTop="5dp">  
  
    <TextView  
        android:id="@+id/lblDate"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/lblDate"  
        android:paddingLeft="10dp"/>  
  
<DatePicker<!-- DatePicker para coleta da data do evento -->  
    android:id="@+id/datAlarm"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"/>  
</LinearLayout>  
  
<LinearLayout  
    android:orientation="horizontal"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:paddingTop="5dp">  
  
    <TextView  
        android:id="@+id/lblTime"  
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="@string/lblTime"
    android:paddingLeft="10dp"/>

<TimePicker<!-- TimePicker para coleta da data do evento -->
    android:id="@+id/timAlarm"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

```

</LinearLayout>

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingTop="30dp"
    android:gravity="center">
```

<Button<!-- Button para armar o alarme -->

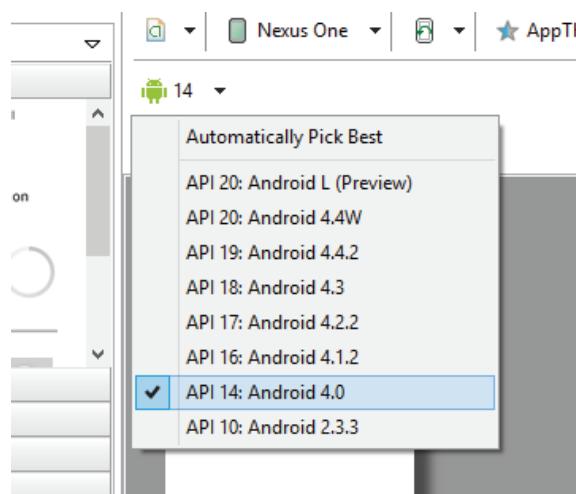
```
    android:id="@+id/setAlarm"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClickSetAlarm"
    android:text="@string/set_alarm"/>

```

</LinearLayout>

```
</LinearLayout>
```

Figura 61. Definindo a API para referência de interface gráfica.

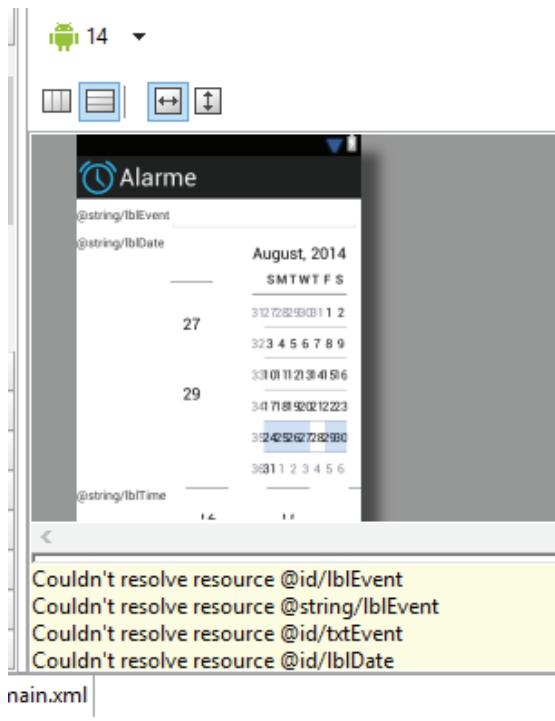


Fonte: próprio autor.

Não se esqueça de definir a API para renderização gráfica no editor gráfico do layout. Utilize a versão 4.0 (Ice Cream Sandwich) definida no campo “Compile With” na criação do projeto.

E teremos como resultado:

Figura 62. Editor gráfico do layout de Main_Activity.



Fonte: próprio autor.

Repare pelos erros apresentados que as strings (@string/...) não foram definitivamente definidas ainda. Devemos então acessar o arquivo strings.xml em res/values e definí-las:

Texto 32: O arquivo de res/values/strings.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">Alarme</string>
<string name="action_settings">Configurações</string>
<string name="set_alarm">Registrar Alarme</string>
<string name="lblEvent">Evento: </string>
<string name="lblDate">Data: </string>
<string name="lblTime">Hora: </string>
</resources>
```

É recomendado verificar o layout rodando o aplicativo, já que o editor gráfico não apresenta um bom resultado, ao menos quando usamos um DatePicker ou TimePicker. Devemos optar pelo seguinte resultado:

Figura 63. O layout da atividade visto a partir de um AVD.



Fonte: próprio autor.

Programando o alarme

Embora a interface seja promissora, de nada servirá sem a devida programação. O primeiro passo é capturar os elementos gráficos da aplicação (também conhecidos aqui com Views ou Widgets da atividade), recuperar os valores informados e, no caso da data e hora, convertê-la para milisegundos, unidade esperada pelo Alarm Manager.

O arquivo `MainActivity.java` presente em no pacote `br.com.wpos.alarme` começa com uma série de referências à bibliotecas Java padrão e bibliotecas Android, todas necessárias para o que virá pela frente.

Texto 33: O arquivo `MainActivity.java` (Parte 1 de 7).

```
package br.com.wpos.alarme;

import java.io.IOException;
import java.util.GregorianCalendar;

import android.app.AlarmManager;
import android.app.NotificationManager;
import android.app.PendingIntent;
```

```

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.res.AssetFileDescriptor;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.support.v4.app.NotificationCompat;
import android.support.v4.app.TaskStackBuilder;
import android.support.v7.app.ActionBarActivity;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.TimePicker;

```

O código segue com a criação de atributos. Entre eles, a presença de algo novo: um BroadCast Receiver. Um dispositivo móvel, durante o seu funcionamento, pode realizar diversas operações, como por exemplo, iniciar uma ligação, finalizar uma ligação, receber um SMS, ou seja, a todo momento o sistema operacional Android está realizando alguma operação. Estes eventos são avisados a todas aplicações e, se desejarem e forem programadas para isso, podem responder a essas ações. O componente utilizado para isso é o Broadcast Receiver.

Texto 34: O arquivo MainActivity.java (Parte 2 de 7).

```

public class MainActivity extends ActionBarActivity {

    // será usado para registrar o gerenciador do alarme
    PendingIntent pendingIntent;
    // usado para armazenar a instância de Alarm Manager em
    // funcionamento
    AlarmManager alarmManager;
    //O BroadcastReceiver atua como uma função de "Callback"
    //para o //evento de Alarm Manager.
    BroadcastReceiver mReceiver;

    DatePicker datAlarm;

```

```
TimePicker timAlarm;
EditText txtEvent;
MediaPlayer player; //será utilizado para disparar o aviso
sonoro
```

Na sequência, no ato a criação da atividade (método `onCreate()`) os elementos do formulário devem ser localizados e suas referências armazenadas nos atributos.

Texto 35: O arquivo `MainActivity.java` (Parte 3 de 7).

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

    //Register AlarmManager Broadcast receive.
    RegisterAlarmBroadcast();

    datAlarm = (DatePicker) findViewById(R.id.datAlarm);
    timAlarm = (TimePicker) findViewById(R.id.timAlarm);
    txtEvent = (EditText) findViewById(R.id.txtEvent);
}
```

Como próximo passo, a programação a ser realizada no ato do clique do Botão do formulário. O dia, mês e ano do DatePicker, assim como a hora e minuto do TimePicker serão agrupados em um objeto do tipo Gregorian Calendar. E este objeto que transformará nossa data e hora em milissegundos, unidade esperada pelo Alarm Manager.

Texto 36: O arquivo `MainActivity.java` (Parte 4 de 7).

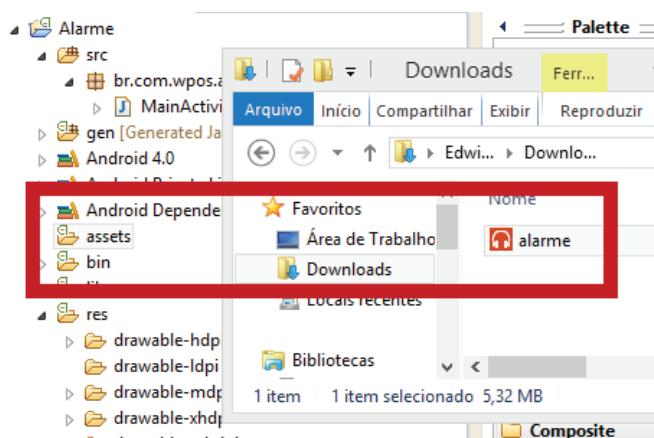
```
public void onClickSetAlarm(View v)
{
    GregorianCalendar data = new
    GregorianCalendar(datAlarm.getYear(), datAlarm.getMonth(), dat-
    Alarm.getDayOfMonth(), timAlarm.getCurrentHour(), timAlarm.get-
    CurrentMinute());
    alarmManager.set(AlarmManager.RTC_WAKEUP, data.getTimeIn-
    millis() , pendingIntent );
}
```

Conforme citado anteriormente, quando a data e hora armada pelo alarme for atingida, um aviso sonoro deve ser emitido. Escolha um arquivo de som no formato MP3 de sua

preferência, ou baixe algum som que se assemelhe a um alarme em um mecanismo de busca na Internet. Renomeie este arquivo para alarme.mp3.

O arquivo alarme.mp3 precisa ser incorporado ao projeto. O local mais indicado para se armazenar este arquivo é a pasta assets, pré-existente na estrutura de nosso projeto Android. Copie o arquivo em questão para dentro desta pasta – recomendamos que o faça arrastando o arquivo diretamente pelo eclipse: esta maneira, o projeto será atualizado com a referência do arquivo.

Figura 64. Arrastando o arquivo do Windows Explorer para a pasta assets do projeto.



Fonte: próprio autor.

Continuamos a programação com o recebimento (`onReceive()`) do Broadcast Receiver, fazendo com que emita o aviso sonoro e realize uma notificação de sistema.

Texto 37: O arquivo `MainActivity.java` (Parte 5 de 7).

```
private void RegisterAlarmBroadcast()
{
    // Esta é uma função de CallBack quando o alarme
    // responder através do BroadcastReceiver
    mReceiver = new BroadcastReceiver()
    {
        @Override
        public void onReceive(Context context, Intent
intent)
        {
            emitirAvisoSonoro();
            gerarNotification(context, txtEvent.get-
Text().toString());
        }
    }
}
```

```

    } ;

    // registrando o alarm broadcast aqui
    registerReceiver(mReceiver, new IntentFilter("br.com.
wpos.alarme") );
    pendingIntent = PendingIntent.getBroadcast( this, 0,
new Intent("br.com.wpos.alarme"),0 );
    alarmManager = (AlarmManager) (this.getSystemService(
Context.ALARM_SERVICE ) );
}

public void emitirAvisoSonoro()
{
    try {
        // Localizando o MP3 em assets
        AssetFileDescriptorafd = getAssets().openFd("alarme.mp3");
        player = newMediaPlayer();
        player.setDataSource(afd.getFileDescriptor(),
afd.getStartOffset(), afd.getLength());
        player.prepare();
        player.start();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

Prosseguimos programando o método gerarNotification(), para emitir uma notificação de sistema, utilizando como mensagem o nome do evento informado no formulário.

Texto 38: O arquivo MainActivity.java (Parte 6 de 7).

```

public void gerarNotification(Context context, String men-
sagem)
{
    NotificationCompat.Builder mBuilder = new Notification-
Compat.Builder(context);
    mBuilder.setSmallIcon(R.drawable.ic_launcher);
    mBuilder.setContentTitle("Alarme!");
}

```

```

mBuilder.setContentText(mensagem);

Intent resI = new Intent(context, MainActivity.
class);

TaskStackBuilderstackBuilder = TaskStackBuilder.
create(context);
stackBuilder.addParentStack(MainActivity.class);
stackBuilder.addNextIntent(resI);

PendingIntentresultPendingIntent = stackBuilder.get-
PendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);

mBuilder.setContentIntent(resultPendingIntent);
NotificationManagermNotificationManager = (Notification-
Manager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(1, mBuilder.build());
}

```

O programação de MainActivity.java se encerrar com os métodos padrões onCreateOptionsMenu() e onOptionsItemSelected().

Texto 39: O arquivo MainActivity.java (Parte 7 de 7).

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}

```

Testando a aplicação de Alarme

O exemplo termina com testes do novo aplicativo no AVD:

Figura 65. Marcando consulta médica.



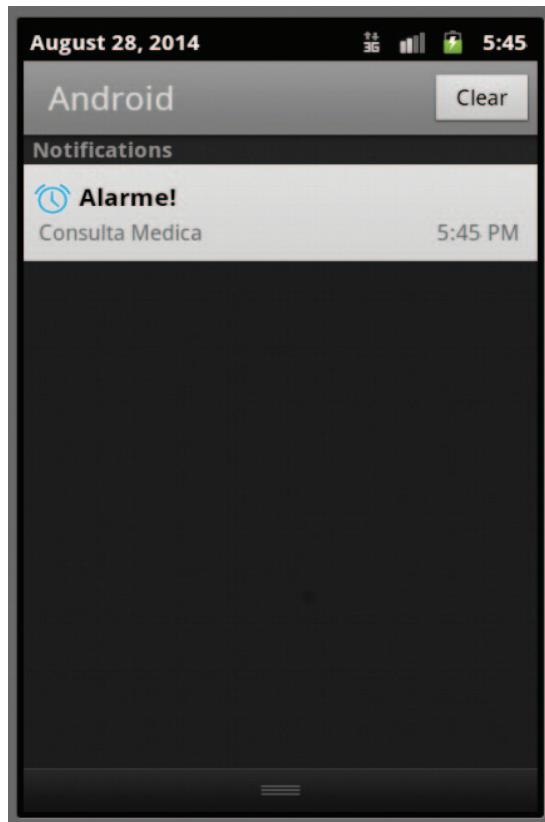
Fonte: próprio autor.

Figura 66. Som funcionando e aviso de notificação.



Fonte: próprio autor.

Figura 67. Notificação do alarme.



Fonte: próprio autor.

CAPÍTULO 1

Testando em um dispositivo real

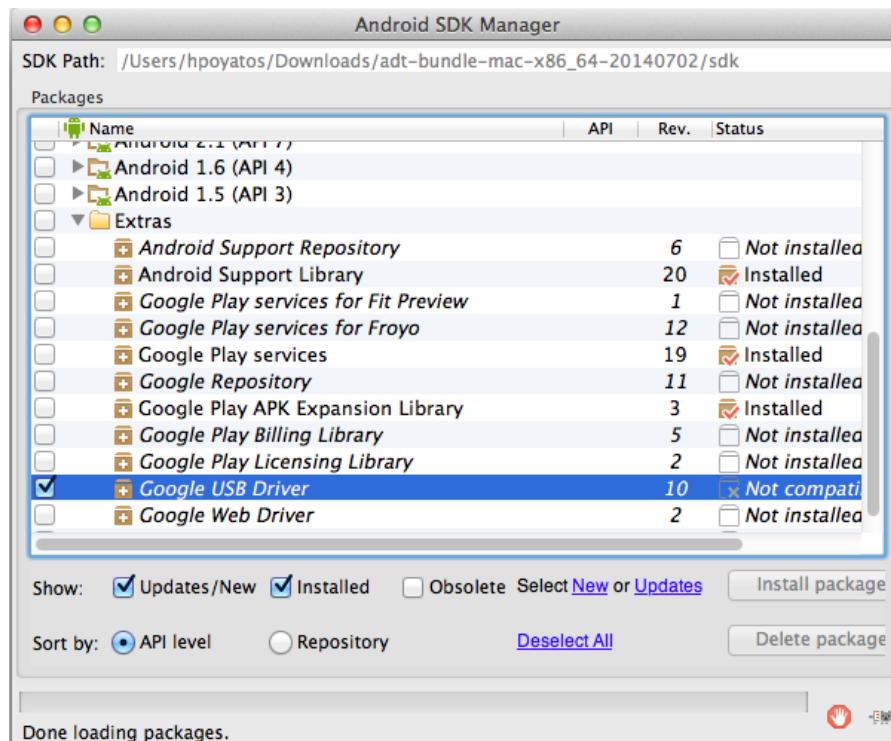
Vamos a um exemplo mais elaborado, que consola tudo aquilo visto até agora.

Instalação dos drivers USB do Android SDK

O primeiro passo é instalar os drivers USB do Google por meio do Android SDK.

Observação: esta opção não é possível em um Mac.

Figura 68. Instalando o Google USB Driver no Android SDK Manager.

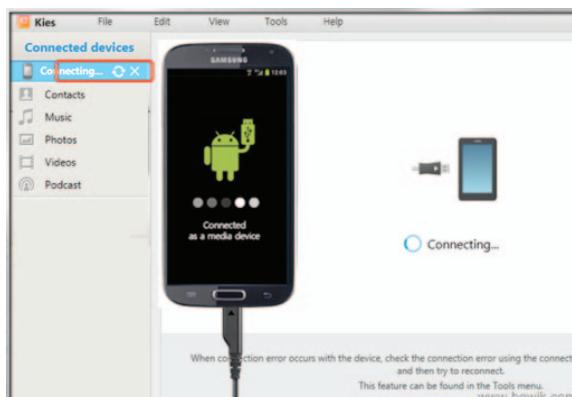


Fonte: próprio autor.

Instalação dos drivers do dispositivo móvel

A Samsung possui um aplicativo chamado Samsung Kies que, além de possibilitar backups de dados e instalação dos novos firmwares, instala drivers do dispositivo Samsung, permitindo testar suas aplicações diretamente no celular, a partir do Eclipse ADT.

Figura 69. Samsung Kies.



Fonte: Disponível em <http://howikis.com/images/c/ca/10_how_to_update_samsung_galaxy_s4_using OTA_using_KIES_and_ODI.PNG>. Acessado em 24 de agosto de 2014.

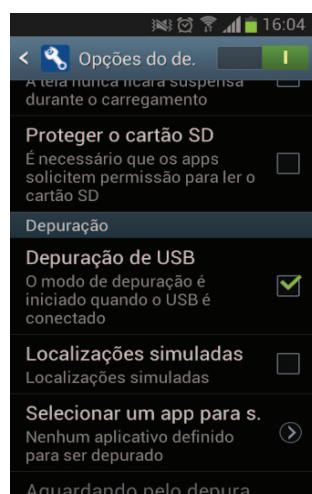
A fabricante Motorola possui produto similar para o mesmo fim.

Habilitar a depuração do dispositivo móvel

É sempre uma boa opção habilitar a depuração via USB a partir do seu dispositivo móvel. Desta forma, ao testar sua aplicação no dispositivo, quaisquer falhas ocorridas serão enviadas via USB e serão exibidas no LogCat do Eclipse ADT.

Para isso, no ícone “Ajustes” de seu Android, na opção “Opções do desenvolvedor”, habilite a “Depuração de USB”.

Figura 70. Habilitando a depuração via USB.



Fonte: próprio autor.

CAPÍTULO 2

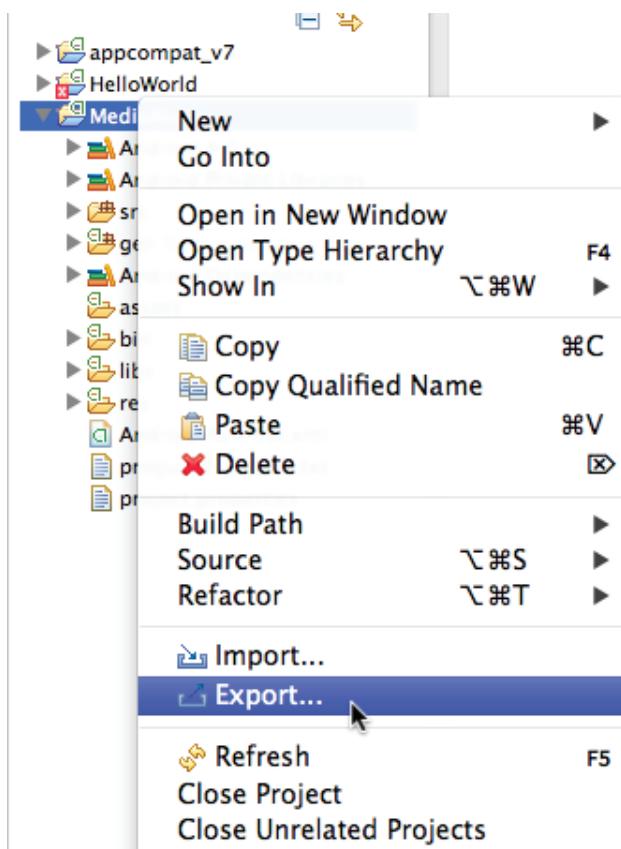
Gerando um pacote de instalação da app

Uma das partes mais interessantes que a programação em Android nos proporciona é a facilidade em gerar pacotes de instalação do seu aplicativo, que poderão ser instalados em qualquer dispositivo dentro das especificações que estipulamos em nosso manifesto.

Exportando o projeto

Com o botão direito do mouse no projeto, selecione a opção “Export”.

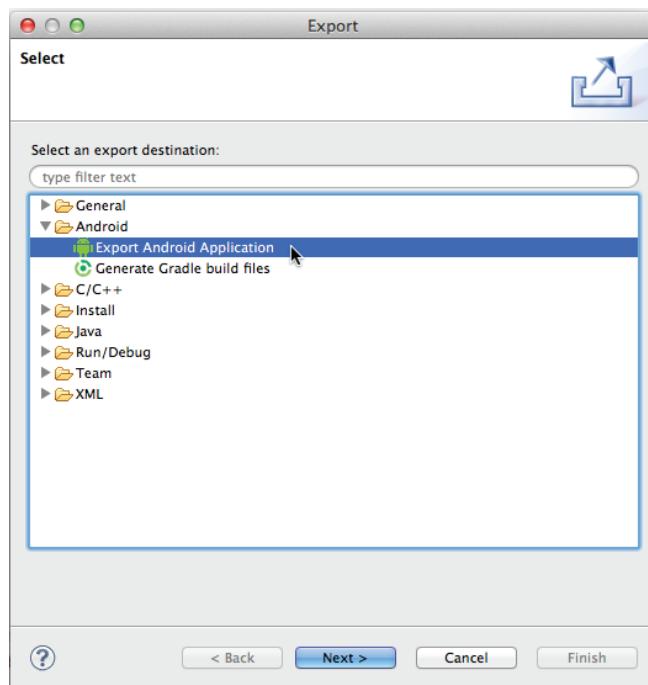
Figura 71. Exportando seu projeto Android



Fonte: próprio autor.

Selecione a opção “ExportAndroidApplication”.

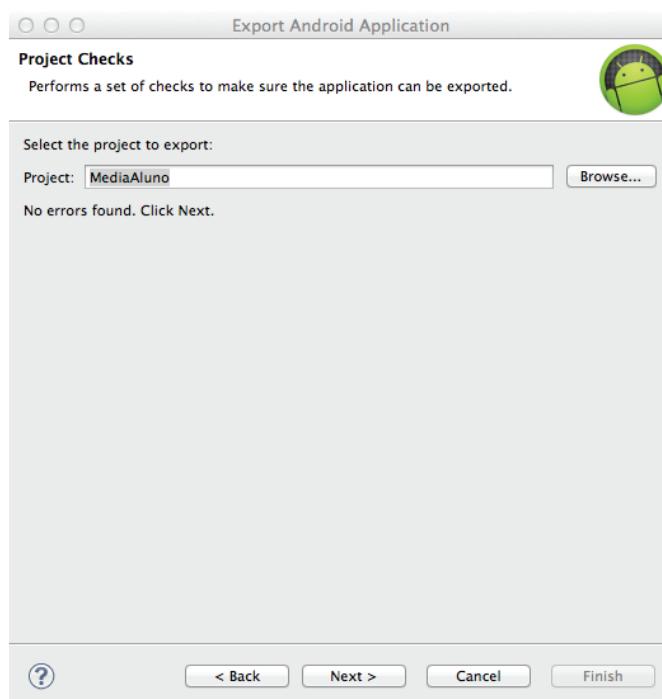
Figura 72. Seleção para Exportar Android Application.



Fonte: próprio autor.

Na tela seguinte, deve-se selecionar qual projeto será exportado. Como o início do processo foi feito com o botão direito do mouse no projeto correto, nada deve ser alterado.

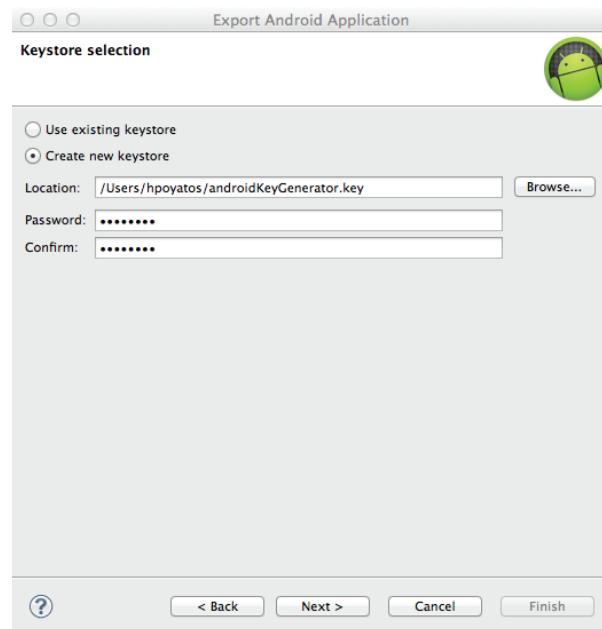
Figura 73. Selezionando o projeto Android a ser exportado.



Fonte: próprio autor.

Toda aplicação Android precisa ser assinada pela empresa ou indivíduo desenvolvedor. Para criar uma chave nova, especifique um caminho para armazená-la (arquivo .key) e defina uma senha para ela.

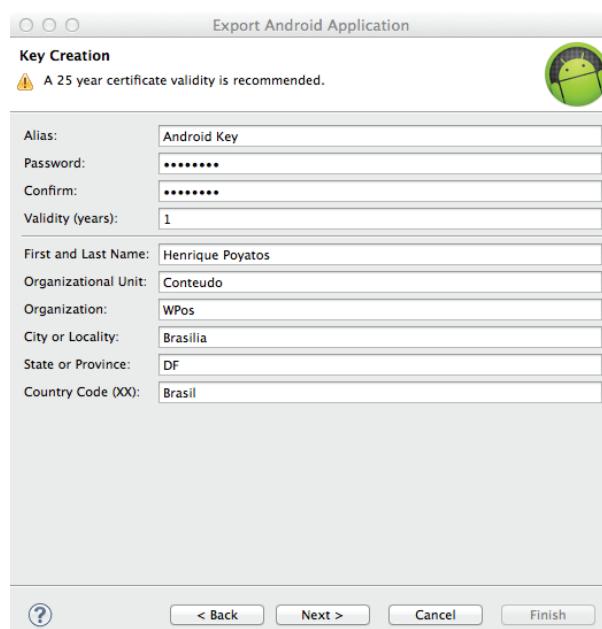
Figura 74. Criando uma chave criptográfica para assinatura



Fonte: próprio autor.

Outras informações precisam ser informadas, como indivíduo, empresa, localização etc.

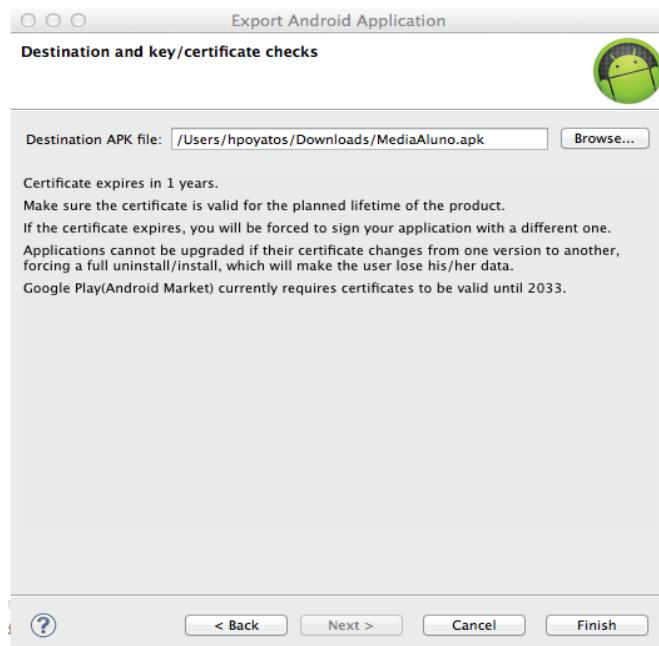
Figura 75. Criando uma chave criptográfica para assinatura (2).



Fonte: próprio autor.

Ao final do processo, definiremos a localização do arquivo .APK, o pacote de instalação contendo nosso aplicativo.

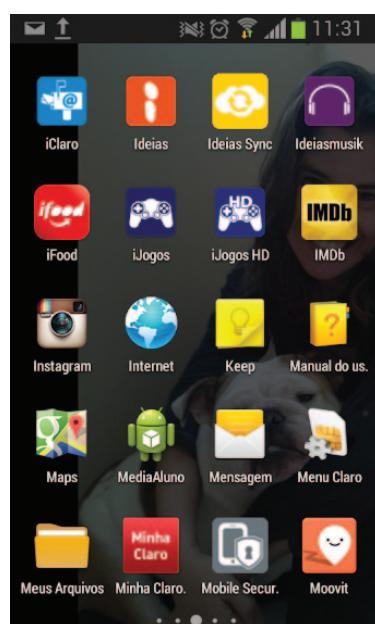
Figura 76. Definindo o destino do arquivo .APK.



Fonte: próprio autor.

O arquivo .APK gerado pode ser enviado via Wi-fi, USB, cartão microSD, qualquer que seja o meio. Uma vez instalado, será como um de seus aplicativos.

Figura 77. Definindo o destino do arquivo .APK.



Fonte: próprio autor.



Para (não) finalizar

O desenvolvimento de aplicativos para plataformas móveis faz parte de um futuro irreversível: os costumes da nova sociedade da informação vão ao encontro da mobilidade, cada vez sem limites. Por esta razão, recomendamos que seus estudos continuem nas seguintes vertentes:

- » Programação avançada envolvendo banco de dados SQLite: uma boa aplicação necessariamente precisa armazenar informações para uso posterior. O SQLite serve justamente para este fim.
- » Programação avançada utilizando Google API: para integração com ferramentas como Google Maps, Agenda, Gmail, entre outros.
- » Comunicações utilizando WebServices: seu aplicativo pode ser uma interface para um sistema maior, que está na nuvem. A maneira mais comum de realizar o acesso será por meio de WebServices.
- » Desenvolvimento em iOS utilizando Objective C no Xcode (Macintosh): trata-se do 2º maior mercado de portáteis, uma fatia de mercado que não deve ser ignorada.
- » Desenvolvimento em Android, iOS e Windows Phone utilizando Phonegap ou Apache Cordova: programar, uma única vez, utilizando HTML5 e gerando pacotes para os três principais sistemas operacionais portáteis pode ser extremamente atraente.

Referências

Android Market. Disponível em: <<http://market.android.com/>>. Acessado em: 9 mar. 2014.

G1 No mundo. 4 a cada 5 smartphones vendidos rodam o sistema Android. Disponível em: <<http://g1.globo.com/tecnologia/noticia/2013/10/no-mundo-4-cada-5-smartphones-vendidos-rodam-o-sistema-android.html>>. Acessado em: 9 mar. 2014.

HIGA, Paulo. **Android ultrapassa 50% de marketshare;** Samsung vende mais. Disponível em: <<http://tecnoblog.net/97184/android-market-share/>>. Acessado em: 9 mar. 2014.

Industry Leaders Announce Open Platform for Mobile Devices. Open Handset Alliance (2007-11-05). Disponível em: <http://www.openhandsetalliance.com/press_110507.html>. Acessado em: 9 mar. 2014.

LECHETA, Ricardo R. **Google Android -- Aprenda aprender a Criar Aplicações** aplicações Para para Dispositivos dispositivos Móveis móveis Com com o AndroidSdkandroidSdk. 2. ed. São Paulo: Novatec, 2010.

PEREIRA, Lúcio Camilo Oliva; DA SILVA, Michel Lourenço. **Android Para Desenvolvedores.** 2. ed. Rio de Janeiro: Brasport, 2012.

Plug-in Eclipse – Android (ADT). Disponível em: <<http://developer.android.com/sdk/eclipse-adt.html>>. Acessado em: 9 mar. 2014.

ROGERS, Rick; LOMBARDO, John; MEDNIEKS, Zigurd; MEIKE, Blake. **Desenvolvimento de aplicações Android.** 1. ed. São Paulo: Novatec, 2009.

Segurança para Android. Disponível em: <<http://www.ibm.com/developerworks/library/x-androidsecurity/>>. Acessado em: 9 mar. 2014.

Site para desenvolvedores Android. Disponível em: <<http://developer.android.com/>>. Acessado em: 9 mar. 2014.