

**INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**
FLUMINENSE
Campus Campos-Centro

Secretaria de Educação
Profissional e Tecnológica

Ministério
da Educação

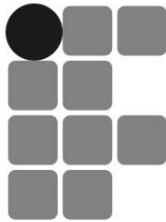


BACHARELADO EM SISTEMAS DE INFORMAÇÃO

EDNA CAMPOS VIANA
IGOR DE CASTRO PONTES

SISTEMA OPERACIONAL ANDROID
DESMISTIFICANDO O DESENVOLVIMENTO DE APLICATIVOS

Campos dos Goytacazes/RJ
2012



**INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**
FLUMINENSE
Campus Campos-Centro

Secretaria de Educação
Profissional e Tecnológica

Ministério
da Educação



BACHARELADO EM SISTEMAS DE INFORMAÇÃO

EDNA CAMPOS VIANA
IGOR DE CASTRO PONTES

SISTEMA OPERACIONAL ANDROID DESMISTIFICANDO O DESENVOLVIMENTO DE APLICATIVOS

Trabalho de conclusão de curso apresentado
ao Instituto Federal Fluminense como requi-
sito parcial para conclusão do Bacharelado
em Sistemas de Informação.

Orientador: Prof. Fábio Duncan de Souza

Campos dos Goytacazes/RJ
2012

EDNA CAMPOS VIANA
IGOR DE CASTRO PONTES

SISTEMA OPERACIONAL ANDROID
DESMISTIFICANDO O DESENVOLVIMENTO DE APLICATIVOS

Trabalho de conclusão de curso apresentado
ao Instituto Federal Fluminense como requi-
sito parcial para conclusão do Bacharelado
em Sistemas de Informação.

Aprovada em 26 de novembro de 2012

Banca avaliadora:

Prof. Fábio Duncan de Souza (Orientador)
Mestre em Pesquisa Operacional e Inteligência Computacional / UCAM Campos
Instituto Federal de Educação, Ciência e Tecnologia Fluminense / Campus
Campos Centro

Prof. Rogério Atem de Carvalho, D. Sc.
Doutor em Ciências de Engenharia / UENF
Instituto Federal de Educação, Ciência e Tecnologia Fluminense / Campus
Campos Centro

Prof. Rogério de Avellar Campos Cordeiro
Mestre em Engenharia de Produção / UENF
Instituto Federal de Educação, Ciência e Tecnologia Fluminense / Campus Guarus

Às nossas famílias,

com amor...

AGRADECIMENTOS

Queremos agradecer a Deus, pois sem ele nada seria possível, nossas famílias que nos apoiam em todas decisões, aos professores que nos apoiaram e incentivaram, permitindo o desenvolvimento deste trabalho.

Os pequenos atos que se executam
são melhores que todos aqueles
grandes que apenas se planejam.

George C Marshall

RESUMO

A demanda por aplicativos para dispositivos móveis aumentou nos últimos anos devido aos avanços nas tecnologias de hardware e nas plataformas de desenvolvimento. Este trabalho propõe apresentar o Sistema Operacional Android, criado para atender as necessidades deste mercado, apresentando a sua estrutura e componentes. Além disso, visa demonstrar a instalação e a configuração do ambiente de desenvolvimento para seus aplicativos, assim como apresentar como criar sua base de dados, layout e como gerenciar a interface entre estes. Por fim, apresenta um estudo de caso que objetiva aplicar os conceitos estudados na criação de um aplicativo, destacando particularidades e desafios encontrados durante o processo de desenvolvimento.

PALAVRAS-CHAVE: Android, Desenvolvimento, Aplicativo, Dispositivos Móveis.

ABSTRACT

The demand for mobile applications has increased in recent years due to the advances in hardware technologies and development platforms. This work aims at proposing to presenting the Android Operating System, created to meet the needs of this market, showing its structure and components. Furthermore, it aims to demonstrate the installation and configuration of development environment for your applications, and present how to create your database and layout and how to manage the interface between them. Finally, it presents a case study that aims to apply the concepts studied in the creation of an application and find peculiarities and challenges encountered during the development process.

KEYWORDS: Android, development, application, mobile devices.

LISTA DE CÓDIGOS

2.1	Estrutura geral do arquivo <i>AndroidManifest.xml</i>	48
4.1	<i>Activity</i> exemplo criada pelo Eclipse	64
4.2	A classe R	65
4.3	Pasta res	66
4.4	<i>String</i>	68
4.5	<i>String Array</i>	68
4.6	<i>Plurals</i>	68
4.7	<i>LinearLayout</i>	70
4.8	Botão com alguns de seus atributos básicos.	70
4.9	Nova identificação de recurso	71
4.10	Recurso identificado na Classe R	71
4.11	Identificação como um recurso Android	71
4.12	Criação de um elemento <i>TextView</i>	73
4.13	Criação de um elemento <i>EditText</i>	73
4.14	Criação de um elemento <i>Button</i>	73
4.15	Criação de um <i>RadioGroup</i>	73
4.16	Criação de um <i>CheckBox</i>	74
4.17	<i>Spinner</i>	74
4.18	<i>Array</i> de <i>Strings</i> adicionado ao <i>Spinner</i>	74
4.19	Criação de um <i>AutoCompleteTextView</i>	75
4.20	Criação de um elemento <i>ImageView</i>	75
4.21	Código básico para implementação do ciclo de vida da atividade	77
4.22	Exemplo de acesso a recurso da Classe R.	77
4.23	Criação de link entre recurso e objeto da atividade.	78
4.24	Abertura de uma nova atividade.	79
4.25	Acesso a dados de uma intenção e fechamento de atividade.	79
4.26	Acesso e edição de texto de um elemento <i>TextView</i>	79
4.27	Acesso e edição de texto de um elemento <i>EditText</i>	79

4.28	Evento <i>OnClick</i> de um botão	80
4.29	Descobrir qual <i>RadioButton</i> em um <i>RadioGroup</i> está marcado.	80
4.30	Descobrir se um <i>CheckBox</i> está marcado	80
4.31	Como adicionar uma lista a um <i>Spinner</i> em tempo de execução.	81
4.32	Como adicionar uma lista a um <i>Spinner</i> em tempo de execução.	81
4.33	Setar uma imagem em um elemento <i>ImageView</i>	81
4.34	Parte do Manifest do projeto obrafacil	82
4.35	Criando uma subclasse com designação de pacote.	84
4.36	Criando uma subclasse sem designação de pacote.	84
4.37	Definição de múltiplas ações em um filtro de intenção.	84
4.38	Algumas das permissões padrão Android.	85
4.39	Utilizando permissões em uma atividade.	85
4.40	Classe Produto	88
4.41	Variáveis da classe <i>DbAdapter</i>	88
4.42	Classe privada <i>DatabaseHelper</i> - Parte 1	88
4.43	Classe privada <i>DatabaseHelper</i> - Parte 2	89
4.44	Classe privada <i>DatabaseHelper</i> - Parte 3	89
4.45	Métodos <i>open</i> e <i>close</i>	89
4.46	Classe BasicoDAO	90
4.47	Cabeçalho da classe <i>ProdutoDAO</i>	91
4.48	<i>Script</i> de criação da tabela <i>Produtos</i>	91
4.49	Conversão de objeto <i>Produto</i> para objeto <i>ContentValues</i>	91
4.50	Cadastro de <i>Produto</i>	92
4.51	Remoção de <i>Produto</i>	92
4.52	Atualização de <i>Produto</i>	92
4.53	Consultar todos os <i>Produtos</i>	92
4.54	Consultar Produtos por Categoria	92
4.55	Conversão de objeto <i>Cursor</i> para objeto <i>Produto</i>	93
4.56	O arquivo <i>AndroidManifest.xml</i> do projeto de teste	98
4.57	Construtor da classe de teste	100
4.58	Método de pré-configuração de variáveis de teste	100
4.59	Variáveis da classe de teste	101

4.60	Teste de pré-condições	101
4.61	Inicialização da constante <i>Adapter_Count</i>	101
4.62	Começo da criação do método de teste <i>TestSpinnerUI</i>	102
4.63	Simulação de interação do usuário com o <i>Spinner</i>	102
4.64	Verificação dos dados obtidos	102
4.65	Criação de variáveis para o método <i>testSpinnerUI()</i> funcionar	103
5.1	Conversão de <i>Cursor</i> para um objeto Categoria	113
5.2	Arquivo XML com a definição do layout de cadastro de Categoria	114
5.3	<i>onCreateCategoria</i> Teste	115
5.4	Exemplo de vinculo da classe de teste com a classe do projeto principal.	116
5.5	Código de teste implementado.	116
5.6	Simulação de preenchimento de campos.	117

LISTA DE FIGURAS

1.1	HTC Dream G1 - primeiro aparelho com o Android	20
1.2	Aparelho com a versão 1.5 - Cupcake (INTERESSANTE, 2009)	21
1.3	Aparelho com a versão 1.6 - Donut (INTERESSANTE, 2009)	22
1.4	Aparelho com a versão 2.0 - Eclair (INTERESSANTE, 2009)	22
1.5	Aparelho com a versão 2.2 - Froyo (INTERESSANTE, 2009)	23
1.6	Aparelho com a versão 2.3 - Gingerbread (INTERESSANTE, 2009)	23
1.7	Aparelho com a versão 3.0 - Honeycomb (INTERESSANTE, 2009)	24
1.8	Aparelho com a versão 4.0 - Ice Cream Sandwich (ROQUEIROJP, 2012) .	25
1.9	Aparelho com a versão 4.1 - Jelly Bean (MURPH, 2012)	25
1.10	Versões do Android por utilização	26
1.11	Sistemas operados nos smartphones (FRAGA, 2012)	27
1.12	Sistemas operacionais por fabricantes (FRAGA, 2012)	27
1.13	Utilização dos sistemas operacionais no Brasil (TELETIME, 2012)	28
2.1	Sistema Operacional Android (ANDROID, 2012a)	29
2.2	Relacionamento da aplicação com o ambiente Android	34
2.3	Ciclo de vida de uma Atividade (ANDROID, 2012c)	37
2.4	Estados da atividade (PEREIRA; SILVA, 2009, p.12)	38
2.5	Hierarquia da classe <i>View</i> , (LECHETA, 2009, p.45)	41
2.6	Hierarquia <i>ViewGroup</i> , (ANDROID, 2012c)	42
2.7	Hierarquia dos <i>Widgets</i> Android (TEAM, 2011)	44
2.8	Métodos de retorno de chamada de um Serviço (ANDROID, 2012c) . . .	46
3.1	O emulador da versão 2.1 em execução	52
3.2	Instalação do <i>plugin ADT</i> - <i>Help > Install New Software</i>	55
3.3	Instalação do <i>plugin ADT</i> - Preencher dados do respositório	55
3.4	Instalação do <i>plugin ADT</i> - Marcar a opção <i>Developer Tools</i>	56
3.5	Configuração do Android SDK	57
3.6	Termos do contrato de uso do Android SDK	58
3.7	Novas funcionalidades do Eclipse	58

3.8	Interface do Android Sdk <i>Manager</i>	59
3.9	Interface do Android <i>Virtual Device Manager</i>	59
3.10	Criando novo dispositivo virtual	60
3.11	Escolha de plataforma.	61
3.12	Criação de um novo projeto.	61
3.13	Estrutura do Projeto Android.	62
4.1	Etapas para criação de uma aplicação Android	63
4.2	Tamanhos relativos a resolução suportada	67
4.3	Hierarquia da <i>View</i> com definição do <i>layout</i> de interface	69
4.4	Elementos gráficos	72
4.5	Ciclo de vida de uma atividade (ANDROID, 2012d)	76
4.6	Diagrama de classe do banco de dados	87
4.7	Fluxo da depuração	94
4.8	Depuração DDMS	95
4.9	Processo de teste padrão	97
4.10	Cadastro de classe de teste	99
4.11	Aba <i>JUnit</i> para testes bem sucedidos	103
4.12	Aba <i>JUnit</i> em caso de falha de um ou mais testes	104
4.13	Painel <i>Failure Trace</i> detalhando uma falha no teste	104
4.14	Modelo sintético do processo de desenvolvimento da aplicação	105
5.1	A tela inicial do aplicativo Obra Fácil	108
5.2	Telas para gerência de categorias de produtos	108
5.3	Menu que aparecerá caso uma categoria seja pressionada na lista	109
5.4	A lista de produtos cadastrados	109
5.5	Interface de cadastro e atualização de produtos	110
5.6	Telas para gerência de obras	110
5.7	Nova função presente no menu ao selecionar uma obra na lista	110
5.8	Listagens de Compras de uma Obra	111
5.9	Interface para adição de produtos a uma lista de compras	111
5.10	Telas para adição de produtos à lista de compras e finalização desta	112
5.11	Layout gerado pelo XML de cadastro de Categoria	115

LISTA DE TABELAS

2.1	<i>User Interface</i> (ANDROID, 2012c)	42
3.1	Janelas visualizadas na perspectiva DDMS (LECHETA, 2009, 53)	53
4.1	Resoluções de imagens.	67
4.2	Unidades dimensionais aceitas pelo Android.	72
4.3	<i>Tags</i> presentes no <i>AndroidManifest.xml</i>	83

SUMÁRIO

INTRODUÇÃO	18
1 INTRODUÇÃO AO ANDROID	19
1.1 Histórico	19
1.2 A Evolução	21
1.2.1 Versão 1.x	21
1.2.2 Versão 2.x	22
1.2.3 Versão 3.x	24
1.2.4 Versão 4.x	24
1.2.5 Um comparativo entre as versões	26
1.3 O Momento Atual	26
1.3.1 O cenário mundial	26
1.3.2 O cenário brasileiro	27
2 O SISTEMA OPERACIONAL ANDROID	29
2.1 Recursos	31
2.2 Interface de Programação e Aplicação (API)	32
2.3 Arquitetura do Aplicativo	33
2.4 Componentes da Aplicação	34
2.4.1 Atividades - <i>Activities</i>	36
2.4.1.1 Ciclo de vida da <i>Activity</i>	36
2.4.1.2 Intent e <i>Intent-Filter</i>	39
2.4.1.2.1 Intenção - <i>Intent</i>	39
2.4.1.2.2 Filtros de Intenção - <i>Intent Filter</i>	40
2.4.1.3 <i>View</i>	40
2.4.1.3.1 <i>Layout</i>	43
2.4.1.3.2 <i>Widgets</i>	43
2.4.2 Serviços - <i>Services</i>	45
2.4.3 Provedor de Conteúdo - <i>Content Provider</i>	47

2.4.4	Receptores de Broadcast ou Receptores de Intenções	47
2.5	<i>AndroidManifest.xml</i>	47
2.5.1	Estrutura do arquivo de Manifesto	48
3	O AMBIENTE DE DESENVOLVIMENTO	50
3.1	Eclipse	50
3.2	<i>Android Development Tools(ADT)</i>	51
3.3	<i>Software Development Kit - SDK</i>	51
3.3.1	<i>Android Virtual Device Manager (AVD Manager)</i>	52
3.3.2	Emulador	52
3.3.3	<i>Dalvik Debug Monitor Service - DDMS</i>	53
3.4	Instalação e Configuração	54
3.4.1	Requisitos do Sistema	54
3.4.2	Instalação do Eclipse	54
3.4.2.1	Instalação do <i>plugin ADT</i> no Eclipse	55
3.4.3	Configurando o SDK	57
3.4.3.1	Utilizando o Android SDK Manager	58
3.4.3.2	<i>Android Virtual Device Manager</i>	59
3.5	Criando uma aplicação no Android Eclipse	60
4	A CRIAÇÃO DE UM APLICATIVO PARA O ANDROID	63
4.1	Diretórios do Projeto	64
4.1.1	A pasta <i>src</i>	64
4.1.2	A pasta <i>gen (Generated Java Files)</i>	65
4.1.3	<i>A pasta assets</i>	66
4.1.4	A pasta <i>res (resources)</i>	66
4.1.4.1	Pasta <i>Layout</i>	66
4.1.4.2	Pasta <i>Drawable</i>	67
4.1.4.3	Pasta <i>Values</i>	67
4.2	Desenvolvimento	69
4.2.1	<i>Graphical User Interface (GUI)</i>	69
4.2.1.1	Declaração do <i>layout</i> da aplicação	70
4.2.1.1.1	Identificação de recurso	70

4.2.1.1.2	Dimensionalidade de elementos	71
4.2.1.1.3	Elementos do <i>layout</i>	72
4.2.2	Atividades (<i>Activity</i>)	75
4.2.2.1	Implementação do Ciclo de Vida da Atividade	76
4.2.2.2	Utilização de recursos	77
4.2.2.3	Interação entre atividades	78
4.2.2.4	Utilização de elementos do <i>layout</i>	79
4.2.3	<i>Android Manifest</i>	81
4.2.3.1	Elementos e atributos	82
4.2.3.2	Declarando os nomes das classes	83
4.2.3.3	Múltiplos valores	84
4.2.3.4	Permissões	85
4.2.4	Armazenamento de Dados	86
4.2.4.1	Classe <i>DbAdapter</i>	88
4.2.4.2	Classe <i>BasicoDAO</i>	90
4.2.4.3	Classe <i>ProdutoDAO</i>	91
4.3	Depuração e testes automatizados	93
4.3.1	Depuração (<i>Debugging</i>)	93
4.3.1.1	O ambiente de depuração	94
4.3.1.2	Depurando com o Eclipse e o plugin ADT	95
4.3.2	Testes	96
4.3.2.1	Criando o Projeto de Teste	98
4.3.2.2	Classe de Teste	98
4.3.2.2.1	Adicionando a classe Caso de Teste	99
4.3.2.2.2	Adicionando o construtor da classe de teste	100
4.3.2.2.3	Adicionando o método <i>setUp()</i>	100
4.3.2.2.4	Adicionando os testes de pré-condições	101
4.3.2.2.5	Adicionando um teste de interface de usuário	101
4.3.2.3	Executando os testes e visualizando os resultados	103
4.4	Publicação	105
4.4.1	Procedimentos para publicação no Google Play	105

5.1	A Aplicação	107
5.1.1	A Interface Inicial	107
5.1.2	Gerência de Categorias para Produtos	108
5.1.3	Gerência de Produtos	109
5.1.4	Gerência de Obras	110
5.1.5	Gerência das Listas de Compras da Obra	111
5.2	O processo de desenvolvimento	112
5.2.1	A base de dados	112
5.2.2	A interface gráfica	113
5.2.3	<i>Activity</i>	115
5.3	Testes automatizados	116
5.4	Publicação	117
	CONCLUSÃO	118
	REFERÊNCIAS BIBLIOGRÁFICAS	119

INTRODUÇÃO

Os avanços de hardware para celulares e plataformas de desenvolvimento aumentaram a demanda por dispositivos móveis. Seguindo as necessidades do mercado, a *Open Handset Alliance* (OHA) lançou em novembro de 2007 uma plataforma open source para dispositivos móveis, denominada Android, e disponibilizou a primeira versão do seu Kit de Desenvolvimento de Software (SDK). A OHA define o Android como uma plataforma móvel, única, aberta, moderna e flexível para o desenvolvimento de aplicações.

No período entre o lançamento da versão 1.5 (*Cupcake*) e da recente 4.1.2 (*Jelly Bean*) o número de *smartphones* vendidos com o Android cresceu, colocando-o em primeiro lugar na lista dos principais sistemas operacionais para dispositivos móveis, segundo pesquisa do *International Data Corporation* (IDC) divulgada em 24 de maio de 2012. Este crescimento gera um aumento na demanda por aplicativos. Porém, a quantidade de material gratuito na literatura em língua portuguesa ainda é insuficiente para atender as necessidades dos desenvolvedores.

Baseado nisto, este trabalho apresenta o Sistema Operacional Android, disponibilizando uma base com o conhecimento necessário para o desenvolvimento de aplicativos nesta plataforma. Sua estrutura, ambiente de desenvolvimento e recursos de programação são descritos. Por fim, é demonstrada a criação de um protótipo de aplicativo, que pode ser instalado e testado em um dispositivo móvel, como um *smartphone* ou um *tablet*, ou no próprio emulador. O trabalho foi estruturado de forma a permitir que desenvolvedores com vários níveis de experiência possam compreender e desenvolver aplicativos para o Android.

Inicialmente, no Capítulo 1 é apresentado um breve histórico sobre o Android. No Capítulo 2 há uma descrição do sistema operacional abordando sua estrutura e principais funcionalidades. Já o Capítulo 3, aborda como executar as instalações dos softwares e aplicativos necessários para o desenvolvimento, bem como a sua configuração. A exemplificação da estrutura e dos componentes que serão utilizados para criar o protótipo do aplicativo está contida no Capítulo 4. O Capítulo 5 traz o estudo de caso deste trabalho e descreve os desafios de se criar um aplicativo para o Android. Por fim, a conclusão traz as considerações sobre o desenvolvimento, além da proposta de melhoria do protótipo apresentado.

1 INTRODUÇÃO AO ANDROID

Afim de contextualizar o Android, este Capítulo irá abordar um breve histórico sobre a plataforma. Será feito um levantamento das principais versões existentes e suas funcionalidades e, por fim, uma análise de sua situação no mercado mundial e no Brasil.

1.1 Histórico

No ano de 2003 foi criada uma empresa embrionária que possuia um projeto promissor denominado Android Inc. Esta foi fundada por Andy Rubin, Rich Miner, Nick Sears e Chris White, e desenvolvia secretamente softwares para dispositivos móveis, (BLOOMBERGBUSINESSWEEK, 2005).

Em 2005 a Google comprou a Android Inc e manteve como desenvolvedores Andy Rubin, hoje vice-presidente senior do setor de mobilidade do Google, e Rick Miner, hoje parceiro no Google Ventures. Seu objetivo era desenvolver uma plataforma de telefone móvel baseada no Linux, tendo como meta ser uma plataforma flexível, aberta e de fácil migração para os fabricantes. Vale lembrar que em 2005 os aparelhos de celulares disponíveis eram modestos e com telas em preto e branco.

No ano de 2006 o Google demonstrou interesse pelo mercado de comunicações móveis, que deu início a uma série de especulações sobre este estar planejando o lançamento de um telefone com sua marca. Notícias veiculadas pela BBC e The Wall Street Journal (GIVOCE, 2012) reforçaram as especulações, principalmente depois do Google oferecer proposta para acesso a banda larga sem fios para a cidade de São Francisco, EUA. Atualmente, o Google oferece acesso Wi-Fi gratuito em Mountain View, Califórnia, sua cidade natal.

Em 5 de novembro de 2007 foi anunciada a criação da Open Handset Alliance (OHA) (ALLIANCE, 2011), formada pelo Google, fabricantes de aparelhos celulares e empresas do ramo de telefonia móvel, semicondutores, software e comercialização. No mesmo dia também foi anunciado o Android. A primeira versão de seu SDK *Software Development Kit* (Kit de desenvolvimento de Aplicativos) foi apresentada a desenvolvedores uma semana depois, ocasião em que o Android ainda não era livre.

Em 21 de outubro de 2008 o Android foi publicado como AOSP (*Android Open Source Project*). O “objetivo do projeto Android open source é criar um produto bem sucedido do mundo real que melhora a experiência móvel para os usuários finais“. Além disso, a AOSP mantém a compatibilidade das aplicações dos desenvolvedores particulares que usam o Android SDK e NDK, publicado sob a licença Apache (ANDROIDOPEN-SOURCE, 2012), (TSUHARESU, 2010). No dia seguinte foi realizado o lançamento do HTC Dream G1 o primeiro aparelho celular com o Android. A Figura 1.1 mostra as novas funcionalidades disponíveis no aparelho.



Figura 1.1: HTC Dream G1 - primeiro aparelho com o Android

Para a Google, a solução para o aumento da utilização de seus recursos nos smartphones via internet seria a criação de uma plataforma livre que pudesse ser adotada por diversos fabricantes, já que até então as plataformas disponíveis eram proprietárias, como o Symbian, o Windows Mobile e o PalmOS. Com o lançamento do Android este objetivo foi alcançado e, dada sua popularidade, este abre novas possibilidades de exploração de seus recursos.

A plataforma Android mostrou possuir os requisitos capazes de conquistar o mercado, porque era capaz de agradar quatro públicos que aparentemente possuíam interesses diferentes: os fabricantes de celulares, que poderiam utilizar a plataforma sem custos; os desenvolvedores, porque o SDK aberto permitiria desenvolver aplicativos de forma mais simples e barata; os fabricantes de chips, porque aceleraria e impulsionaria o desenvolvimento de processadores cada vez mais rápidos; e os consumidores, que poderiam adquirir aparelhos com mais recursos a preços mais baixos (MORIMOTO, 2008).

Para impulsionar ainda mais o desenvolvimento de aplicativos para o Android, no dia seguinte ao lançamento oficial, o Google divulgou que estava oferecendo US\$ 10 milhões (GLOBO.COM, 2007) para serem distribuídos, de acordo com os critérios definidos por este, para aqueles que desenvolvessem os melhores aplicativos para a plataforma Android, dentro das categorias pré-determinadas (DEVELOPERS, 2007).

1.2 A Evolução

Desde o lançamento da versão 1.0, disponível no HTC Dream G1, até os dias atuais foram lançadas 9 versões, sendo que uma delas exclusiva para tablets. As informações sobre as inovações disponibilizadas a cada nova versão foram retiradas do site xcubelabs (LABS, 2011).

1.2.1 Versão 1.x

No ano de 2008 foi realizado o lançamento da versão 1.0 que rodava no HTC Dream G1, como apresentado na Figura 1.1, que trazia: integração com serviços do Google, navegador para web, aplicativo do AndroidMarket para acesso aos aplicativos disponíveis para uso gratuito ou compra, sistema multitarefa, Wi-Fi e Bluetooth.

Em 2009 foram lançadas duas versões, a primeira em abril e a segunda em setembro. Em fevereiro houve uma atualização do Android apenas para os aparelhos T-Mobile G1.

- A versão 1.5

Em abril foi lançada a versão 1.5 denominada *Cupcake*, baseada no Kernel Linux 2.6.27, que trouxe evoluções como: inicialização da câmera mais rápida, o teclado virtual, os aplicativos de terceiros, integração com o Youtube e capacidade de realizar os comandos copiar e colar. A Figura 1.2 a seguir apresenta um aparelho com esta versão.



Figura 1.2: Aparelho com a versão 1.5 - Cupcake (INTERESSANTE, 2009)

- A versão 1.6

Lançada em setembro, a versão 1.6 *Donut*, sobre o Linux 2.6.29, que trouxe mais inovações como: uma caixa de pesquisa rápida, compatibilidade com câmeras, gravador de voz, galeria de fotos, indicador de nível de bateria, busca por voz, suporte para CDMA. Estas mudanças podem ser observadas na Figura 1.3.



Figura 1.3: Aparelho com a versão 1.6 - Donut (INTERESSANTE, 2009)

1.2.2 Versão 2.x

Em outubro de 2009 foi lançada uma nova versão que recebeu atualizações durante o ano de 2010, nos meses de maio e dezembro.

- A versão 2.0

Acrescentando mais funcionalidades em outubro foi lançada a versão 2.0 *Eclair*, que utiliza a mesma versão do Linux, e que disponibilizou o uso de múltiplas contas de e-mail, suporte a Bluetooth 2.1 e a HTML 5; trouxe também novas ferramentas de calendário, navegação pelo Google Maps e papéis de parede animados. A Figura 1.4 apresenta um aparelho com esta nova versão.



Figura 1.4: Aparelho com a versão 2.0 - Eclair (INTERESSANTE, 2009)

No mês de dezembro foi lançado o Android SDK 2.0.1.

- A versão 2.2

No mês de janeiro de 2010 foi lançada a versão 2.1 do SDK.

Em maio, a versão 2.2 *Froyo*, sobre a versão 2.6.32 do Linux, e que trouxe novidades como: o suporte para o Adobe Flash, discagem por voz, conexão USB, a possibilidade de instalar aplicativos, o aumento da velocidade na realização de tarefas, a opção de salvar aplicativos em cartões de memória e a nova funcionalidade de roteamento de internet, teclado multilinguagem. A Figura 1.5 apresenta algumas das interfaces desta nova versão.



Figura 1.5: Aparelho com a versão 2.2 - Froyo (INTERESSANTE, 2009)

Após o lançamento desta versão, as pesquisas da época, apresentaram um crescimento considerável de aproximadamente 454%, ou seja, o Android passou de 3,9% para 17,7% do mercado.

- A versão 2.3

No mês de dezembro foi lançada a versão 2.3 *Gingerbread* que trouxe mais estabilidade e novidades como por exemplo: suporte para WXGA e telas de maior resolução, além de oferecer suporte para chamadas pela internet (câmera frontal), um teclado reformulado e a otimização das opções de edição (copiar, recortar e colar). Estas inovações foram pensadas levando em conta os tablets que estavam sendo lançados. A Figura 1.6 mostra a nova interface.



Figura 1.6: Aparelho com a versão 2.3 - Gingerbread (INTERESSANTE, 2009)

Em fevereiro de 2011 foi lançada a atualização da versão 2.3.

1.2.3 Versão 3.x

- A versão 3.0

Em maio de 2011 foi lançada a versão 3.0 *Honeycomb*, baseada no Linux 2.6.36, exclusiva para tablets e o seu SDK. A Figura 1.7 apresenta funcionalidades desenvolvidas para os tablets, que teve quase todo sistema reprojetado para tentar sanar os problemas de compatibilidade. Desta forma, do teclado ao aplicativo do e-mail, foi redesenhado para a tela maior e o processamento foi melhorado, deixando ainda mais rápido. Os maiores destaques desta versão foram: a possibilidade de customização da tela principal e a opção de compartilhar por bluetooth (INTERESSANTE, 2009).



Figura 1.7: Aparelho com a versão 3.0 - Honeycomb (INTERESSANTE, 2009)

1.2.4 Versão 4.x

Lançada a primeira em 2011 e a segunda em 2012 voltadas para smartphones.

- A versão 4.0

Em maio no evento Google I/O foi realizado o anúncio da nova versão do Android 4.0, *Ice Cream Sandwich*, e seu lançamento oficial ocorreu em outubro do mesmo ano.

A Figura 1.8 apresenta as inovações que levaram muitos a considerá-la a melhor plataforma móvel do mundo. O sistema operacional passou a oferecer maior facilidade de compartilhamento de arquivos, calendário unificado, opções de alinhamento de câmera, desbloqueio por reconhecimento facial e um centro de análise de dados (INTERESSANTE, 2009). Além do redimensionamento de widgets, processos multitarefas, telas personalizáveis e novas formas de comunicação.



Figura 1.8: Aparelho com a versão 4.0 - Ice Cream Sandwich (ROQUEIROJP, 2012)

No mês de julho foi disponibilizada a versão 3.2 do SDK.

- A versão 4.1

No dia 21 de junho de 2012 foi anunciada a nova versão do Android denominada Jelly Bean, que é uma evolução do Ice Cream Sandwich, e dentre as melhorias traz ferramentas de acessibilidade destinadas a deficientes visuais que facilitam o uso por estes. Além disso, foram aprimorados também, a interface gráfica melhorando a qualidade para os jogos, melhorias no áudio. Isto é possível porque a velocidade de resposta foi melhorada, tornando-o estremamente rápido. Há um melhor gerenciamento de bateria, a interface *homescreen* está mais limpa o que permite uma melhor organização dos widgets pelo usuário. Para a versão em inglês um novo editor que permite escrever textos offline através da voz, ou seja, um ditado; novos recursos agregados a câmera, novas notificações, novo motor de busca, um novo Google now, atualizações de aplicativos em pequenas partes, apenas o que foi mudado. A Figura 1.9 apresenta um exemplo da nova versão.

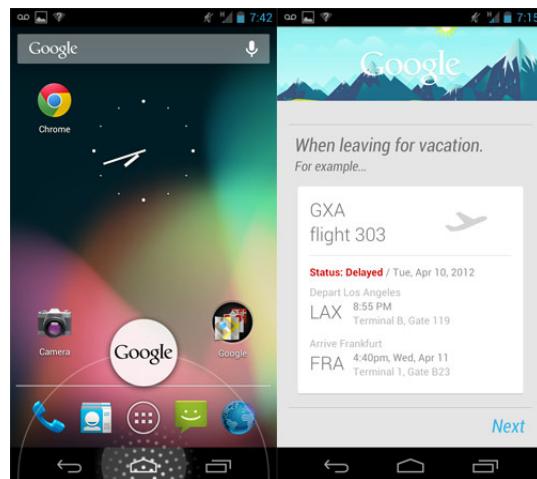


Figura 1.9: Aparelho com a versão 4.1 - Jelly Bean (MURPH, 2012)

1.2.5 Um comparativo entre as versões

Segundo Android 2012m, apesar da evolução do Android a versão que ainda domina os aparelhos disponibilizados no mercado é a Gingerbread (2.3). Isto ocorre porque os modelos mais novos com as versões mais recentes não estão disponíveis ou o preço é alto demais.

A Figura 1.10 apresenta um gráfico onde é possível observar qual o grau de utilização das versões do Android. Estes dados foram publicados pela Google, baseados nas informações provenientes dos acessos feitos ao Google Play durante o período de 18 de outubro de 2012 até 1º de novembro do mesmo ano.

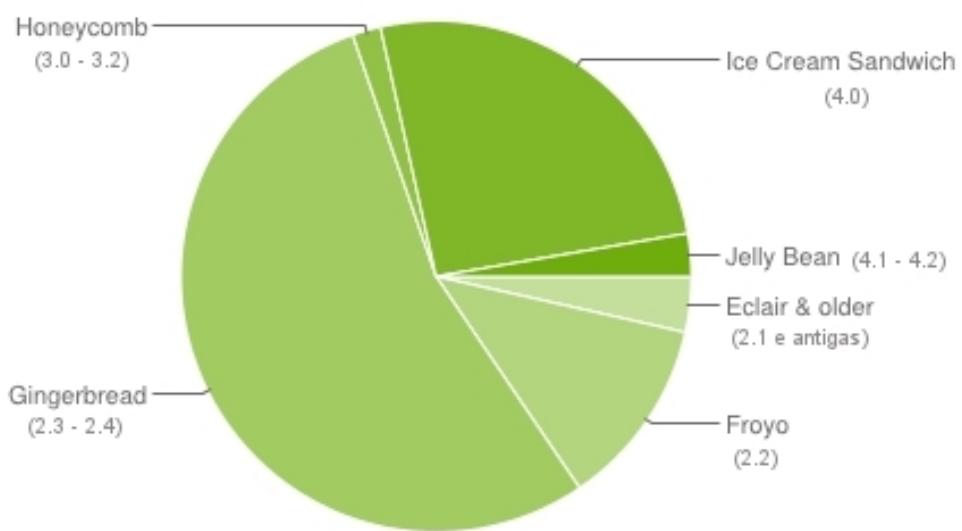


Figura 1.10: Versões do Android por utilização

1.3 O Momento Atual

O crescimento desta plataforma impressiona e os concorrentes que detinham uma boa fatia do mercado de *smartphones* tendem a desaparecer ou fundir-se para tentar sobreviver. (EXAME.COM, 2010)

1.3.1 O cenário mundial

O crescimento do Android no mercado mundial apresenta um novo cenário para os sistemas operacionais para smartphones. Em pesquisas recentes, publicadas pela Nielsen Company, o mercado norte americano é dominado pelas plataformas Android (Google) e iOS (Apple) que detêm 90% deste, como explicitado no Google Discovery (FRAGA, 2012). A Figura 1.11 apresenta um comparativo entre os sistemas operacionais naquela data e nos últimos três meses.

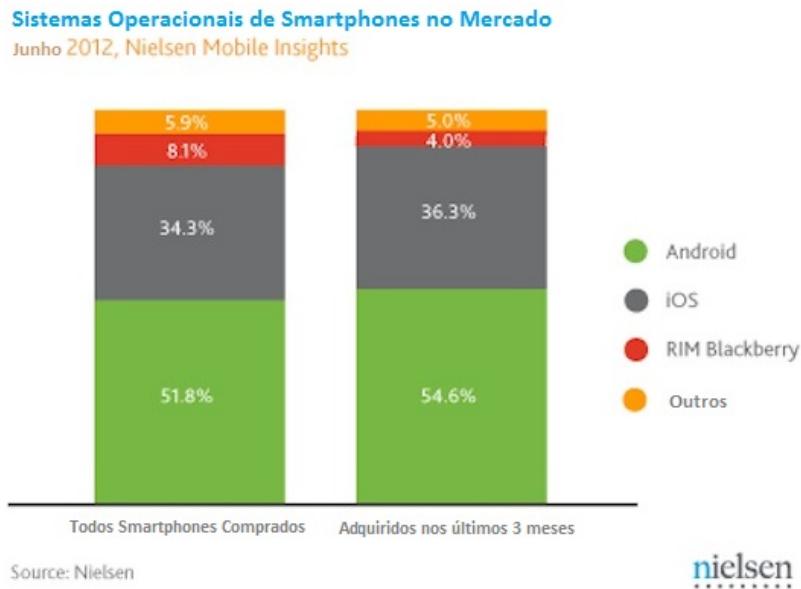


Figura 1.11: Sistemas operados nos smartphones (FRAGA, 2012)

A Figura 1.12 apresenta os principais fabricantes de aparelhos smartphones e os sistemas operacionais disponibilizados, e permite visualizar que alguns fabricantes disponibilizam mais de uma plataforma.

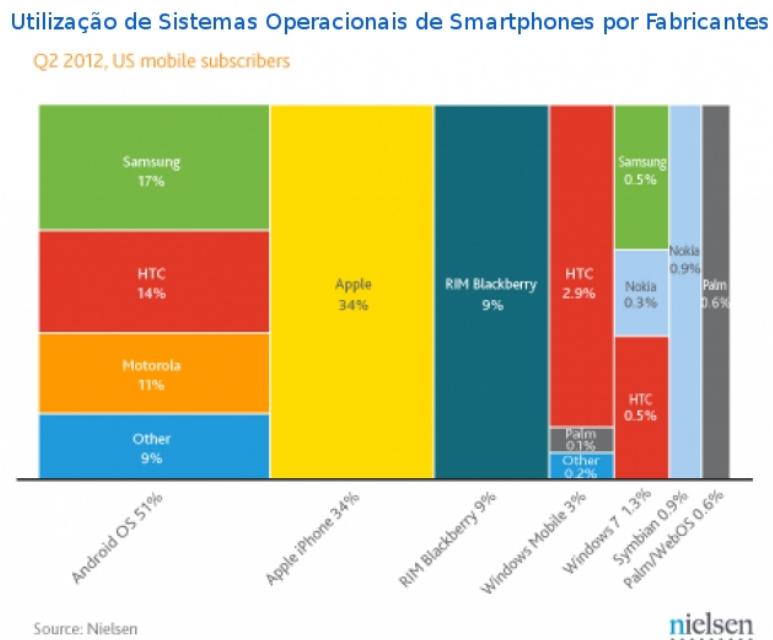


Figura 1.12: Sistemas operacionais por fabricantes (FRAGA, 2012)

1.3.2 O cenário brasileiro

Se nos Estados Unidos, Europa e China o Android apresenta crescimento e domínio de mercado, o mesmo ainda não se reflete no Brasil onde este ainda é exercido pelo

Symbian. Uma provável explicação pode ser os preços dos aparelhos que utilizam o Android ainda serem altos para a maioria da população. Mas com a crescente queda nos valores acredita-se que em breve o Brasil estará no mesmo nível que os outros países.

A Kantar Worldpanel Comtech (TELETIME, 2012) divulgou uma pesquisa na qual apresenta uma análise dos últimos doze meses (de maio de 2011 a maio 2012) em que a plataforma Symbian ainda lidera o mercado e mostra ainda o crescimento do Android. Contrariando a pesquisa da IDC (*International Data Corporation*) divulgada em março que aponta o Android com 50% do mercado nacional, sem contudo informar sobre as outras plataformas.

A Figura 1.13 apresenta os dados da pesquisa divulgados pelo Kantar, e aqui reunidas em forma de gráfico gerada a partir destes dados.

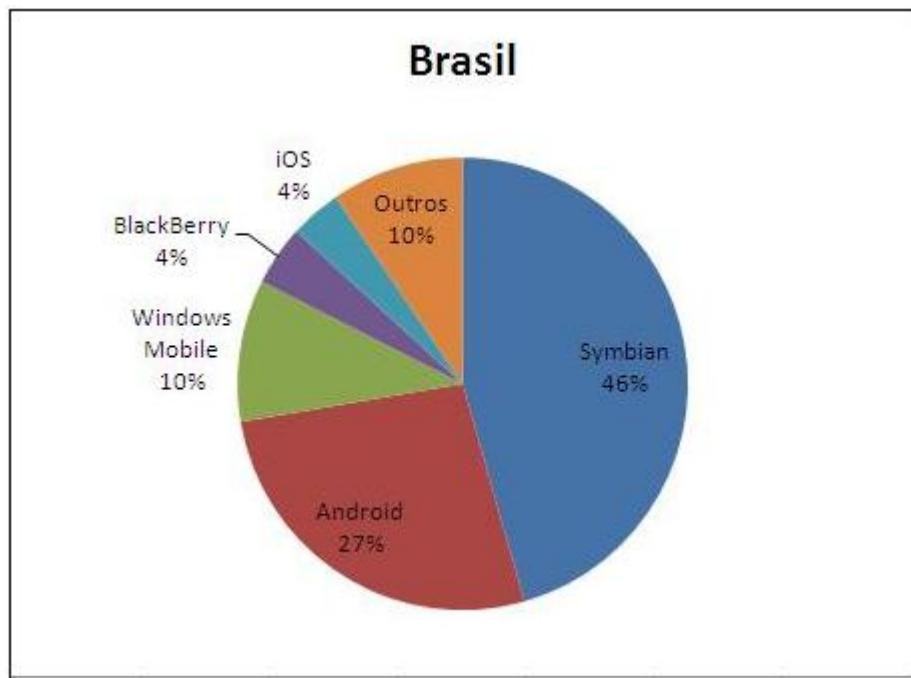


Figura 1.13: Utilização dos sistemas operacionais no Brasil (TELETIME, 2012)

Observa-se que apesar do Symbian dominar o mercado brasileiro, sua participação vem caindo, e em contrapartida, o Android vem crescendo, assim como o iOS que também apresentou aumento, mas sua participação comparada ao Android é 23% menor.

2 O SISTEMA OPERACIONAL ANDROID

Neste Capítulo será abordado o sistema operacional Android, baseado no kernel do Linux, responsável pelo gerenciamento de memória, processos, threads, segurança dos arquivos e pastas, redes e drivers.

A sua estrutura física e funcionalidades são descritas no site oficial Android Developers (ANDROID, 2012c), no livro Android para Desenvolvedores (PEREIRA; SILVA, 2009), dentre outros.

A Figura 2.1 mostra a estrutura da arquitetura do Android e seus principais componentes são detalhados abaixo.

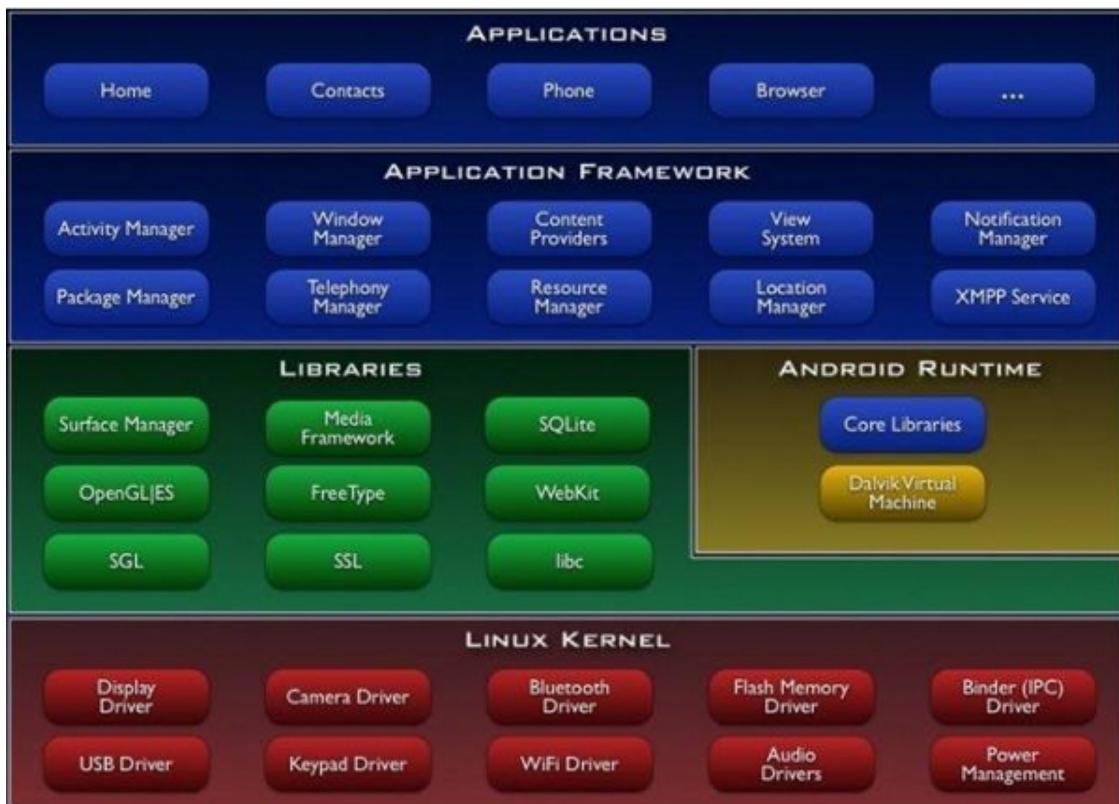


Figura 2.1: Sistema Operacional Android (ANDROID, 2012a)

- Aplicações (*Applications*)

São os aplicativos desenvolvidos em Java, alguns nativos e outros criados pela comunidade de desenvolvedores Android. São aplicações nativas como calendário, calculadora, agenda, cliente de e-mail e SMS, bússola, entre outros.

- Framework de Aplicação (*Application Framework*)

Nesta camada se encontram as API's e os elementos necessários para a manipulação de aplicativos.

Tem como principal objetivo facilitar a reutilização de componentes. Todas as aplicações presentes no sistema podem ter seus recursos utilizados por outras aplicações, garantindo uma grande integração das API's Android.

Dentre os elementos desta camada podemos destacar:

- *Activity Manager* - é quem gerencia o ciclo de vida de todas as atividades;
- *Package Manager* - é utilizado pelo *Activity Manager* para acesso às informações dos APKs (Pacotes de Arquivos do Android). Ele é quem faz a comunicação com o restante do sistema permitindo o conhecimento de quais pacotes são utilizados e quais são suas capacidades;
- *Window Manager* - gerenciador de apresentação de janelas;
- *Content Providers* - possibilita a troca de informações entre os aplicativos;
- *View System* - responsável por disponibilizar a parte gráfica do aplicativo.

- Bibliotecas (*Libraries*)

Possui um conjunto de bibliotecas C/C++ utilizadas pelo sistema, mais as bibliotecas para as áreas de: multimídia, visualização de camadas 2D e 3D, funções para navegadores web, funções para gráficos, funções de aceleração de hardware, renderização 3D, fontes bitmap e vetorizadas e funções de acesso ao banco SQLite.

- Ambiente de Execução (*Android Runtime*)

- *Dalvik Virtual Machine*

Para cada aplicação executada no Android é instanciada uma pequena camada da máquina virtual Dalvik. Toda e qualquer aplicação roda dentro de seu próprio processo, isto é, encapsulada no contexto da sua máquina virtual. A Dalvik é uma máquina virtual com melhor desempenho para dispositivos móveis, permitindo maior integração tecnológica de hardware. Seu funcionamento foi projetado para sistemas com baixa frequência de CPU, pouca memória RAM disponível e sistema operacional em espaço de swap, além de ter sido otimizada para consumo mínimo de memória, bateria e CPU. É capaz de executar paralelamente várias instâncias de sua máquina virtual com eficiência.

- *Core Libraries*

Conjunto de bibliotecas responsável pelo fornecimento da maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem Java, como estrutura de dados, acesso a arquivos, a redes e gráficos.

- *Linux Kernel*

O *kernel* do Linux é utilizado para o gerenciamento dos serviços centrais do sistema como segurança, gestão de memória, gestão de processos, pilha de protocolos de rede e modelo de *drivers*. O *kernel* exerce ainda, a atividade de separação entre o hardware e o restante da arquitetura.

Cada aplicativo dá início a um novo processo no sistema operacional. Isso ocorre porque uma aplicação Android roda em um processo separado, com sua própria máquina virtual, número de processos e usuário. Desta forma, caso uma aplicação apresente problemas, ela pode ser isolada e removida da memória sem comprometer o restante do sistema.

Um aplicativo pode ser exibido para o usuário ou ficar rodando em segundo plano por tempo indeterminado. Como diversos processos e aplicativos podem ser executados simultaneamente, o controle da memória é realizado pelo kernel do sistema. Desta forma, caso seja necessário o próprio sistema operacional pode optar por encerrar algum processo para liberação de memória e recursos, podendo posteriormente decidir pelo seu reinicio quando a situação estiver sob controle.

2.1 Recursos

A arquitetura do sistema operacional Android, apresentada na Figura 2.1, é estruturada em camadas desde o nível mais baixo, onde está localizado o *kernel*, até a camada dos aplicativos, no nível mais alto. Os componentes presentes em cada uma destas camadas tem acesso a vários recursos presentes no sistema. Os principais recursos desta plataforma, segundo (PEREIRA; SILVA, 2009), são:

- Framework de Aplicação: permite a incorporação, a reutilização e a substituição de recursos de componentes e manutenção;
- Máquina Virtual Dalvik: criada e otimizada para dispositivos móveis e suas limitações;
- Navegador web integrado: baseado no open source webkit;

- Gráficos otimizados: gráficos e tratamento de imagens por meio de uma biblioteca 2D e gráfico 3D baseados na especificação Open GLES 1.0 e pode ter aceleração por hardware como opcional;
 - SQLite: gerenciador de banco de dados para armazenamento de dados estruturados;
 - Suporte multimídia: compatibilidade com os principais formatos do mercado, som (MP3, AAC, AMR), vídeo (MPEG4, H.264), imagens (JPG, PNG e GIF) nativamente;
 - Telefonia GSM: permite integração com as tecnologias GSM, mas apresenta limitações dependentes do hardware;
 - Protocolos de comunicação wireless: suporte a tecnologia Bluetooth, EDGE, 3G e WiFi, porém possui as mesmas limitações da telefonia;
 - Câmera, GPS, bússola e acelerômetro: fácil integração com hardwares, mas depende das limitações do mesmo;
 - Poderoso ambiente de desenvolvimento: inclui um emulador de dispositivo, ferramenta para depuração, analisador de memória e performance e um plugin para a IDE Eclipse (ADT).
- (PEREIRA; SILVA, 2009, p.9,10)

O Android foi construído de forma básica visando permitir aos desenvolvedores criarem aplicativos capazes de explorar ao máximo todos os recursos disponíveis. Ele não faz diferença entre os aplicativos nativos e os de terceiros, todos podem acessar os recursos disponíveis.

Para desenvolver novos aplicativos, são disponibilizadas ferramentas e API's pelo Android SDK (Kit de Desenvolvimento de Software). A codificação utiliza a linguagem Java como explicitado por Android (2012e) e por Pereira e Silva (2009).

2.2 Interface de Programação e Aplicação (API)

Permite ao desenvolvedor criar aplicativos utilizando os recursos disponíveis apenas importando-os. Isto é possível porque uma API é um conjunto de rotinas e padrões pré-estabelecidos no sistema. As principais API's do Android são:

- android.util: contém várias classes utilitárias (classes de containers e utilitários XML);

- android.os: contém serviços referentes ao sistema operacional, passagem de parâmetros e comunicação entre processos;
 - android.database: contém as API's para a comunicação com o banco de dados SQLite;
 - android.content: API's de acesso a dados no dispositivo, como as aplicações instaladas e seus recursos;
 - android.view: pacote que contém as principais funções e componentes de interface gráfica;
 - android.widget: contém widgets prontos (botões, listas, grades, etc.) para serem utilizados nas aplicações;
 - android.app: API's de alto nível referentes ao modelo da aplicação;
 - android.provider: API que contém os padrões de provedores de conteúdos(*Content Providers*);
 - android.telephony: API para interagir com funcionalidades de telefonia e telecomunicações;
 - android.webkit: inclui API's para conteúdo web, bem como um navegador embutido para utilização geral;
 - android.maps: possui as bibliotecas necessárias para se trabalhar com mapas.
- (PEREIRA; SILVA, 2009, p.11)

2.3 Arquitetura do Aplicativo

A máquina virtual Dalvik, desenvolvida especialmente para o sistema operacional Android, é baseada em registradores e otimizada para atender as necessidades dos equipamentos que dispõem de pouca memória (DALVIK, 2008). A IBM em seu tutorial sobre o desenvolvimento do Android (ABLESON, 2009) apresenta não apenas a arquitetura, mas também os componentes da aplicação, assim como (BIRK, 2010) em sua apresentação sobre a plataforma explica como uma aplicação é executada na Dalvik.

Cada aplicativo Android é executado em uma instância da máquina virtual Dalvik, que, por sua vez, reside em um processo que é gerenciado pelo *kernel Linux*. A Figura 2.2 remodelada por (BIRK, 2010) demonstra esse relacionamento.

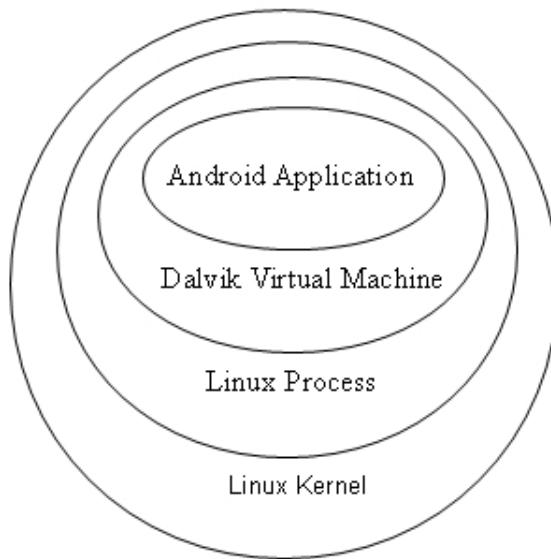


Figura 2.2: Relacionamento da aplicação com o ambiente Android

Por padrão cada aplicativo é executado em seu próprio processo Linux. Cada processo tem sua própria máquina virtual (VM), por isso o código do aplicativo é executado de forma isolada. A cada aplicativo é atribuído um único ID de usuário Linux.

É possível duas aplicações compartilharem o mesmo ID de usuário, neste caso todos os arquivos seriam visíveis aos dois. Para conservar os recursos do sistema, aplicações com o mesmo ID podem também ser organizadas em um mesmo processo Linux e compartilhar uma mesma VM.

2.4 Componentes da Aplicação

O conhecimento e domínio dos componentes que fazem uma aplicação Android é fundamental para a máxima otimização. Podemos dizer que a *activity* é a sua “alma”, mas os outros componentes também são importantes. A sapiência no seu uso é capaz de tornar o aplicativo ágil, como descrito em Android Developers (2012c), além dos livros do Pereira e Silva (2009), do Lecheta (2009) e de Rogers et al (2009).

Um aplicativo não faz uso obrigatório de todos os componentes (atividade, serviço, broadcast receiver e provedor de conteúdo), mas observando a forma como o Android foi desenvolvido e como as aplicações deverão ser executadas em dispositivos que possuem espaço de memória limitado, o emprego de todos os mecanismos disponíveis que visem à funcionalidade, reuso e associação a outros aplicativos devem ser observados no momento do desenvolvimento.

As ações são executadas pelo sistema operacional através de intenções. A classe android.content.Intent representa as ações que a aplicação deseja executar através de mensagens assíncronas que são enviadas via broadcast e que permitem a comunicação entre diferentes aplicações.

Os dados de uma aplicação estão ligados ao aplicativo que os criou. Para que outras aplicações tenham acesso a esses dados, estes devem estar disponíveis de alguma forma. O Android possibilita o compartilhamento através da API de Provedor de Conteúdo. Essa API permite à aplicação consultar o sistema operacional para obtenção de dados de seu interesse utilizando um mecanismo Identificador de Recurso Uniforme (URI¹) semelhante às solicitações feitas pelo navegador na internet. Isto ocorre porque um URI possui um ID que identifica de forma única determinado recurso e um Intent, objeto que detém a informação desejada por um aplicativo em forma de mensagem. Eles trabalham juntos, mas não estão ligados, o que torna o ambiente de desenvolvimento flexível e permite que aplicações possam ser adicionadas, substituídas e excluídas de forma fácil.

Todos os componentes de uma aplicação precisam estar declarados em um arquivo denominado AndroidManifest.xml que possui todos os nomes das classes e tipos de eventos que o aplicativo pode processar, além das permissões necessárias para a execução. Este pode ser comparado ao arquivo web.xml das aplicações Java web.

Um aplicativo não incorpora o código de outro ou o link de acesso para ele. Em vez disso, quando necessitar irá inicializar aquela parte do aplicativo. Para que isso seja possível um aplicativo Java deve instanciar objetos da parte que necessita. Isso é possível porque um aplicativo Android possui um grupo de componentes essenciais que o sistema pode instanciar e executar sempre que necessário. Existem quatro tipos de componentes:

- *Activity* – geralmente representa a interface com o usuário. É responsável por tratar os eventos desta interface;
- *Service* – são códigos que rodam em background, sem interface com o usuário. É responsável por prover um determinado tipo de serviço até que seja finalizado;
- *Content Provider* – o trabalho de um provedor de conteúdo é gerenciar o acesso aos dados entre aplicações;
- *Broadcast Receiver* ou *Intent Receiver* – um aplicativo Android pode ser ativado por um evento externo. O *Broadcast Receiver* é responsável por tratar a reação deste evento.

¹(Uniform Resource Identifier) é uma cadeia de caracteres que identifica um recurso de forma única (celeiroandroid.blogspot celeiroandroid.blogspot, 2011).

2.4.1 Atividades - *Activities*

No Android uma aplicação é uma *activity* ou um conjunto de *activities*, cada uma representando uma interface da aplicação. Mesmo que uma aplicação possua muitas *activities*, cada *activity* é considerada em separado. O seu bom entendimento é fundamental por aqueles que desejam criar aplicativos para o Android.

Ao se criar a classe *Activity* para as telas de uma aplicação esta será a responsável pelo controle dos eventos, pelo estado desta tela, além de definir qual *View* será responsável por desenhar a interface gráfica. Isto ocorre porque uma *Activity* é uma classe que gerencia as interfaces do usuário (UI).

2.4.1.1 Ciclo de vida da *Activity*

O Android foi projetado para atender requisitos específicos para as aplicações móveis e suas limitações, desta forma seu ciclo de vida é bem definido. O sistema operacional é responsável por cuidar desse ciclo de vida. A mudança de estado de uma *activity* é tratada de forma padrão pelo sistema, mas o desenvolvedor pode designar como ela irá se comportar durante o processo.

Quando uma *activity* é iniciada esta é inserida no topo da pilha de atividades do sistema. Nesse processo a atividade que estava em execução é pausada e passa a ocupar a posição abaixo da nova. Isso significa que o topo da pilha reflete o primeiro plano do sistema, onde está a aplicação em execução, já as demais posições refletem o segundo plano, onde estão as aplicações pausadas ou paradas. A Figura 2.3 demonstra o ciclo de vida da atividade.

O ciclo de vida da atividade define os estados ou eventos pelos quais atravessam a Atividade, desde sua criação até seu término. Cada atividade possui um ciclo de vida específico. A classe *Activity* possui métodos para cada evento:

- *onCreate()*: é um método obrigatório chamado para dar início ao ciclo de vida da atividade;
- *onStart()*: é chamado quando a atividade está ficando visível ao usuário, ou seja, já possui uma view criada. Pode ser chamado após os métodos *onCreate()* ou *onRestart()*;
- *onResume()*: chamado após a execução de *onStart()*. Nesse momento a atividade encontra-se em execução e interagindo com o usuário;
- *onPause()*: é chamado para salvar o estado da aplicação que será recuperado posteriormente quando a atividade for reiniciada;

- *onStop()*: é chamado quando a atividade não estiver mais sendo utilizada, iniciou-se outra atividade ou foi encerrada;
- *onDestroy()*: pode ser chamado quando a atividade é finalizada ou quando o sistema precisa finalizar atividades para liberação de recursos;
- *onRestart()*: é chamado para reiniciar uma atividade. Precede a execução do método *onStart()*;

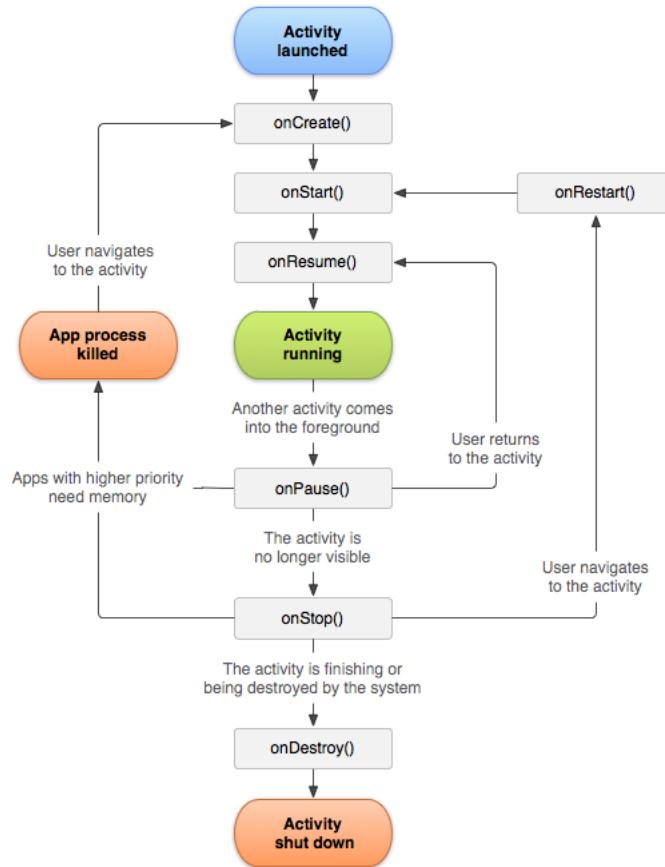


Figura 2.3: Ciclo de vida de uma Atividade (ANDROID, 2012c)

A Figura 2.4 apresenta os estados no qual uma atividade pode se encontrar desde sua criação.

1. Executando: a atividade está sendo utilizada pelo usuário;
2. Parada: uma atividade parou de ser executada. Mantém todas as informações do estado, podendo voltar a ser executada ou finalizada;
3. Interrompida: pode ser reiniciada ou finalizada;

4. Finalizada: pode ser removida caso tenha sido interrompida de forma temporária ou esteja parada. Para restaurar o estado anterior há a necessidade do desenvolvedor ter previsto anteriormente.

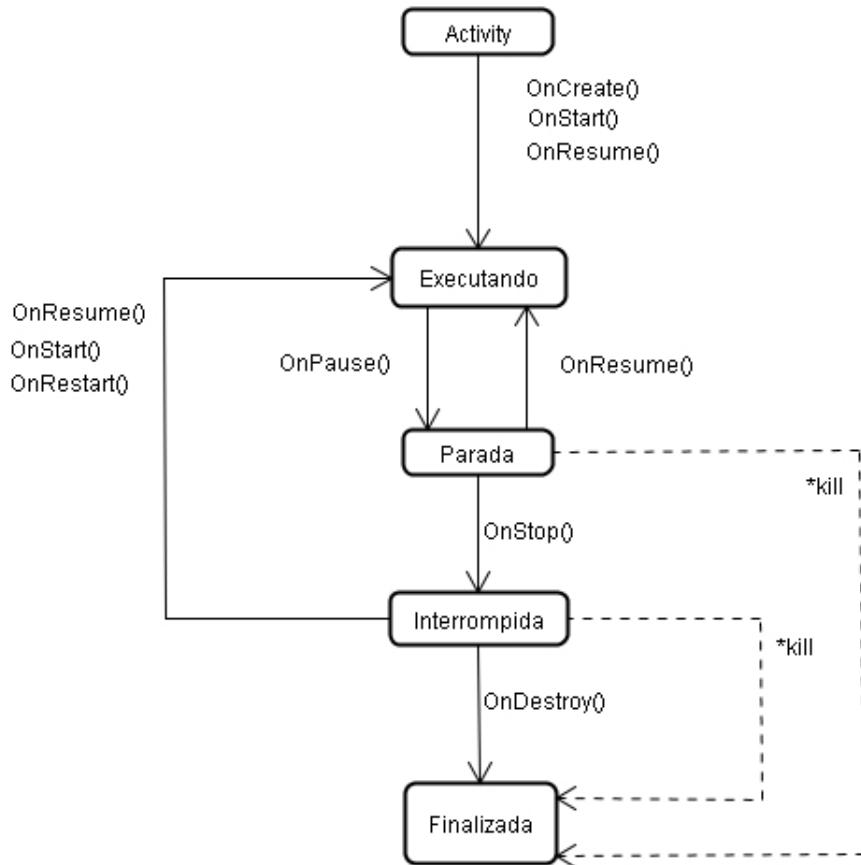


Figura 2.4: Estados da atividade (PEREIRA; SILVA, 2009, p.12)

No estado *Parada* ou *Interrompida* o sistema pode optar por finalizar a atividade caso os recursos estejam baixos, sem que sejam chamados os métodos `onDestroy` ou `onStop`.

Basicamente o ciclo de uma atividade pode ser classificado em três tipos. São eles:

- Ciclo completo: inicia-se no método `onCreate()`, percorre todo ciclo mudando seu estado e termina no `onDestroy()`, liberando os recursos e a memória utilizados;
- Ciclo de vida visível: inicia-se no método `onStart()` e termina no `onStop()`. Compreende o momento quando a atividade está visível ao usuário, mesmo que ele não esteja interagindo com ela, até quando o usuário não a visualiza;
- Primeiro ciclo da atividade: ocorre entre os métodos `onResume()` e `onPause()`. Período em que a atividade está visível e interagindo com o usuário.

2.4.1.2 Intent e *Intent-Filter*

2.4.1.2.1 Intenção - *Intent*

É a responsável pela ativação de componentes. Os provedores de conteúdo são ativados quando requisitados pelo *Content Resolver*. Já as atividades, serviços e *broadcast receiver* são por mensagens assíncronas denominadas *Intent*.

Uma *Intent* é um objeto de intenções que contém o conteúdo da mensagem. Para que as atividades e serviços sejam ativados há a necessidade da utilização de um URI.

Um objeto de intenções é um pacote de informações. Ele contém informações de interesse para o componente que recebe a intenção, além de informações de interesse para o sistema Android. Ele deve conter principalmente:

- **Nome do componente**

É quem irá receber a intenção, mas este não é obrigatório, dependerá do tipo de intenção, se explícita ou implícita, neste último caso o Android usará as outras informações contidas no objeto de intenções para achar o destino adequado.

- **Ação**

É uma string com o nome da ação a ser executada. A classe de intenções define um mínimo constante de ações. Ela determina em grande parte como o restante da intenção é estruturada, em especial os dados e campos extras.

- **Dados**

Ao combinar a intenção de um componente que é capaz de manipular dados é sempre importante saber o tipo de dado. Por exemplo, um componente capaz de exibir dados de imagem não deve ser chamado para executar um arquivo de áudio.

- **Categoria**

É uma sequência contendo informações adicionais sobre o tipo de componente que deve lidar com a intenção. Qualquer número de descrições de categoria pode ser colado em um objeto de intenções, como se faz para as ações, a classe de intenções define constantes de diversas categorias.

- **Extras**

Pares de chaves para informações adicionais entregues ao manipulador de intenção.

- **Flags**

São de vários tipos. Muitas instruem o sistema Android ao iniciar de uma atividade (por exemplo, a que tarefa a atividade pertence) e como tratá-la depois de

lançada (por exemplo, se ela pertence à lista de atividades recentes). Todos esses sinalizadores são definidos na classe intenções (ANDROID, 2012c).

As intenções são divididas em dois grupos:

1. Intenções explícitas – designa o componente de destino pelo seu nome, geralmente são utilizadas para aplicações de mensagens internas;
2. Intenções implícitas – não possui um nome explícito sendo geralmente utilizados para ativar os componentes em outros aplicativos.

2.4.1.2.2 Filtros de Intenção - *Intent Filter*

É um objeto que contém a descrição de quais *Intents* uma *activity* ou *broadcast receiver* são capazes de tratar. Após passar por este a *Intent* é enviada para a atividade correspondente. As *Intents* explícitas não utilizam este componente por que são entregues diretamente a *activity* que pode tratá-la.

Cada filtro descreve a capacidade do componente, um conjunto de intenções que o componente está apto a receber. Deve possuir as referências implícitas que podem ser utilizadas pelas atividades, serviços e *broadcast receiver*.

Um filtro possui os campos definidos na classe de intenções, a saber: ação, dados e categoria de um objeto de intenções. Uma intenção implícita é testada contra o filtro em todas as três áreas. Para ser entregue tem obrigatoriamente que passar nas três, se falhar em alguma delas o sistema Android não irá entregá-la ao componente, pelo menos não com base naquele filtro. No entanto, uma vez que um componente pode ter vários filtros de intenção, uma intenção que não passe em um filtro de um componente pode fazê-lo através de outro. Os testes são:

- Testes de ação;
- Testes de categoria;
- Testes de dados (ANDROID, 2012c).

2.4.1.3 *View*

O Android possui um sistema de multiprocessamento capaz de suportar múltiplas aplicações concorrentes, aceitando várias formas de entradas de dados, sendo altamente interativo e flexível. Tudo isso se reflete na interface do usuário, mas para criar uma interface agradável há necessidade da intervenção da *View*.

O ambiente Java disponibiliza as ferramentas de construção da GUI (AWT, Swing, SWT e J2ME). O Android, por sua vez, adiciona um *kit* GUI às ferramentas já existentes. Esse *kit* é um conjunto de interfaces gráficas que visam simplificar o *design* de aplicativos e o desenvolvimento, permitindo ao desenvolvedor focar na parte funcional do programa.

Uma *Activity* é responsável por controlar os eventos e o estado da tela que representa. Porém, esta não tem a capacidade de desenhar componentes visuais, como botões e campos de texto. Para isso, a *activity* é auxiliada pela classe android.view.View.

A Figura 2.5 mostra um diagrama resumido da hierarquia da classe *View* e todas as subclasses onde se encontram os componentes necessários para a criação das telas.

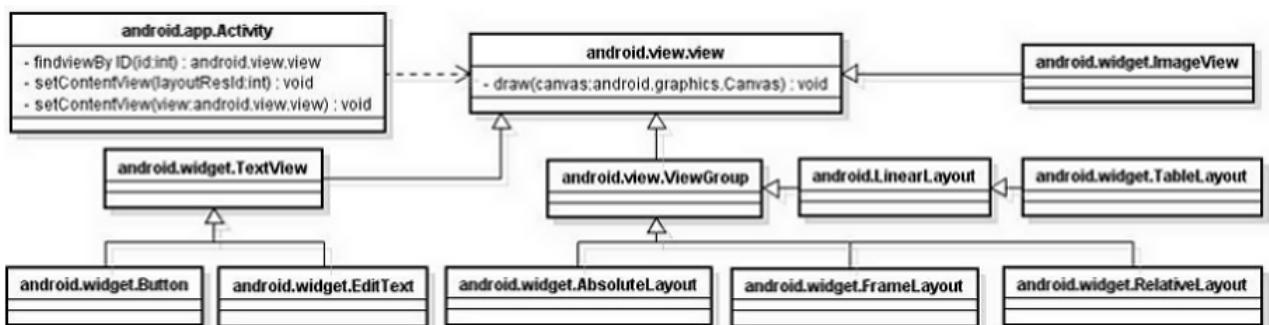


Figura 2.5: Hierarquia da classe *View*, (LECHETA, 2009, p.45)

É possível verificar a existência de dois métodos *setContentView(view)* na classe *Activity* do diagrama da Figura 2.5. Esse método é responsável por solicitar a classe android.view.View que implemente um determinado layout. É possível informar a este método número de identificação de um arquivo de layout ou a instância de um objeto do tipo android.view.View contendo o *layout* criado diretamente no código-fonte da classe *Activity* em execução. Possibilitando assim duas opções de construção da interface.

A classe android.view.View é a classe pai de todos os componentes visuais do Android e suas filhas são:

- Android.widget.TextView – Contém as caixas de texto, botões comuns e de seleção;
- Android.widget.ImageView – Contém ícones e fotos;
- Android.view.ViewGroup – Gerencia as classes de *layout*.

A interface de uma atividade é estruturada como uma árvore, onde cada nó é uma *view* ou um *viewGroup*. Essa árvore pode ser simples ou complexa dependendo das necessidades do desenvolvedor. Esta pode ser construída utilizando o conjunto GUI Android que possui *layouts* e *widgets* pré-definidos ou criando outros especificamente para o aplicativo. A Figura 2.6 apresenta a estrutura da árvore onde cada nó é uma *view* ou *viewGroup*.

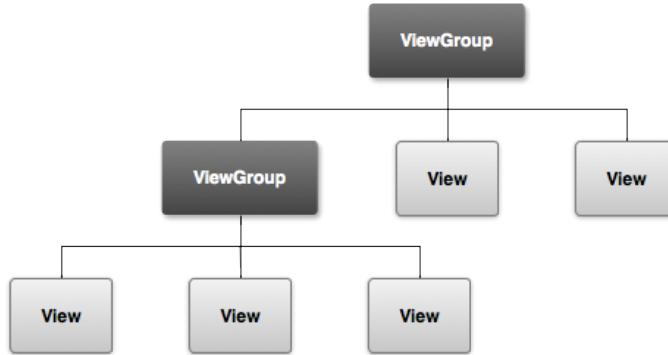


Figura 2.6: Hierarquia *ViewGroup*, (ANDROID, 2012c)

Como é possível visualizar na Figura 2.6, um *viewGroup* permite montar uma interface de usuário através de uma estrutura que gerencia *views* e outros *viewgroups*. A Tabela 2.1 descreve alguns dos *ViewGroups* mais importantes.

<i>Classes</i>	<i>Descrição</i>
FrameLayout	Layout que atua como uma moldura para exibir um único objeto.
Gallery	Um display de rolagem horizontal de imagens, a partir de uma lista.
GridView	Exibe uma grade de rolagem de m colunas e n linhas.
LinearLayout	Um layout que organiza seus filhos em uma única linha horizontal ou vertical. Ele cria uma barra de rolagem se o comprimento da janela excede o tamanho da tela.
ListView	Exibe uma lista simples com barra de rolagem.
RelativeLayout	Permite especificar a localização dos objetos filhos relativamente, uns com os outros (filho A à esquerda do filho B) ou ao pai (alinhado com a parte superior do pai).
ScrollView	Uma coluna de elementos com barra de rolagem.
SurfaceView	Fornece acesso direto a uma superfície de desenho dedicada. Pode alojar views em camadas no topo da superfície, mas é indicado para aplicações que precisam desenhar pixels ao invés de widgets.
TabHost	Fornece uma lista com guia de seleção que permite a aplicação mudar a tela sempre que uma guia é selecionada.
TableLayout	Um layout tabulado com um número arbitrário de linhas e colunas, cada célula com um item de sua escolha. As linhas se redimensionam para caber a coluna. As bordas das células não são visíveis.
ViewFlipper	Uma lista que exibe um item por vez, dentro de uma caixa de texto de uma linha. Pode ser configurado para trocar itens em intervalos de tempos definidos, como uma apresentação de slides.

Tabela 2.1: *User Interface* (ANDROID, 2012c)

Para associar uma árvore à tela para renderização², a atividade deve chamar o método *setContentView()* e passar a referência do nó raiz do objeto. O sistema Android recebe esta referência e usa para validar, medir e desenhar a árvore (ANDROID, 2012c).

²Renderização é o processo de formação de uma interface utilizando dados recém-adquiridos.

O nó raiz da hierarquia requisita aos seus filhos seus componentes de desenho. Cada nó filho é responsável por chamar suas próprias subclasses views pelo método *onDraw()*. Os filhos solicitam o tamanho e a posição ao pai, mas este é quem determina o tamanho final que cada nó filho terá.

2.4.1.3.1 Layout

O *layout* é a arquitetura para interface gráfica de uma atividade. Este define a forma em que os componentes estarão dispostos na *view*, tendo em vista que estes devem se apresentar de forma agradável ao usuário. O *layout* pode ser declarado de duas formas (ANDROID, 2012c):

- Declarar os elementos da interface do usuário em XML – O Android fornece um vocabulário XML simples que corresponde as classes *view* e suas subclasses, tais como os *widgets* e *layouts*. Como o XML possui uma estrutura muito parecida com o HTML, geralmente é utilizado no desenvolvimento de *layouts* Android;
- Instanciar os elementos de *layout* em tempo de execução – O aplicativo pode criar objetos *view* e *viewGroup* (e manipular suas propriedades) da mesma forma feita em Java utilizando bibliotecas como o Swing e SWT.

Cabe ao desenvolvedor escolher a forma que irá declarar seu *layout*. Mas vale ressaltar que utilizar um arquivo XML para criação da tela permite separar a parte visual da lógica de negócios da aplicação, o que além de deixar o código limpo também simplifica sua manutenção. Por esse motivo, o XML será adotado nesse projeto.

Cada elemento em XML é um objeto *view*, *viewGroup* ou descendentes deles. O nome de cada um de seus elementos corresponde exatamente ao que ele referencia na classe Java. Desta forma, um elemento *<TextView>* cria um *TextView* na sua interface de usuário. Quando um recurso de *layout* é carregado o sistema Android inicializa esses objetos em tempo de execução com os elementos correspondentes.

É recomendado definir um ID (de preferência único) para ser associado a um objeto de uma *View*. Esta associação será efetuada no arquivo XML do *layout* da aplicação. Esse procedimento visa permitir ao desenvolvedor não perder a referência no arquivo XML quando houver alterações.

2.4.1.3.2 Widgets

Um *widget* é um objeto que serve como uma interface de interação com o usuário. O Android fornece um conjunto de *widgets* totalmente implementadas (ANDROID, 2012c). A Figura 2.7 apresenta a hierarquia dos *widgets* Android.

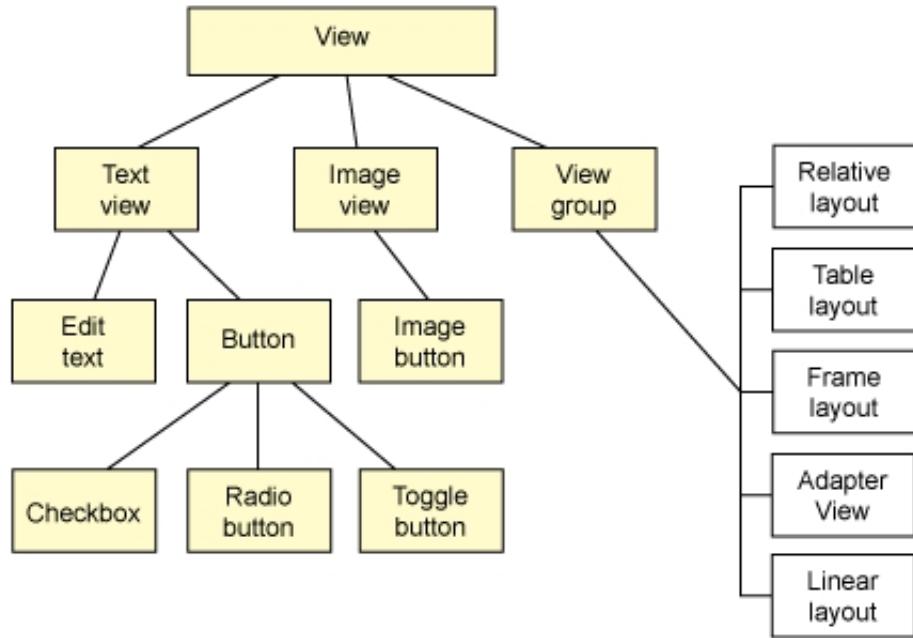


Figura 2.7: Hierarquia dos *Widgets* Android (TEAM, 2011)

O desenvolvedor não é obrigado a utilizar as *widgets* do *kit* do Android, ele pode criar as suas próprias fazendo uso dos gráficos 2D e 3D, mas todas deverão ser armazenadas em classes filhos da *View* e referenciadas no XML.

Para criar seu próprio pacote de elementos *widget* o desenvolvedor deverá criar um *widget* que estende uma *View* ou uma subclasse.

Para usar o *widget* no arquivo layout XML há a necessidade de se criar outros arquivos adicionais:

- *Java implementation file* (arquivo de implementação Java) – é o arquivo que implementa o comportamento do *widget*. O objeto pode ser instanciado no *layout XML* ou pode-se criar um código construtor que recupera todos os valores do atributo do arquivo *layout XML*;
- *XML definition file* (arquivo de definição XML) um arquivo XML em res/layout/ que define o elemento XML usado para instanciar o *widget* e os atributos que este suporta;
- *Layout XML* (opcional) – um arquivo XML opcional dentro de res/layout/ que descreve o *layout* do *widget*. Também pode ser feito no código Java do seu arquivo (ANDROID, 2012g).

2.4.2 Serviços - *Services*

O *Service* é um componente utilizado para executar uma funcionalidade em segundo plano, por tempo indeterminado e gerando um alto consumo de recursos, memória e CPU. Geralmente é iniciado a partir de um *Broadcast Receiver* de forma assíncrona.

Como explicado em developer.Android (ANDROID, 2012c) um serviço pode possuir duas formas:

- Iniciado (*started*) quando um componente da aplicação (como uma atividade) o inicia chamando o método *startService()*. Este executa uma única operação, não deve retornar um resultado para quem o chamou e após a sua conclusão ele deve se encerrar.
- Ligado (*bound*) quando um componente da aplicação se liga à aplicação chamada pelo método *bindService()*. Um serviço ligado oferece uma interface cliente-servidor que permite aos componentes se interagirem com o serviço enviar pedidos, obter resultados e até mesmo fazê-lo através de comunicação entre processos (IPC - *Interprocess Communication*). Um serviço ligado ocorre apenas enquanto outro componente é ligado a ele. Múltiplos componentes podem estar ligados ao serviço de uma só vez, mas todos são desvinculados quando o serviço é destruído.

Ao se criar um Serviço, é fundamental ter em mente que este não deve ser deixado executando ininterruptamente. Assim que o serviço completar seu objetivo este deve ser finalizado. Caso este necessite executar uma função várias vezes, durante o tempo em que ficar ocioso deve ser finalizado e no momento que for necessário deve ser reiniciado.

A classe Service faz parte do ciclo de vida dos processos controlados pelo sistema operacional Android, sendo semelhante ao da *Activity*, mas difere em:

- Por não possuir interface com o usuário não há necessidade dos métodos *onResume()*, *onPause()* e *onStop()*;
- O método *onBind()* pode iniciar um serviço que ainda não está sendo executado, mas ele só o faz no método *onCreate()*;
- O método *onDestroy()* pode ser chamado quando o serviço estiver próximo de seu fim, ou ser encerrado pelo Android quando houver insuficiência de memória, neste caso o sistema operacional tentará restaurar o serviço assim que a escassez for解决 (solved) pela solução reiniciando pelo método *onStart()*.

A Figura 2.8 apresenta o ciclo de vida do serviço. O diagrama à esquerda mostra o

ciclo quando o serviço é criado com `startService()` e o da direita quando o serviço é criado com `bindService()`.

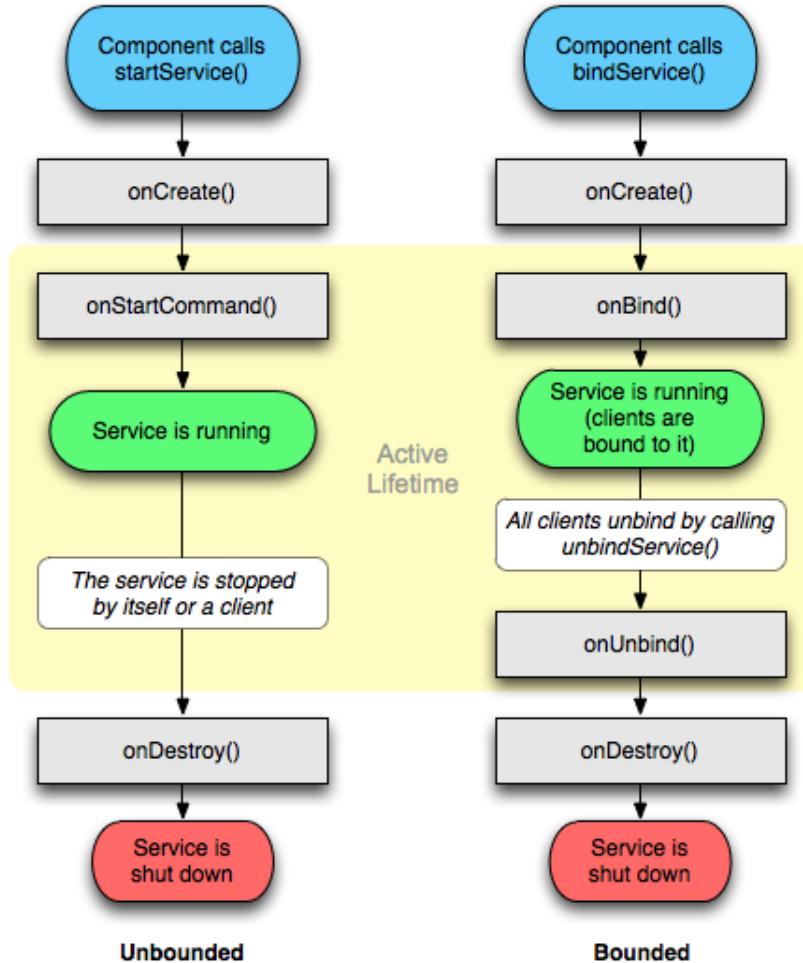


Figura 2.8: Métodos de retorno de chamada de um Serviço (ANDROID, 2012c)

Quando usar um serviço ou uma *thread*?

Serviço, como dito anteriormente, é um componente que roda em segundo plano consumindo recursos, desta forma só deve ser criado se for realmente necessário.

Threads são fluxos de execução que rodam um processo (aplicação) de forma assíncrona e por tempo indeterminado. Em uma aplicação Android podemos criar uma *thread* utilizando a classe *Thread* tradicional da API Java. Por exemplo, se quisermos realizar um trabalho fora do processo principal podemos criar uma *thread* pelo método `onCreate()`, iniciar a sua execução com o método `onStart` e pará-lo com o método `onStop`. Contudo, o site oficial recomenda a utilização das classes *AsyncTask* ou *HandlerThread* em substituição à classe *Thread* tradicional.

2.4.3 Provedor de Conteúdo - *Content Provider*

Um provedor de conteúdo faz com que um conjunto de dados específicos de um aplicativo fique disponível para outros. Os dados podem ser armazenados no arquivo do sistema, em um banco de dados SQLite ou de qualquer outra forma lógica.

O provedor de conteúdo estende a classe base *Content Provider* para implementar um conjunto padrão de métodos que permitam a outros aplicativos recuperar e armazenar dados do tipo que ele controla. Entretanto, as aplicações não chamam este método diretamente, elas usam o objeto *Content Resolver* que por sua vez chama os métodos.

Um *Content Resolver* pode se comunicar com qualquer provedor de conteúdo. Ele coopera com o gerenciador de provedor entre os processos de comunicação que os envolva(ANDROID, 2012c).

O desenvolvedor que deseja tornar público os dados de sua aplicação pode:

- Criar um provedor de conteúdo próprio que seria uma subclasse, ou
- Adicionar os dados a um provedor já existente, uma vez que é permitido fazê-lo.

Isto é possível porque a forma de comunicação da aplicação com o provedor de conteúdo é personalizada e faz uso do URI através da classe android.net.Uri.

2.4.4 Receptores de Broadcast - *Broadcast Receiver* ou Receptores de Intenções - *Intent Receiver*

Esta classe é responsável pela reação de uma aplicação a um determinado evento gerado por uma *Intent*.

Executada sempre em segundo plano e por pouco tempo, o *Broadcast Receiver* recebe uma *Intent* e a trata. É comum este gerar um novo *Service* ou abrir uma *Activity*, sem a necessidade que esta última esteja ativa.

2.5 O arquivo AndroidManifest.xml

Segundo Android (2012c), o arquivo AndroidManifest.xml deve constar em toda aplicação Android em seu diretório raiz e com essa denominação. O manifesto apresenta informações essenciais sobre o aplicativo. Estas informações devem ser conhecidas antes de se executar o código, isto porque o sistema centraliza as configurações da aplicação. A lista a seguir descreve as principais informações presentes neste arquivo:

- O nome do pacote Java para o aplicativo – Este nome serve como um identificador exclusivo para o aplicativo;
- Descreve os componentes da aplicação – As atividades, serviços, *broadcast receiver* e *content provider* de que a aplicação é composta. Ele cita as classes que implementam cada um dos componentes e publica as suas capacidades. Estas declarações deixam o sistema Android saber o que os componentes são e em que condições eles podem ser lançados;
- Determina quais processos irão hospedar componentes da aplicação;
- Declara as permissões que o aplicativo deve ter para acessar partes protegidas da API e interagir com outras aplicações;
- Declara também quais permissões outros aplicativos são obrigados a ter de forma a interagir com os componentes do aplicativo;
- Lista as classes de instrumentação que fornece informações de perfil e outras que o aplicativo é executado. Estas declarações estão presentes no manifesto somente enquanto o aplicativo é desenvolvido e testado, eles são removidos antes da aplicação ser publicada;
- Declara o nível mínimo da API do Android que o aplicativo requer;
- Lista as bibliotecas que o aplicativo deve estar ligado.

2.5.1 Estrutura do arquivo de Manifesto

O Código 2.1 apresenta a estrutura geral do arquivo *AndroidManifest.xml* e cada elemento que este pode conter.

```

1 <?xml version="1.0" encoding="utf-8"?>
2
3 <manifest>
4
5   <uses-permission />
6   <permission />
7   <permission-tree />
8   <permission-group />
9   <instrumentation />
10  <uses-sdk />
11  <uses-configuration />
12  <uses-feature />
13  <supports-screens />
14  <compatible-screens />
15  <supports-gl-texture />
16
17  <application>
18
```

```

19   <activity>
20     <intent-filter>
21       <action />
22       <category />
23       <data />
24     </intent-filter>
25     <meta-data />
26   </activity>
27
28   <activity-alias>
29     <intent-filter> . . . </intent-filter>
30     <meta-data />
31   </activity-alias>
32
33   <service>
34     <intent-filter> . . . </intent-filter>
35     <meta-data/>
36   </service>
37
38   <receiver>
39     <intent-filter> . . . </intent-filter>
40     <meta-data />
41   </receiver>
42
43   <provider>
44     <grant-uri-permission />
45     <meta-data />
46   </provider>
47
48   <uses-library />
49
50 </application>
51
52 </manifest>

```

Código 2.1: Estrutura geral do arquivo *AndroidManifest.xml*.

Neste capítulo foi apresentado o Sistema Operacional Android e como este é estruturado. Este conhecimento é necessário para podermos desenvolver aplicativos capazes de fazer bom uso do potencial disponível no dispositivo, procurando minimizar a utilização de recursos e maximizar seu desempenho.

3 O AMBIENTE DE DESENVOLVIMENTO

No Capítulo 2 foram apresentados os componentes e recursos disponibilizados pelo Android, conhecimento base para criação de programas para a plataforma. Neste capítulo será apresentado o ambiente de desenvolvimento de aplicativos Android e as principais ferramentas que o forma. Em seguida, será descrito o processo de instalação e configuração deste ambiente.

As ferramentas de desenvolvimento disponíveis para o Android permitem a criação de aplicativos flexíveis e leves para diferentes plataformas. Para isto, é necessário que estas ferramentas sejam instaladas e configuradas com a finalidade de suprir todas as necessidades do desenvolvedor. Ao final desse processo, a união destas formará o chamado ambiente de desenvolvimento.

É comumente encontrado na literatura autores que utilizam a plataforma Windows como base de seu ambiente de desenvolvimento, como nos livros Google Android (LECHETA, 2009), Criação de Aplicativos para Celulares (JOBSTRAIBIZER, 2009), Android para Desenvolvedores (PEREIRA; SILVA, 2009), Desenvolvimento de Aplicações Android (ROGERS et al., 2009), além do site da tecmundo (STARCK, 2010). Este trabalho terá seu ambiente de desenvolvimento baseado na plataforma Linux, objetivando demonstrar a implantação deste ambiente em uma plataforma Open Source.

Tendo como principal fonte o site oficial Android *Developers* (ANDROID, 2012c), as seções seguintes descreverão as ferramentas que irão compor o ambiente de desenvolvimento deste projeto.

3.1 Eclipse

O Eclipse é uma plataforma de desenvolvimento aberta onde é possível criar novas aplicações. Esta plataforma facilita a criação e o gerenciamento dos arquivos e pacotes de um projeto, possui uma interface que torna processos repetitivos do desenvolvimento automatizados, como, por exemplo, a criação de cabeçalhos padronizados em novos arquivos de código fonte. Também auxilia o desenvolvedor verificando automaticamente se as estruturas de dados estão corretas, se variáveis utilizadas foram declaradas, procura por

erros de sintaxe no código e aponta as possíveis soluções, dentre outras funções.

Esta plataforma foi criada inicialmente para atender aos desenvolvedores que utilizavam a linguagem de programação Java, e, posteriormente, passou a atender a outras linguagens como C, C++, Python, Ruby, dentre outras. Desta forma, é amplamente utilizado no processo de desenvolvimento de aplicações.

3.2 *Android Development Tools*(ADT)

O ADT é um *plugin* gratuito responsável por estabelecer uma interface entre a plataforma Eclipse e o Android SDK. Desta forma, este possibilita a utilização dos recursos presentes nesta plataforma para o desenvolvimento de aplicações Android com amplo suporte as ferramentas presentes no SDK, ou seja, este *plugin* é necessário para que o Eclipse possa criar aplicativos Android.

A utilização do ADT conjuntamente com a plataforma Eclipse é recomendada porque permite a criação das aplicações de forma mais ágil. Estes tornam parte do processo de desenvolvimento automatizado, sendo assim, o tempo que seria gasto em uma tarefa repetitiva passará a ser empregado em outra, otimizando o desenvolvimento.

3.3 *Software Development Kit* - SDK

O Android SDK é um *kit* de desenvolvimento, disponibilizado gratuitamente pela Google, formado por uma grande variedade de ferramentas que auxiliam no desenvolvimento de aplicativos para o Sistema Operacional Android. Este *kit* é de suma importância para a criação de qualquer aplicação, uma vez que todas as ferramentas básicas fazem parte dele.

Uma informação importante a respeito do SDK é que todas as suas ferramentas podem ser acessadas via terminal. Porém, o Eclipse disponibiliza uma interface que facilita a utilização destas sem a necessidade de acessar o terminal. Nesse caso, a escolha de qual método utilizar fica a cargo do desenvolvedor.

No site Android *Developers* é possível encontrar a versão mais recente do SDK para os Sistemas Operacionais Windows, Linux ou Mac OS X.

Dentre as ferramentas presentes no Android SDK as que serão mais utilizadas durante o desenvolvimento serão: o Gerenciador de Dispositivo Virtual Android (AVD Manager), o Emulador e o Monitor de Depuração de Serviço Dalvik (DDMS). Cada uma dessas ferramentas serão abordadas a seguir.

3.3.1 *Android Virtual Device Manager (AVD Manager)*

O *Android Virtual Device Manager (AVD Manager)* é uma ferramenta presente no Android SDK que tem como objetivo disponibilizar uma interface onde o desenvolvedor poderá criar e configurar diferentes modelos de hardware e software. Estes modelos serão utilizados posteriormente pelo Emulador para simular um dispositivo móvel real que utilizará o Sistema Operacional Android.

No *AVD Manager* é possível escolher qual a versão do Sistema Operacional Android o modelo possuirá, qual a resolução da tela o aparelho utilizará, se este possuirá suporte para GPS e Acelerômetro, dentre outras coisas.

Os cartões de memória são utilizados em smartphones e tablets com o objetivo de expandir o espaço de memória do aparelho. Ao se criar um modelo pelo *AVD Manager*, um dos passos mais importantes é a definição do cartão de memória virtual que o dispositivo irá utilizar, chamado de *SD Card*. Este passo será importante para o desenvolvimento pois será nesse local que o Eclipse obrigatoriamente irá armazenar o aplicativo gerado a partir do código fonte que, em seguida, será instalado no Emulador.

3.3.2 *Emulador*

O Emulador simula diferentes plataformas de dispositivos móveis baseadas no modelo criado anteriormente pelo *AVD Manager*, como pode ser visto na Figura 3.1. Este permite avaliar se a aplicação vai se comportar de forma adequada em um ambiente próximo a realidade, podendo ser inicializado pelo *AVD Manager* ou por linha de comando.

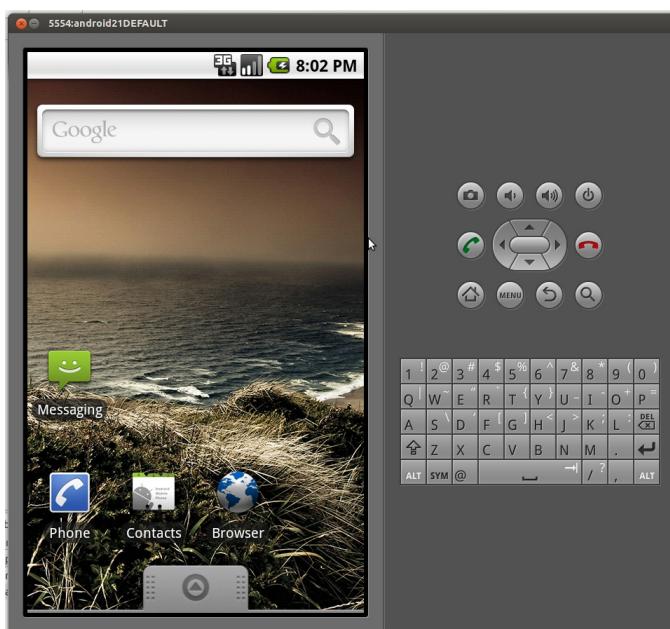


Figura 3.1: O emulador da versão 2.1 em execução

A inicialização do Emulador é lenta uma vez que este simula toda a inicialização do Sistema Operacional. Felizmente, após a inicialização do Emulador, este não precisará ser reiniciado a cada teste da aplicação. Desta forma, ao efetuar uma atualização na aplicação basta executá-la pelo Eclipse que esta será atualizada automaticamente. Também é possível emular vários modelos criados pelo AVD em paralelo, possibilitando ao desenvolvedor avaliar rapidamente sua aplicação em diferentes situações.

Caso haja a necessidade de se realizar alterações na configuração de um modelo, o Emulador deverá ser fechado para que as estas entrem em vigor.

3.3.3 *Dalvik Debug Monitor Service - DDMS*

É uma ferramenta para depuração das aplicações, que possibilita o acompanhamento da execução dos processos. Este pode ser executado via terminal ou pelo Eclipse.

O DDMS atua como um intermediário entre a plataforma Eclipse e os aplicativos em execução no dispositivo virtual. No Android cada aplicativo é executado em seu próprio processo que possui sua própria máquina virtual. Cada processo é depurado em uma porta diferente.

Essa ferramenta permite simular o envio de SMS, recebimentos de chamadas, dados de localização, pode analisar processos, *threads*, memória *heap*, possibilita a visualização dos *logs* da aplicação, etc. Além da depuração das informações, esta pode também depurar a execução a partir de um dispositivo conectado no computador.

A ferramenta DDMS pode ser acessada pela perspectiva do Eclipse, disponível após a instalação do *plugin ADT*. Para iniciar o seu uso, deverá ser utilizado o atalho de teclado “Ctrl+3” e na janela que se abrir deverá ser digitado “DDMS”.

A Tabela 3.1 descreve as janelas visualizadas na perspectiva DDMS:

Janela	Descrição
Devices	Exibe os Emuladores ativos. Útil para verificar em que porta o Emulador foi aberto.
Emulator Control	Simula uma mensagem SMS ou uma ligação para o Emulador.
File Explorer	Permite navegar na estrutura de diretórios do Emulador. Também permite enviar e baixar arquivos do Emulador.
LogCat	Esta janela permite visualizar as mensagens (<i>logs</i>) do Sistema.
Threads	Utilizada para visualizar as <i>threads</i> ativas na JVM.
Heap	Utilizada para monitorar a memória <i>heap</i> do Emulador.

Tabela 3.1: Janelas visualizadas na perspectiva DDMS (LECHETA, 2009, 53)

3.4 Instalação e Configuração

O ambiente de desenvolvimento e suas principais ferramentas foram apresentadas anteriormente. Nesta seção será demonstrado o processo de instalação e configuração deste ambiente na plataforma Linux.

Inicialmente será apresentado como obter o Eclipse, como instalar e configurar o *plugin* ADT e o Android SDK. Por fim, será demonstrado como criar um novo projeto.

3.4.1 Requisitos do Sistema

A seguir são apresentados os requisitos necessários para instalar o Android SDK conforme disponibilizado em *Android Developers* (ANDROID, 2012i).

- Sistemas Operacionais:
 - Windows XP (32 bits), Vista (32 e 64 bits) e Seven (32 e 64 bits);
 - Mac OS X 10.5.8 ou posterior (somente x86);
 - Linux Ubuntu 8.04 ou posterior;
- Ambiente de desenvolvimento:
 - Eclipse versão 3.5 ou superior;
 - *Plugin* ADT;
 - Android SDK;
 - JDK³ 5 ou superior (apenas a JRE⁴ não é suficiente).

Caso a distribuição Linux utilizada seja 64 *bits*, será necessário instalar o pacote *ia32-libs*. Esse pacote habilitará o funcionamento de aplicativos 32 *bits*.

3.4.2 Instalação do Eclipse

O SDK do Android necessita da versão igual ou posterior a 3.5 do Eclipse. Este pode ser obtido no endereço '<http://eclipse.org/downloads>' onde deverá ser escolhida a

³*Java Development Kit* (JDK) significa Kit de Desenvolvimento Java, e é um conjunto de utilitários que permitem criar sistemas de software para a plataforma Java. É composto por compilador e bibliotecas Java.

⁴*Java Runtime Environment* (JRE) significa Ambiente de Tempo de Execução Java, que é utilizado para executar as aplicações da plataforma Java. É composto por bibliotecas Java e pela Máquina virtual Java (JVM).

versão IDE compatível com o sistema operacional utilizado, denominada: “Eclipse IDE for Java Developers”.

O Eclipse poderá ser instalado no Sistema Operacional Linux via terminal utilizando o comando:

```
1 sudo apt-get install eclipse
```

3.4.2.1 Instalação do *plugin* ADT no Eclipse

1. Inicie o Eclipse;
2. No menu **Help** escolher a opção **Install New Software...**, conforme a Figura 3.2;

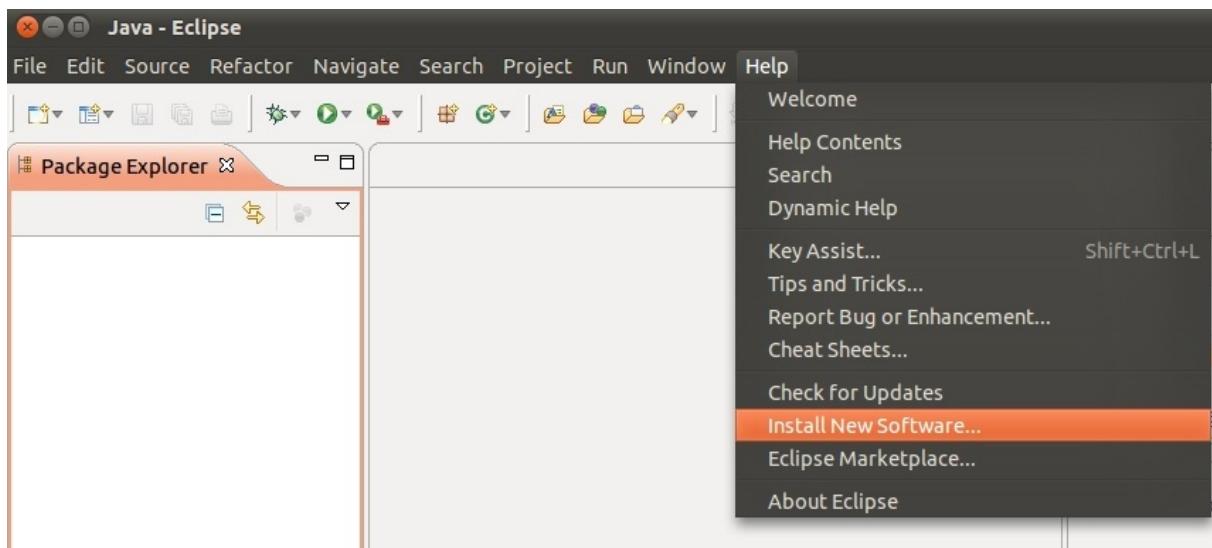


Figura 3.2: Instalação do *plugin* ADT - *Help > Install New Software*

3. Na nova tela, clicar no botão **Add**;

- (a) Na tela seguinte, representada pela Figura 3.3, preencher os campos e clicar em **OK**;

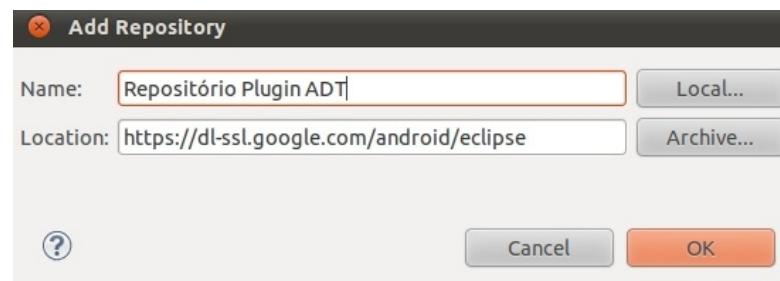


Figura 3.3: Instalação do *plugin* ADT - Preencher dados do respositório

- i. *Name* -> Nome para o repositório;

- ii. *Location* -> digitar o endereço: <https://dl-ssl.google.com/android/eclipse>;
4. Ao retornar a tela anterior, marcar a opção ***Developer Tools*** e pressionar o botão ***Next***, conforme Figura 3.4;

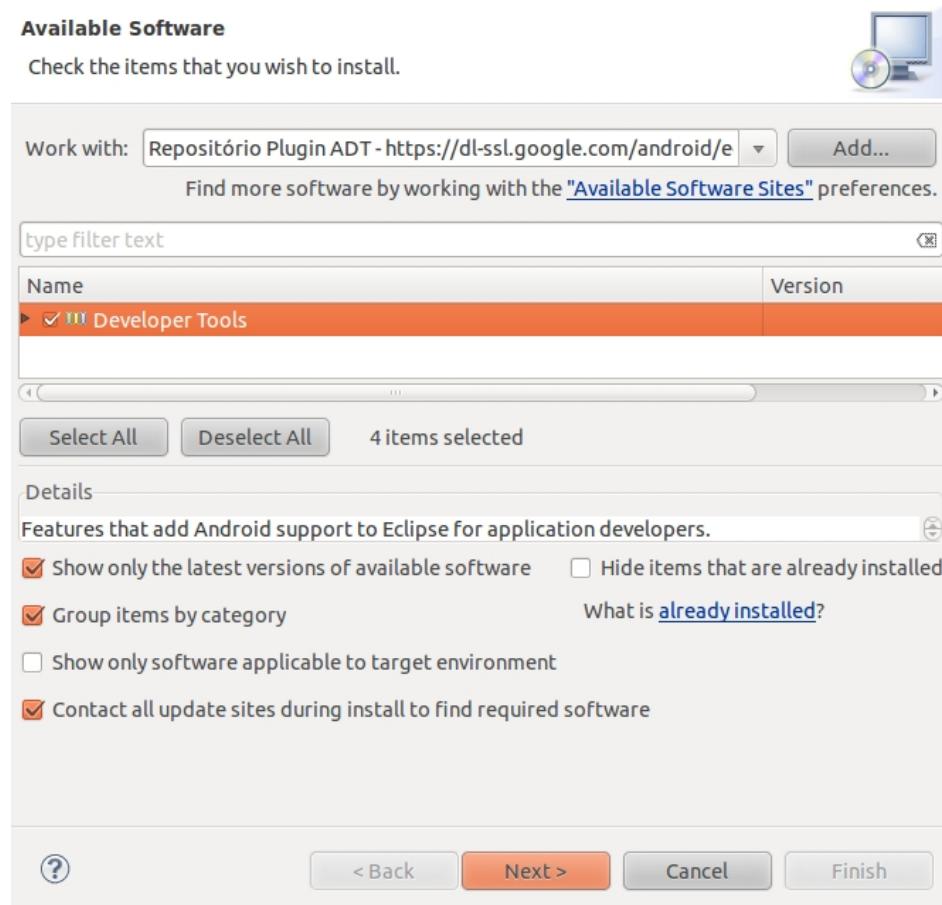


Figura 3.4: Instalação do *plugin* ADT - Marcar a opção *Developer Tools*

5. Na tela seguinte, clicar novamente no botão ***Next***;
6. Feito isso, leia os termos do contrato de uso e, caso concorde, marcar a opção ***I accept the terms of the license agreements***;
7. Após aceitar os termos, clicar no botão ***Finish*** para começar a instalação do *Plugin ADT*.
- Observações:
 - Caso apareça uma mensagem de aviso durante a instalação, clicar no botão ***OK*** para continuar;
 - Após instalar é necessário reiniciar o Eclipse para que as modificações sejam estabelecidas, clicar no botão ***Restart Now*** na caixa de diálogo que aparecer.

3.4.3 Configurando o SDK

Após reiniciar o Eclipse a tela representada pela Figura 3.5 irá aparecer. Nela você poderá escolher entre instalar um novo SDK ou localizar um SDK já existente.

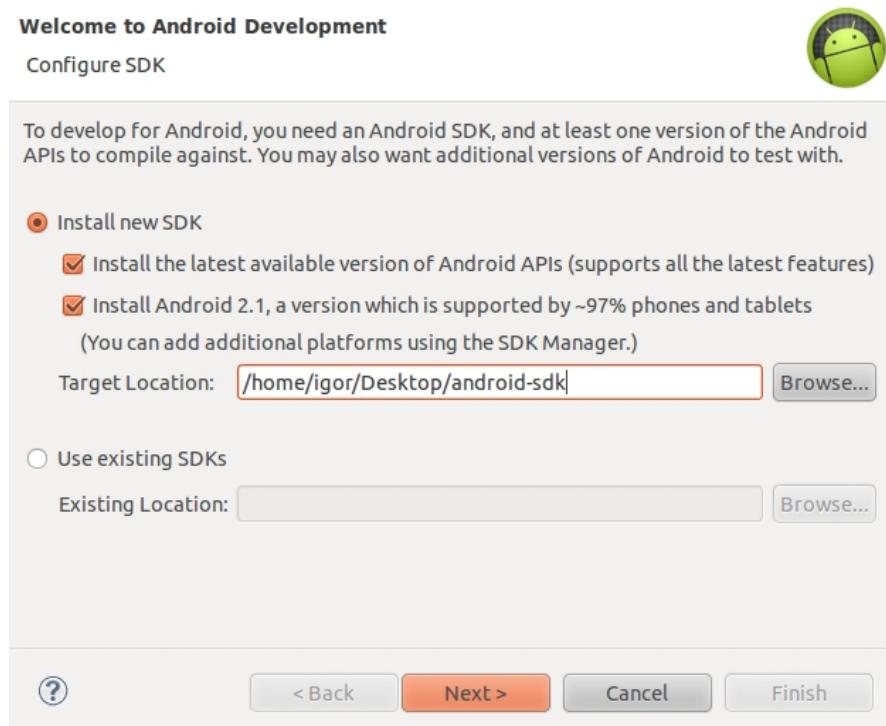


Figura 3.5: Configuração do Android SDK

Caso você nunca tenha instalado o SDK do Android ou não o possua em seu computador, escolha a primeira opção ***Install new SDK***. Após esta opção ser marcada você deverá:

1. Caso queira transferir a versão mais recente do Android, que suporta aplicações de todas as versões anteriores, marcar a opção ***Install the latest available version of Android APIs***;
2. Caso queira transferir a versão 2.1, versão com maior suporte de celulares e *tablets* (aproximadamente 97%), marcar a opção ***Install Android 2.1***;
3. Informar o local onde deverão ser guardados os arquivos referentes ao SDK;
4. Clicar no botão ***Next***;
5. Escolher se deseja contribuir com as estatísticas de uso marcando a opção ***Yes*** ou a opção ***No***, na tela seguinte. Feito isso, Clicar no botão ***Finish***;

6. A tela representada pela Figura 3.6 deve aparecer. Nela, leia os termos do contrato de uso e, caso concorde, marcar a opção **Accept All**. Em seguida, clicar no botão **Install**.

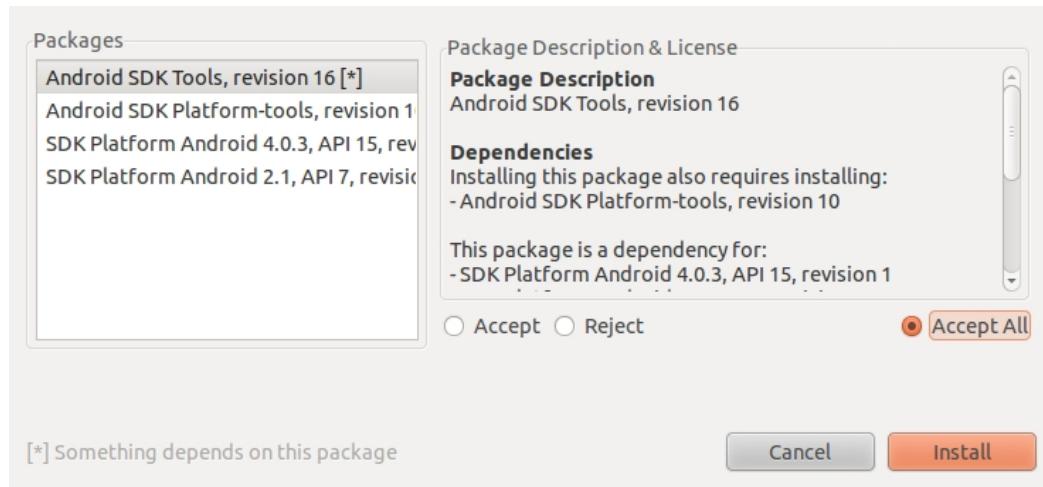


Figura 3.6: Termos do contrato de uso do Android SDK

- Observação:

- Outras versões do Android poderão ser transferidas posteriormente utilizando o **SDK Manager**;

Caso já exista uma pasta que contenha os arquivos referentes ao Android, a opção **Use Existing SDK** deverá ser marcada e posteriormente deve-se informar o local onde esta se encontra.

3.4.3.1 Utilizando o Android SDK Manager

Após a instalação do *Plugin ADT*, novas funcionalidades são incorporadas a interface do Eclipse. Dentre elas, se destacam dois novos botões, um para acessar o **Android SDK Manager** e outro para acessar o **Android Virtual Device Manager**, como representados na Figura 3.7:

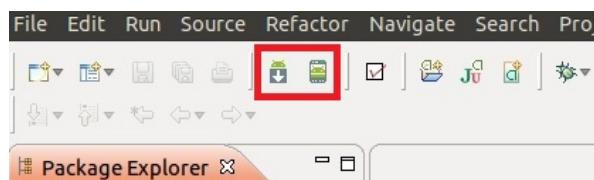


Figura 3.7: Novas funcionalidades do Eclipse

Ao clicar no primeiro botão, a tela do **Android SDK Manager** se abrirá. Com o objetivo de gerenciar os recursos disponíveis para o SDK, esta possui uma interface simples e intuitiva, como pode ser visto na Figura 3.8.

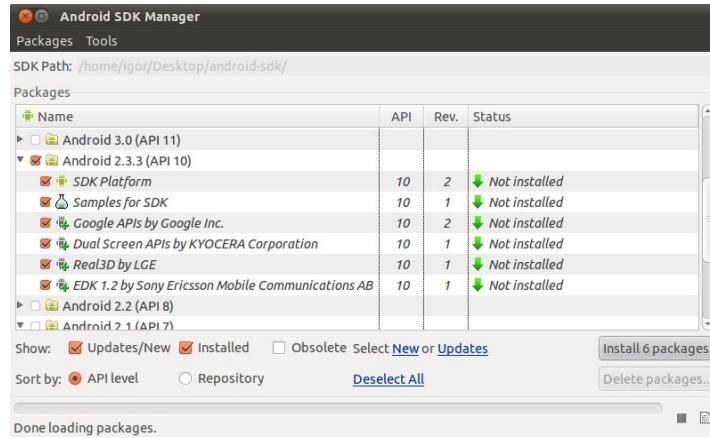


Figura 3.8: Interface do *Android Sdk Manager*

Para transferir ou atualizar algum pacote do SDK é necessário que este seja marcado dentre os pacotes listados nesta interface, podendo selecionar mais de um deles. Nesta lista é possível observar na coluna **API** o nível dos pacotes. Essa informação deve ser considerada pois cada aplicação Android deve conter o nível a qual pertence e esta só funcionará em plataformas Android do mesmo nível ou superior. Ou seja, se sua aplicação pertence ao nível 7 (Android 2.1), ela não poderá ser executada no nível 6 (Android 2.0), mas poderá ser executada no nível 8 (Android 2.2).

Na coluna **status** pode-se verificar a situação do pacote, se este está instalado, atualizado, ou ainda se é compatível com a plataforma escolhida.

Feita a seleção, basta clicar no botão **Install X packages**, onde X representa o número de pacotes selecionados. Uma nova tela deve aparecer contendo os termos do contrato de uso dos pacotes, que devem ser aceitos para a conclusão da instalação.

3.4.3.2 *Android Virtual Device Manager*

A interface do **Android Virtual Device Manager**, Figura 3.9, pode ser visualizada clicando na tela principal do Eclipse, no botão localizado no lado direito na seção anterior. Nesta é possível gerenciar os dispositivos virtuais criados e inicializá-los.



Figura 3.9: Interface do *Android Virtual Device Manager*

Para criar um novo dispositivo virtual você deve:

1. Clicar no botão **New** e uma nova tela, onde será efetuado o cadastro, aparecerá. Conforme Figura 3.10;

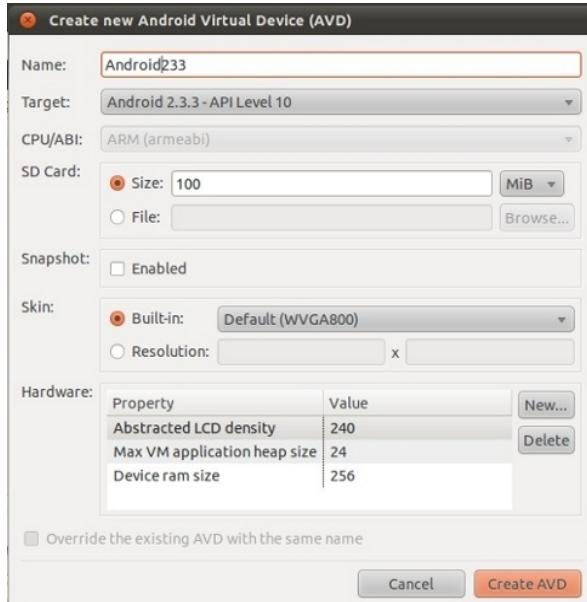


Figura 3.10: Criando novo dispositivo virtual

2. Na nova tela digitar um nome para o novo AVD;
 - (a) Na opção **Target** escolher a versão do Android;
 - (b) Na opção SD **Card**, especificar o tamanho do cartão virtual que será criado ou o local onde haja um arquivo de cartão virtual;
 - (c) Na opção **Hardware** é possível adicionar as opções de hardware que o Emulador será capaz de simular;
3. Clicar em *Create AVD*.

Feito isso, um novo dispositivo virtual será criado. Para inicializá-lo, selecione-o na lista de dispositivos e clique no botão **Start....**. Na tela seguinte, clique no botão **Launch**. O Emulador será chamado, como pode ser visto na Figura 3.1.

3.5 Criando uma aplicação no Android Eclipse

1. No menu **File** abrir submenu **New** e clicar na opção **Project...**;
2. Selecionar **Android Project** na pasta **Android**. Em seguida, clicar no botão **Next**;

3. Na tela seguinte, informar o nome do projeto no campo ***Project Name*** e clicar em ***Next***;
4. Agora informar a qual plataforma sua aplicação irá pertencer, conforme a Figura 3.11

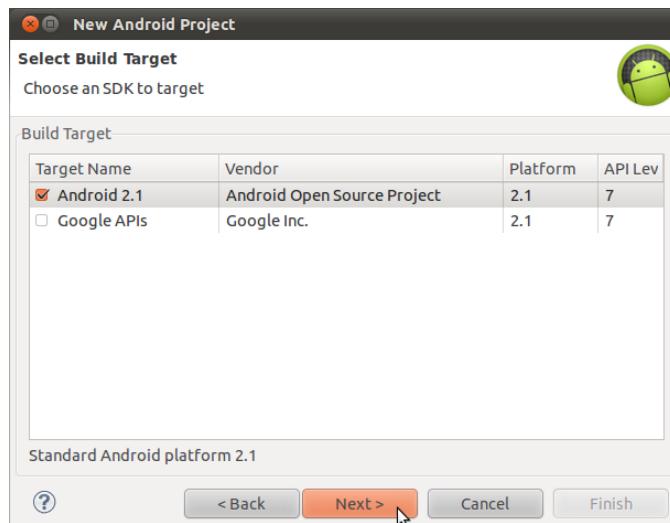


Figura 3.11: Escolha de plataforma.

5. Na nova tela preencher os campos descritos a seguir, conforme Figura 3.12:

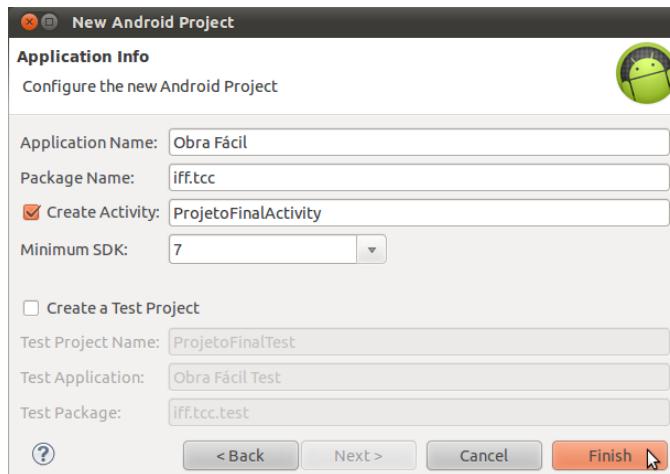


Figura 3.12: Criação de um novo projeto.

- (a) No campo ***Application Name*** deverá ser informado qual será o nome da aplicação. Este será visível na lista de aplicações do dispositivo e poderá ser alterado posteriormente;
- (b) O campo ***Package Name*** deve conter o nome do pacote que contém a *activity* principal e deve ser único. Este nome deve seguir o padrão de nomeclatura de pacotes Java;

- (c) Em **Create Activity** deverá ser informado o nome da atividade principal do projeto;
6. Clicar em **finish**;
 7. Caso o novo projeto apresente algum erro, clicar com o **botão direito do mouse** na **pasta do projeto** listada em **Package Explorer**, clicar no menu **Android Tools** e, em seguida, escolher a opção **Fix Project Properties**. Feito isso, automaticamente o sistema irá procurar por erros e, caso encontre, tentará solucioná-los.
 8. Caso o erro persista, tente adicionar manualmente a biblioteca referente a versão do Android utilizada pelo projeto.
 - (a) Clicar na **pasta do projeto** e, em seguida, no menu **Project**, escolher a opção **Properties**;
 - (b) Na **aba esquerda** escolher a opção **Java Build Path**;
 - (c) No **centro da página** escolher na **aba superior** a opção **Order and Export** e marcar a opção correspondente a **versão do projeto**;
 - (d) Clicar em **OK**.

Após a criação do projeto a estrutura de diretórios e arquivos gerados deve possuir a estrutura apresentada pela Figura 3.13.

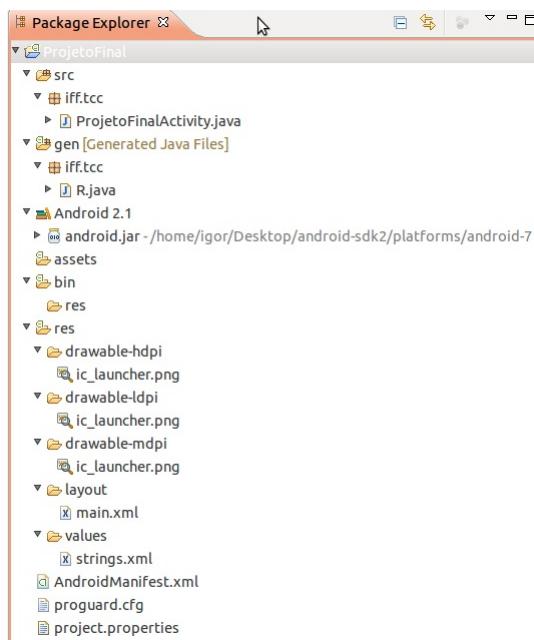


Figura 3.13: Estrutura do Projeto Android.

Ao concluir a criação do projeto ele pode ser executado ao clicar no menu **Run**, em seguida no submenu **Run as** e escolher a opção **Android Application**.

4 A CRIAÇÃO DE UM APLICATIVO PARA O ANDROID

Este Capítulo aborda as etapas do desenvolvimento de um aplicativo Android, como criação e manipulação de banco de dados, implementação do *layout* e o controle da interface entre estes. Além disso, as etapas de depuração, teste e publicação serão descritas. Os exemplos apresentados foram extraídos de (OGLIARI, 2010), (SILVA, 2010), (SILVEIRA, 2010) e (ANDROID, 2012b).



Figura 4.1: Etapas para criação de uma aplicação Android

A Figura 4.1⁵ apresenta e descreve as etapas para a criação de um aplicativo para o sistema operacional Android.

Na primeira etapa deve-se preparar o ambiente onde o aplicativo será criado, através da instalação e configuração das ferramentas necessárias. Esta etapa foi apresentada no Capítulo 3. As demais são abordadas nas próximas seções.

A fase de desenvolvimento aborda desde a criação do projeto até a implementação das funcionalidades do aplicativo.

Com o aplicativo criado, a etapa seguinte consiste da depuração deste a procura de problemas de desempenho e da criação de testes automatizados para avaliar se o mesmo está atendendo aos requisitos estabelecidos.

A última etapa consiste da publicação do aplicativo, ou seja, a liberação para que outros usuários possam acessá-lo através da loja virtual do Android, Google Play.

4.1 Diretórios do Projeto

O Eclipse pode ser utilizado para criar automaticamente um projeto com seus principais diretórios, como pode ser observado no final do Capítulo 3, Figura 3.13. A seguir o conteúdo e objetivo de cada uma destas pastas são descritos:

4.1.1 A pasta src

Esta pasta tem por objetivo alocar as classes Java do projeto. Inicialmente, o Eclipse cria neste diretório uma classe com nomeclatura idêntica a do projeto que exemplifica a implementação de uma *activity*. O Código 4.1 apresenta uma forma genérica desta classe.

```

1 package <nome_do_pacote>;
2 import android.app.Activity;
3 import android.os.Bundle;
4 import android.widget.TextView;
5 public class <nome_do_aplicativo> extends Activity {
6     /** Called when the activity is first created */
7     @Override
8     public void onCreate(Bundle savedInstanceState){
9         super.onCreate(savedInstanceState);
10        TextView view = new TextView (this);
11        view.setText ("‘‘texto’’);
12        setContentView(view);
13    }
14 }
```

Código 4.1: *Activity* exemplo criada pelo Eclipse

⁵ Adaptada de The Development Process for Android Applications em (ANDROID, 2012k)

A linha 1 do Código 4.1 corresponde a chamada do pacote principal do projeto criado.

Nas linhas 2, 3 e 4 são realizados os *imports* necessários para o funcionamento da classe.

A linha 8 apresenta o início da implementação do método *onCreate()*, responsável por inicializar os componentes essenciais da *activity*. Este recebe como parâmetro um objeto do tipo *Bundles*, que atua como um “pacote” que guarda o estado da atividade quando esta é passada para segundo plano (ANDROID, 2012j).

Nas linhas 10 e 11 um objeto do tipo *TextView* é criado e este recebe um texto. Na linha 12 o método *setContentView(view)*, onde *view* é o objeto *TextView*, é responsável por exibir na tela da aplicação o texto.

4.1.2 A pasta gen (*Generated Java Files*)

Este diretório contém a classe R, uma classe obrigatória que possui as referências dos recursos da aplicação. Esta não deve ser alterada manualmente, sendo gerenciada pelo Eclipse, desta forma, quando se altera, adiciona ou remove um arquivo das pastas src, assets ou res, a classe R é atualizada de automaticamente.

Como é possível observar no Código 4.2, cada recurso listado nesta classe possui um valor hexadecimal único que é utilizado pelo gerenciador de recursos do Android para carregar dados reais, como *strings*, cores, imagens ou outros recursos presentes no pacote da aplicação.

```

1  /* AUTO-GENERATED FILE.  DO NOT MODIFY.
2  *
3  * This class was automatically generated by the
4  * aapt tool from the resource data it found.  It
5  * should not be modified by hand.
6  */
7 package iff.tcc.obrafacil;
8 public final class R {
9     public static final class array {
10         public static final int unidades=0x7f050000;
11     }
12     public static final class attr {
13     }
14     public static final class categoria_cadastro {
15         public static final int btnCadastrarCategoria=0x7f080001;
16         public static final int nome=0x7f080000;
17     }

```

Código 4.2: A classe R

Outras informações sobre esta classe, como o acesso a tais recursos, serão disponibilizadas durante o capítulo.

4.1.3 A pasta assets

Nesta pasta devem ser guardados todos os arquivos customizados, ou seja, recursos externos que serão acessados quando chamados no código do aplicativo.

Por exemplo, caso o desenvolvedor queira utilizar um estilo de fonte diferente dos disponibilizados pelo Android, este deverá ser salvo nesta pasta. O desenvolvedor pode criar um novo estilo de fonte, neste caso é importante que este conte cole todos os caracteres, incluindo os especiais.

4.1.4 A pasta res (*resources*)

Arquivos contendo recursos utilizados pelo aplicativo, como imagens, *layouts* e arquivos de internacionalização, devem ser adicionados a esta pasta. Quando ocorre uma adição de um novo recurso a esta pasta ele é mapeado e adicionado a lista de recursos na Classe R.

A pasta res é sub-dividida nas pastas *layout*, *drawable* e *values* que serão apresentadas a seguir.

4.1.4.1 Pasta *Layout*

Nesta pasta são armazenados os arquivos XML que definem os diversos *layouts* utilizados pela aplicação.

Quando se cria um novo projeto Android pelo Eclipse, este gera um *layout* de exemplo chamado main.xml. O Código 4.3 apresenta a estrutura deste *layout*.

```

1 <?xml version="1.0" encoding = "utf-8"?>
2 <LinearLayout
3     xmlns: android="http://schemas.android.com/apk/res/android"
4
5     android:orientation = "vertical"
6     android:layout_width = "fill_parent"
7     android:layout_height = "fill_parent" />
8
9     <TextView
10        android:layout_width = "fill_parent"
11        android:layout_height = "wrap_content"
12        android:text="@string/hello" />
13 </LinearLayout>
```

Código 4.3: Pasta res

Na linha 2 é declarado o gerenciador de *layout*, definido para organizar a forma em que as informações adicionadas a interface serão dispostas. Neste caso, foi escolhido o

LinearLayout onde as informações serão adicionadas linearmente na interface na vertical ou horizontal. Na linha 5, foi definido que as informações serão dispostas na vertical.

Na linha 9, um elemento *TextView*, responsável por mostrar um texto na tela, é adicionado ao *layout*. Já na linha 12, o texto deste elemento é alimentado com uma *string* definida no arquivo de *strings* do aplicativo.

4.1.4.2 Pasta *Drawable*

Nesta pasta são armazenadas as imagens utilizadas na aplicação. No Android as imagens são agrupadas em quatro tipos de acordo com sua densidade de resolução. Para cada resolução existe uma subpasta correspondente onde o arquivo deve ser adicionado. A Figura 4.2 demonstra a diferença entre as resoluções.

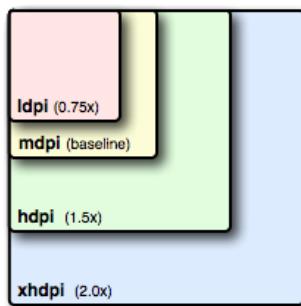


Figura 4.2: Tamanhos relativos a resolução suportada

A seguir, a Tabela 4.1 descreve as resoluções apresentadas na Figura 4.2:

Tipo	Resolução	Descrição
ldpi	36x36	Para baixa resolução
mdpi	48x48	Para média resolução
hdpi	72x72	Para alta resolução
xhdpi	96x96	Para extra alta densidade

Tabela 4.1: Resoluções de imagens.

4.1.4.3 Pasta *Values*

Este diretório contém os arquivos XML utilizados para a internacionalização de aplicativos. Nele deve estar contido um arquivo chamado *string.xml*. Neste arquivo são adicionadas todas as *strings* utilizadas pelo aplicativo. Estas são referenciadas no layout como é exemplificado na linha 12 do Código 4.3, visto anteriormente.

Caso o desenvolvedor queira disponibilizar seu aplicativo em outros idiomas, ele deverá criar um nova pasta chamada *values-xx*, onde **xx** corresponde a sigla deste idioma

definido pela ISO 639.2. Cada uma destas pastas deve ser adicionada diretamente a pasta res e deve conter um novo arquivo *string.xml*. Caso queira, por exemplo, disponibilizar o aplicativo em língua francesa, deve criar uma pasta chamada *values-fr* e o arquivo *string.xml* desta deve conter a tradução.

Desta forma, de acordo com a configuração de idioma do dispositivo, este escolherá automaticamente qual será utilizado. Caso não haja uma tradução que conte a este idioma, o aplicativo será instalado com a versão presente na pasta values.

O arquivo *string.xml* pode definir três tipos de elementos diferentes. São eles:

- ***String***

Recurso onde se define uma única *string*. O Código 4.4 apresenta um exemplo da utilização deste recurso.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="string_name"> text_string </string>
4 </resources>
```

Código 4.4: *String*

- ***String Array***

Recurso onde é definido um vetor de *strings* que pode ser referenciado na aplicação.

O Código 4.5 apresenta um exemplo deste recurso.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string-array name="string_array_name">
4         <item> text_string1 </item>
5         <item> text_string2 </item>
6         <item> text_string3 </item>
7     </string-array>
8 </resources>
```

Código 4.5: *String Array*

- ***Quantity Strings (Plurals)***

Este recurso é utilizado para definir como uma determinada *string* será apresentada de acordo com a quantidade de elementos que esta representar. O Android suporta as seguintes quantidades: *zero*, *one*, *two*, *few*, *many*, e *other*. O método *getQuantityString()* pode ser utilizado para retornar a *string* correspondente a uma quantidade de elementos informada. O Código 4.6 demonstra a utilização deste recurso.

```

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

```

4     <item quantity="one">%d carro</item>
5     <item quantity="other">%d carros</item>
6   </plurals>
7 </resources>

```

Código 4.6: *Plurals*

Todas as sequências são capazes de aplicar a marcação de estilo e formatação de argumentos.

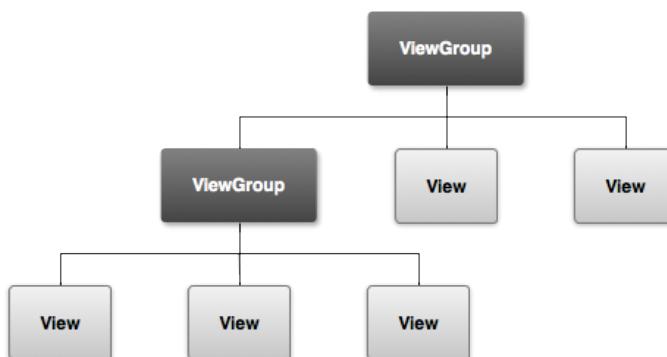
4.2 Desenvolvimento

Nas próximas seções são abordados os passos para implementação de um aplicativo para o Android. Inicialmente, será demonstrado como criar sua interface gráfica. Na seção seguinte, descrito como gerenciar os recursos da aplicação através das atividades. Em seguida, apresentado como utilizar o arquivo `AndroidManifest.xml`. Por fim, exemplificado como criar e manipular o banco de dados.

4.2.1 *Graphical User Interface (GUI)*

O *layout* é a interface entre o usuário e a aplicação. Por este motivo, este deve ser bem estruturado a fim de proporcionar ao utilizador uma experiência agradável com o aplicativo, de forma simples, clara e intuitiva.

No Android, uma interface gráfica é composta por elementos *View*, que representam objetos de interface, como campos de texto e botões. Estes elementos são dispostos em *containers*, chamados *ViewGroups*, objetivando a melhor organização destes no *layout*. A Figura 4.3 apresenta um exemplo genérico da hierarquia entre estes elementos.

Figura 4.3: Hierarquia da *View* com definição do *layout* de interface

Quanto mais complexa esta hierarquia for, menos eficiente será. Por conta disto, o desenvolvedor deve procurar criar interfaces simples objetivando a maior eficiência de sua aplicação.

4.2.1.1 Declaração do *layout* da aplicação

O Android oferece duas formas de se criar o *layout* da aplicação: por meio de XML e por meio de classes Java. Neste trabalho será abordada a criação do *layout* por meio XML, como foi explicado na seção *Layout* do Capítulo 2.

O XML é uma linguagem bastante intuitiva. Observando o Código 4.7, é possível constatar que não é necessário um vasto conhecimento sobre a linguagem para se construir interfaces gráficas no Android. Contudo, não é foco deste trabalho abordar seus aspectos básicos, devendo este conhecimento ser adquirido previamente.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:orientation="vertical" >
6     <TextView android:id="@+id/text"
7         android:layout_width="wrap_content"
8         android:layout_height="wrap_content"
9         android:text="Eu sou um TextView" />
10    <Button android:id="@+id/button"
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:text="Eu sou um Botao" />
14 </LinearLayout>
```

Código 4.7: *LinearLayout*

Cada elemento que compõe a interface gráfica é representado em XML por uma *tag*, como é o caso da *tag* *<Button>* presente na linha 10 do Código 4.7. Além disso, cada um destes elementos possuem atributos que devem ser preenchidos para configurá-los. Um exemplo disto podemos observar na linha 13 do Código 4.7, onde o elemento *Button* recebe uma *string* em seu atributo *android:text*, responsável por guardar o texto que será apresentado pelo elemento na interface.

4.2.1.1.1 Identificação de recurso .

Todo elemento que posteriormente precisar ser manipulado pelo aplicativo deve definir uma identificação única que será utilizada para referenciá-lo. Esta identificação deve constar no atributo *android:id*, como pode ser observado na linha 1 do Código 4.8. Após este ser preenchido, o elemento será adicionado automaticamente a lista de recursos do aplicativo, na classe R.

```

1 <Button android:id="@+id/button"
2     android:layout_width="wrap_content"
3     android:layout_height="wrap_content"
4     android:text="Eu sou um Botao" />
```

Código 4.8: Botão com alguns de seus atributos básicos.

A identificação de um elemento pode ser feita de duas formas distintas: criando-se uma nova identificação ou identificando-o como um recurso do Android. A seguir estes formas são descritas:

- Criando uma nova identificação para o recurso

Utilizando o Código 4.9 é possível criar uma nova identificação para um elemento adicionado a interface. O símbolo '@' sinaliza que este texto é uma instrução que deverá ser processada. Já o símbolo '+' indica que este identifica um novo tipo de recurso adicionado a Classe R, como pode ser observado no Código 4.10.

```
1 android:id="@+id/botao"
```

Código 4.9: Nova identificação de recurso

```
1 public static final class id {
2     public static final int botao=0x7f070000;
3 }
```

Código 4.10: Recurso identificado na Classe R

- Identificando como um recurso do Android

O Android oferece identificações padronizadas aos recursos. O Código 4.11 exemplifica a atribuição deste tipo de identificação, neste caso uma lista (*list*). Ao contrário da forma anterior, o símbolo '+' não é necessário, já que não é criada uma nova referência a um recurso. Porém deve-se informar que um recurso Android do tipo *id* será acessado.

```
1 android:id="@+id/list"
```

Código 4.11: Identificação como um recurso Android

4.2.1.1.2 Dimensionalidade de elementos .

Ao se adicionar um elemento a interface é necessário estipular a largura e altura que este irá ocupar. Esta definição pode ser efetuada através dos atributos *android:layout_width* e *android:layout_height*, responsáveis por definir a largura e altura, respectivamente. Para esta definição podem ser utilizadas constantes pré-definidas ou valores específicos. A seguir serão descritas as constantes disponíveis:

- *fill_parent* - Define que o elemento deve ocupar toda área disponível na tela, utilizado até a API 8;
- *match_parent* - Define que o elemento deve ocupar toda área disponível na tela, utilizado após a API 8;

- *wrap_content* - Define que o elemento deve ocupar apenas a área necessária na tela.

Como foi dito anteriormente, também é possível definir um valor que especifique o tamanho deste elemento. A Tabela 4.2 lista as unidades dimensionais conhecidas pelo Android para este procedimento.

Unidade	Representação
Densidade de <i>Pixel</i> Independente	dip, dp, dps
Escala de <i>Pixel</i> Independente	sip, sp
Milímetro	mm
<i>Pixel</i>	px
Polegada	in
Ponto	pt

Tabela 4.2: Unidades dimensionais aceitas pelo Android.

A utilização de unidades absolutas, como o *pixel*, não é recomendada, visto que estas não se adaptam as diferentes resoluções de tela. Em vez disso deve-se utilizar medidas relativas, como a Densidade de *Pixel* Independente, ou constantes, para garantir a exibição correta do *layout* independente do tamanho da tela.

4.2.1.1.3 Elementos do *layout*

O Android dispõe de vários elementos gráficos que podem ser utilizados pelo desenvolvedor a fim de criar suas interfaces, como campos de texto e listas. Nesta seção, serão descritos alguns dos elementos mais utilizados e como adicioná-los a uma interface. A Figura 4.4 apresenta os elementos que descritos a seguir.

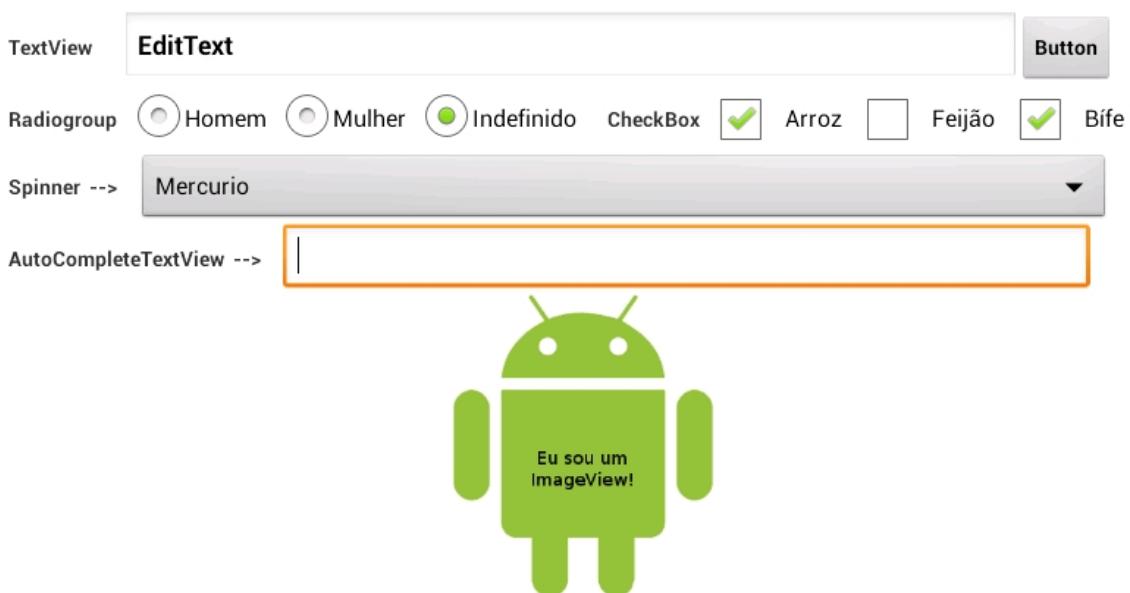


Figura 4.4: Elementos gráficos

TextView

Um elemento *TextView* representa um texto na tela, como a classe *JLabel* presente no pacote *Swing* do Java. No Código 4.12 é possível observar como adicioná-lo ao *layout*.

```

1 <TextView
2     android:layout_width = "wrap_parent"
3     android:layout_height = "wrap_parent"
4     android:text = "Este que aparece na tela."/>

```

Código 4.12: Criação de um elemento *TextView*

EditText

O elemento *EditText* cria uma caixa de texto na tela para inserção de dados pelo usuário. Um exemplo básico é apresentado no Código 4.13.

```

1 <EditText
2     android:layout_width = "wrap_parent"
3     android:layout_height = "wrap_parent"
4     android:text = ""/>

```

Código 4.13: Criação de um elemento *EditText*

Button

O Código 4.14 representa um botão adicionado a interface.

```

1 <Button
2     android:layout_width="wrap_content"
3     android:layout_height="wrap_content"
4     android:text="Texto do Botao" />

```

Código 4.14: Criação de um elemento *Button*

RadioGroup

Um elemento *RadioGroup* representa uma lista de opções onde apenas uma destas poderá ser selecionada. Cada uma destas opções é representada por um elemento *RadioButton* adicionado a estrutura do *RadioGroup*. O Código 4.15 demonstra a criação de uma *RadioGroup*.

```

1 <RadioGroup android:id="@+id/radiogroup"
2     android:layout_width="fill_parent"
3     android:layout_height="wrap_content"
4     android:orientation="vertical" />
5
6     <RadioButton android:id="@+id/radio_masc"
7         android:layout_width="wrap_content"
8         android:layout_height="wrap_content"
9         android:text="Masculino" />
10
11    <RadioButton android:id="@+id/radio_fem"
12        android:layout_width="wrap_content"
13        android:layout_height="wrap_content"

```

```

14     android:text="Feminino" />
15 </RadioGroup>
```

Código 4.15: Criação de um *RadioGroup*

CheckBox

O *Checkbox* cria uma opção que pode ser marcada ou desmarcada pelo usuário. É possível adicioná-lo ao *layout* conforme o Código 4.16.

```

1 <CheckBox
2     android:id="@+id/checkbox"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:text="Clique para receber e-mails" />
```

Código 4.16: Criação de um *CheckBox*

Spinner

Um *Spinner* é um elemento que, caso seja selecionado, apresenta uma lista de opções onde apenas uma poderá ser escolhida. Este é parecido a classe *JComboBox* do pacote *Swing* do Java. Um exemplo da implementação deste elemento pode ser observado no Código 4.17, onde é criada uma lista de planetas para seleção.

```

1 <Spinner
2     android:id="@+id/spinner"
3     android:layout_width="fill_parent"
4     android:layout_height="wrap_content"
5     android:prompt="@string/planetas" />
```

Código 4.17: *Spinner*

Na linha 5 do Código 4.17 o *Spinner* é carregado com um *array* de *strings* presentes no arquivo *strings.xml*. O conteúdo deste *array* pode ser observado no Código 4.18.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="planetas">Escolha um planeta</string>
4     <string-array name="planetas_array">
5         <item>Mercurio</item>
6         <item>Venus</item>
7         <item>Terra</item>
8         <item>Marte</item>
9         <item>Jupiter</item>
10        <item>Saturno</item>
11        <item>Urano</item>
12        <item>Netuno</item>
13    </string-array>
14 </resources>
```

Código 4.18: *Array* de *Strings* adicionado ao *Spinner*

AutoCompleteTextView

O elemento *AutoCompleteTextView* possui aparência de um elemento *EditText*, mas esse pode oferecer ao usuário uma lista com opções de acordo com o que for digitado. O Código 4.19 apresenta como adicionar este elemento ao *layout*.

```

1 <AutoCompleteTextView
2     android:id="@+id/autocomplete"
3     android:layout_width="fill_parent"
4     android:layout_height="wrap_content"/>
```

Código 4.19: Criação de um *AutoCompleteTextView*

Na seção *Activity* é descrito como adicionar uma lista de opções a este elemento.

ImageView

Um elemento do tipo *ImageView* é responsável por adicionar figuras a interface. Um exemplo de sua utilização pode ser observado no Código 4.20, onde uma imagem presente na pasta *drawable* é referenciada.

```

1 <ImageView
2     android:layout_width="wrap_content"
3     android:layout_height="wrap_content"
4     android:src="@drawable/imagem"/>
```

Código 4.20: Criação de um elemento *ImageView*

4.2.2 Atividades (*Activity*)

A atividade é o componente responsável por gerir o *layout* da aplicação. Sendo assim, cada *layout* terá uma atividade que deverá implementar suas funções. Dentre as funções pertinentes a uma atividade, podemos destacar: setar o *layout* da aplicação, gerenciar a interação entre o usuário e o *layout*, processar dados, dentre outras.

Quando um aplicativo é iniciado, este acessa o arquivo *AndroidManifest.xml* buscando descobrir qual é a atividade principal, ou seja, a que deve ser chamada quando o aplicativo estiver em inicialização. Após ser chamada, esta monta o *layout* para a utilização do usuário. Durante o uso da aplicação, uma atividade pode chamar outra a fim de abrir uma nova interface. Neste processo é utilizado o sistema de pilha, onde uma nova atividade é colocada em um plano acima da atividade que a chamou. Quando a atividade em execução for fechada o aplicativo retornará a anterior.

Na próxima seção será descrito o ciclo de vida de uma atividade. Em seguida será descrito como utilizar os recursos do aplicativo e como as atividades interagem durante a execução. Por fim, será exemplificada a utilização de alguns dos principais componentes da

interface pela atividade. Outras informações podem ser encontradas na seção 'Atividades' do Capítulo 2.

4.2.2.1 Implementação do Ciclo de Vida da Atividade

Durante a sua execução, uma atividade está suscetível a inúmeras mudanças de estado que devem ser previstas e tratadas pelo desenvolvedor. Se, por exemplo, durante a execução de um aplicativo o usuário receber uma ligação a atividade em execução deverá ser paralizada e, antes disso, esta deve garantir a integridade dos dados que não estão salvos. Para este fim, uma atividade deve implementar seu ciclo de vida, onde é definido os procedimentos necessários adotados por esta em uma mudança de estado.

A Figura 4.5 apresenta o ciclo de vida de uma atividade, de sua criação até sua destruição. Cada um dos estados deste ciclo de vida é representado por um retângulo que contém seu nome. Com exceção do estado de criação, *onCreate*, a implementação dos estados é facultativa, ou seja, se o aplicativo não necessitar efetuar alguma função em um determinado estado, este não necessitará ser implementado.

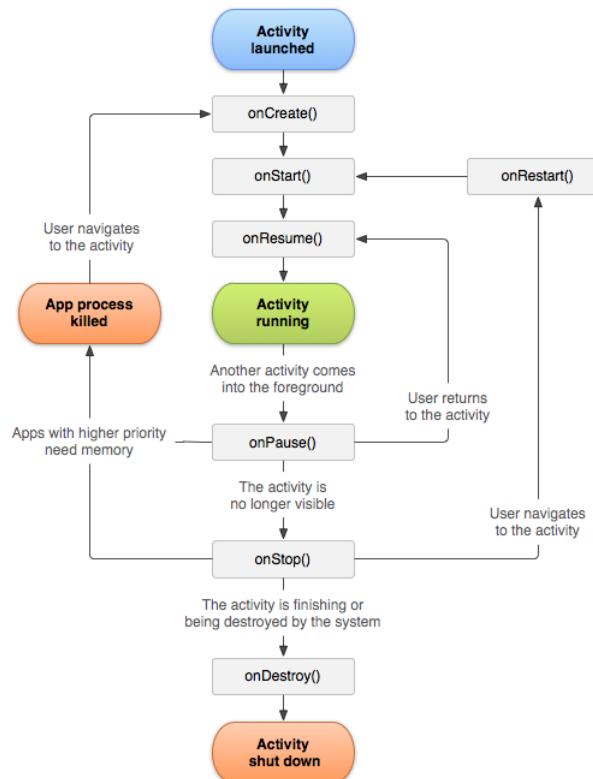


Figura 4.5: Ciclo de vida de uma atividade (ANDROID, 2012d)

Cada um dos estados do ciclo de vida deve ser implementado na atividade como um método, com mesmo nome. O Código 4.21 demonstra a criação de uma atividade que implementa a estrutura básica para criação destes métodos.

```

1  public class exemploCicloDeVidaActivity extends Activity {
2      @Override
3      public void onCreate(Bundle savedInstanceState) {
4          super.onCreate(savedInstanceState);
5          // A atividade é criada.
6      }
7      @Override
8      protected void onStart() {
9          super.onStart();
10         // A atividade está prestes a tornar-se visível.
11     }
12     @Override
13     protected void onResume() {
14         super.onResume();
15         // A atividade torna-se visível ou é "retomada".
16     }
17     @Override
18     protected void onPause() {
19         super.onPause();
20         // Outra atividade está ganhando foco ou está e "pausada".
21     }
22     @Override
23     protected void onStop() {
24         super.onStop();
25         // A atividade não é mais visível.
26     }
27     @Override
28     protected void onDestroy() {
29         super.onDestroy();
30         // A atividade está prestes a ser destruída.
31     }
32 }
```

Código 4.21: Código básico para implementação do ciclo de vida da atividade

4.2.2.2 Utilização de recursos

Por diversos motivos uma aplicação pode necessitar utilizar alguns dos recursos disponíveis no aplicativo. Como já foi dito anteriormente, todos os recursos da aplicação estão referenciados na Classe R. Desta forma, para referenciar um destes recursos na atividade deve-se acessar essa informação.

Utilizando como base os exemplos presentes nos Códigos 4.9 e 4.10, o Código 4.22 apresenta o comando utilizado para acessar o recurso denominado botao listado na Classe R.

¹ R.id.botao

Código 4.22: Exemplo de acesso a recurso da Classe R.

Para poder manipular os recursos do aplicativo dentro da atividade, o desenvolvedor deve criar um *link* entre este recurso e um objeto Java, que será utilizado para

manipulá-lo em tempo de execução. Tendo como base o exemplo anterior, o Código 4.23 demonstra como este *link* pode ser feito. Na linha 3, um objeto do tipo *Button* é criado. Já na linha 9, o método *findViewById* recebe como parâmetro a referência do recurso "botao". Este método procura um elemento *view* com esta identificação e seu retorno é convertido em um tipo *button* que, por fim, é passado ao objeto criado na linha 3. Desta forma, qualquer alteração feita em um influenciará o outro.

```

1  public class exemploLinkRecursoActivity extends Activity {
2
3      private Button btnBotao;
4
5      @Override
6      protected void onCreate(Bundle savedInstanceState) {
7          super.onCreate(savedInstanceState);
8
9          btnBotao = (Button) findViewById(R.id.botao);
10     }
11 }
```

Código 4.23: Criação de link entre recurso e objeto da atividade.

É importante ressaltar que o ato de se criar esta ligação entre os recursos e objetos da atividade deve ser implementado, impreterivelmente, no método *onCreate*.

4.2.2.3 Interação entre atividades

Um aplicativo é composto por inúmeras interfaces que interagem entre si. No Android, a atividade é responsável por gerir esta interação, solicitando a abertura de outras atividades e recuperando dados destas.

Para que estas interfaces interajam entre si é necessário um instrumento capaz de mediar a troca de informações. Para este propósito é utilizada a intenção (*intent*). Esta é responsável por enviar dados e requisições de atividades, recuperar uma intenção enviada por outra atividade e acessar seus dados.

O Código 4.24 demonstra como é feita uma solicitação de abertura de um nova atividade. Na linha 1 é criado e instanciado um objeto do tipo *intent*. Este recebe como parâmetro o nome da classe que define a atividade responsável pela interface que se deseja chamar. Na linha 2, o método *putExtra*, responsável por carregar a intenção com dados, recebe dois parâmetros: o primeiro é o nome que será posteriormente utilizado para recuperar a informação e o segundo é a informação propriamente dita, que pode ser de qualquer tipo de dado Java (int, float, String, boolean, etc.). Por fim, na linha 3 o método *startActivity* solicitará que a atividade definida no objeto *intent* seja aberta pelo aplicativo.

```

1 Intent intencao = new Intent(this, NovaAtividade.class);
2 intencao.putExtra("nome_de_identificacao", valor);
3 startActivity(intencao);

```

Código 4.24: Abertura de uma nova atividade.

O Código 4.25 exemplifica como a nova atividade pode lidar com os dados recebidos. O método *getIntent*, na linha 1, recupera a intenção enviada por outra atividade. Na linha 2 o método *getStringExtra* é utilizado a fim de recuperar na intenção o valor referente ao nome informado. Já na linha 4 é exemplificado como fechar uma atividade em execução, retornando a atividade anterior.

```

1 Intent intencao = getIntent();
2 valor = intencao.getStringExtra("nome_de_identificacao");
3
4 finish();

```

Código 4.25: Acesso a dados de uma intenção e fechamento de atividade.

4.2.2.4 Utilização de elementos do *layout*

Na seção 'Elementos do Layout' foi exemplificado como adicionar alguns dos principais elementos de interface utilizado em aplicativos Android. Nesta seção será demonstrado como utilizar algumas funções destes elementos pela atividade. Para os exemplos a seguir, considere que os objetos que representam cada elemento da interface estão vinculados a interface, conforme pode ser observado no Código 4.23.

TextView

O Código 4.26 demonstra como é possível acessar o texto presente em um *TextView*, linha 1, e como modificar este texto, linha 2.

```

1 String texto = textView.getText();
2 textView.setText("Outro texto!");

```

Código 4.26: Acesso e edição de texto de um elemento *TextView*

EditText

O acesso e edição de textos do elemento *EditText* é semelhante ao elemento *TextView*, como pode ser observado no Código 4.27.

```

1 String texto = editText.getText();
2 editText.setText("Outro texto!");

```

Código 4.27: Acesso e edição de texto de um elemento *EditText*

Button

O Código 4.28 exemplifica como deve ser criada a estrutura onde deve ser implementada a ação de um botão quando este é clicado. Esta estrutura deve ser, obrigatoriamente, implementada durante a criação da atividade, ou seja, no estado *onCreate*.

```

1 public void onCreate(Bundle savedInstanceState) {
2     ...
3     btnAdicionarProdutoLista = (Button) findViewById(R. lista . btnAdicionar);
4
5     btnAdicionarProdutoLista.setOnClickListener(new View.OnClickListener() {
6         public void onClick(View v) {
7             //Implementar aqui a ação do botão.
8         }
9     });
10 }
```

Código 4.28: Evento *OnClick* de um botão

RadioGroup

Usando como base o Código 4.15, o Código 4.29 demonstra como é possível descobrir qual *RadioButton* está marcado.

```

1 RadioGroup radioGroup = (RadioGroup) findViewById(R. id . radiogroup);
2
3 int checkedRadioButton = radioGroup.getCheckedRadioButtonId();
4 String radioButtonSelected = "";
5
6 switch (checkedRadioButton) {
7     case R. id . radio_masc : radioButtonSelected = "Masculino";
8         break;
9     case R. id . radio_fem : radioButtonSelected = "Feminino";
10        break;
11 }
```

Código 4.29: Descobrir qual *RadioButton* em um *RadioGroup* está marcado.

CheckBox

O Código 4.30 exemplifica como descobrir se um *Checkbox* está marcado.

```
1 Boolean estaMarcado = checkBox.isChecked();
```

Código 4.30: Descobrir se um *CheckBox* está marcado

Spinner

O exemplo demonstrado no Código 4.31 demonstra como é possível adicionar uma lista de opções a um *spinner* em tempo de execução. Nas linhas 1 e 2 são criados a lista de opções que serão adicionadas ao *spinner* e o objeto que representará este elemento, respectivamente. Na linha 4, é criado um objeto do tipo *ArrayAdapter*, responsável por armazenar os dados em um formato que poderá ser reconhecido pelo *spinner*. Este objeto é adicionado ao *spinner* na linha 7.

```

1 String [] items = new String [] {"Opção1", "Opção2", "Opção3"};
2 Spinner spinner = (Spinner) findViewById(R.id.meuSpinner);
3
4 ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
5         android.R.layout.simple_spinner_item, items);
6
7 spinner.setAdapter(adapter);

```

Código 4.31: Como adicionar uma lista a um *Spinner* em tempo de execução.

O Código 4.32 demonstra como adicionar o *listener* responsável por efetuar uma ação caso um elemento do *spinner* seja selecionado ou caso não haja nada selecionado. O objeto **pos**, recebido como parâmetro do método *onItemSelected*, guarda a posição do valor selecionado para que este possa ser acessado, conforme exemplificado na linha 5.

```

1 spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
2     @Override
3     public void onItemSelected(AdapterView<?> arg0, View arg1, int pos, long id) {
4         //Implementar aqui processo apos a selecao de uma opção.
5         String opção = items[pos];
6     }
7     @Override
8     public void onNothingSelected(AdapterView<?> arg0) {
9         //Implementar aqui processo caso nao haja opção selecionada.
10    }
11 });

```

Código 4.32: Como adicionar uma lista a um *Spinner* em tempo de execução.

AutoCompleteTextView

Para adicionar uma lista de opções a um elemento *AutoCompleteTextView* o procedimento é similar ao demonstrado no Código 4.31. O Código 4.27 pode ser utilizado como referência de como acessar o texto deste elemento.

ImageView

O código 4.33 demonstra como definir dinamicamente uma imagem, que faz parte do conjunto de recursos do aplicativo, em tempo de execução.

```

1 imagem.setImageResource(R.drawable.nome_da_imagem);

```

Código 4.33: Setar uma imagem em um elemento *ImageView*

4.2.3 *Android Manifest*

Este arquivo contém o registro da atividade e possui todas as informações necessárias para o funcionamento da aplicação. É gerado automaticamente quando o projeto é criado, mas deve ser editado ao decorrer do projeto para o ideal funcionamento da aplicação.

Como exemplo, o Código 4.34 apresenta uma versão simplificada do arquivo *AndroidManifest.xml* do aplicativo implementado neste projeto.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="iff.tcc.obrafacil"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
7     <application
8         android:icon="@drawable/ic_launcher"
9         android:label="@string/app_name" >
10        <activity
11            android:label="@string/app_name"
12            android:name=".PrincipalObraFacilActivity" >
13            <intent-filter >
14                <action android:name="android.intent.action.MAIN" />
15                <category android:name="android.intent.category.LAUNCHER" />
16            </intent-filter >
17        </activity >
18        <activity android:name=".CategoriaCadastroActivity"
19                 android:label="@string/app_name">
20            </activity >
21        </application >
22
23 </manifest>
```

Código 4.34: Parte do Manifest do projeto obrafacil

Na linha 3 do Código 4.34 é definido o nome do pacote principal do aplicativo. Para o controle de atualizações do aplicativo é necessário definir os campos *android:versionCode* (linha 4) e *android:versionName* (linha 5). O primeiro deve ser incrementado em um a cada nova versão e o segundo define a numeração correspondente a versão do aplicativo visível ao usuário.

É importante destacar no Código 4.34 o conteúdo presente entre as linhas 7 e 21, onde são definidas quais atividades poderão ser utilizadas pelo aplicativo durante sua execução. A tag *<application>* guardará informações gerais da aplicação, como o ícone (linha 8) e seu nome (linha 9). Dentro desta tag é possível adicionar quais as atividades poderão ser utilizadas na aplicação, como é possível observar entre as linhas 10 e 20. A atividade principal deverá implementar o código presente entre as linhas 13 e 16. Por fim, o atributo *android:nome* define o nome da classe que implementa a atividade, presente no pacote da aplicação, como pode ser observado nas linhas 12 e 18.

4.2.3.1 Elementos e atributos

Para uma aplicação poder ser iniciada é imprescindível que as tags *<manifest>* e *<application>* sejam utilizadas dentro do *AndroidManifest.xml* e estas também não devem ser repetidas. A maioria dos outros elementos podem ocorrer muitas vezes ou

não, embora pelo menos alguns destes devam estar presentes no manifesto para que seja realizado algo significativo pela aplicação.

Elementos de mesmo nível geralmente não são ordenados. Por exemplo, `<activity>`, `<provider>` e `<service>` podem ser arrumados em qualquer sequência. Um elemento `<activity-alias>` é a exceção a esta regra. Este só deve ser chamado após um elemento `<activity>`.

Todos os elementos que podem aparecer neste arquivo estão presentes na Tabela 4.3. Estes são os únicos elementos permitidos, ou seja, não é possível adicionar novos elementos ou atributos.

<code><action></code>	<code><activity></code>	<code><activity-alias></code>
<code><application></code>	<code><category></code>	<code><data></code>
<code><grant-uri-permission></code>	<code><instrumentation></code>	<code><intent-filter></code>
<code><manifest></code>	<code><meta-data></code>	<code><permission></code>
<code><permission-group></code>	<code><permission-tree></code>	<code><provider></code>
<code><receiver></code>	<code><service></code>	<code><supports-screens></code>
<code><uses-configuration></code>	<code><uses-feature></code>	<code><uses-library></code>
<code><uses-permission></code>	<code><uses-sdk></code>	

Tabela 4.3: Tags presentes no *AndroidManifest.xml*

Os atributos são usados para prover informação adicional sobre os elementos. Geralmente todos os atributos são opcionais. Entretanto, existem alguns que devem ser especificados por um elemento para que este cumpra com a sua finalidade. Os atributos recebem um valor padrão caso não sejam mencionados.

Exceto por alguns atributos do elemento raiz `<manifest>`, todos os nomes dos atributos começam com o prefixo 'android:' como, por exemplo, `android:icon`. Como o prefixo é universal, na documentação este é geralmente omitido referindo-se ao atributo somente pelo nome, neste caso *icon*.

4.2.3.2 Declarando os nomes das classes

Muitas das classes Java correspondem a algum elemento presente no *AndroidManifest.xml*, como os elementos aplicação (`<application>`), atividades (`<activity>`), serviços (`<services>`), *broadcast receiver* (`<receiver>`) e provedor de conteúdo (`<provider>`).

Ao definir uma classe como uma subclasse, esta deve declarar um nome de atributo. Este nome deve ser acompanhado do pacote onde a subclasse pertence. Como exemplo, podemos criar uma subclasse de serviço conforme o Código 4.35.

```

1 <manifest . . . >
2   <application . . . >
3     <service android:name="com.example.project.SecretService" . . . >
4     . .
5   </service>
6   .
7 </application>
8 </manifest>

```

Código 4.35: Criando uma subclasse com designação de pacote.

Podemos reescrever o Código 4.35 especificando o pacote padrão onde as demais subclasses serão criadas, e informar a subclasse apenas o seu nome. Isto evita uma repetição desnecessária de referências ao pacote e, por consequência, poupa-se tempo. O código 4.36 possui resultado igual ao Código 4.35, porém especifica um pacote padrão.

```

1 <manifest package="com.example.project" . . . >
2   <application . . . >
3     <service android:name=".SecretService" . . . >
4     . .
5   </service>
6   .
7 </application>
8 </manifest>

```

Código 4.36: Criando uma subclasse sem designação de pacote.

Quando um componente é iniciado pelo Android, para cada subclasse chamada uma instância é criada. Caso não seja especificada a subclasse, o Android criará uma instância da classe base.

4.2.3.3 Múltiplos valores

Para evitar que um elemento seja repetido inúmeras vezes, por conter mais de um valor especificado, podemos listar estes valores dentro de um único elemento. É possível exemplificar isto através do Código 4.37, onde várias ações (*action*) são listadas dentro de um filtro de intenção (*Intent Filter*).

```

1 <intent-filter . . . >
2   <action android:name="android.intent.action.EDIT" />
3   <action android:name="android.intent.action.INSERT" />
4   <action android:name="android.intent.action.DELETE" />
5   .
6 </intent-filter>

```

Código 4.37: Definição de múltiplas ações em um filtro de intenção.

4.2.3.4 Permissões

A permissão tem como principal objetivo restringir o acesso a uma parte do código ou a dados no dispositivo, protegendo estes de uma manipulação não autorizada. Cada permissão possui um identificador único e na maioria das vezes este pode indicar a ação restrita por esta permissão. O android possui várias permissões padrão. Algumas delas podem ser vistas no Código 4.38:

```

1 android.permission.CALL_EMERGENCY_NUMBERS
2 android.permission.READ_OWNER_DATA
3 android.permission.SET_WALLPAPER
4 android.permission.DEVICE_POWER

```

Código 4.38: Algumas das permissões padrão Android.

Para um aplicativo poder ter acesso a um recurso protegido, este deve declarar que necessita da permissão utilizando o elemento `<uses-permission>` no *AndroidManifest.xml*. Ao instalar um aplicativo, o seu instalador informa ao usuário quais permissões este precisará ter para funcionar. O usuário poderá conceder ou não tal permissão. Em caso negativo, o aplicativo simplesmente não conseguirá acessar tais recursos e nesses casos o usuário não receberá nenhum tipo de notificação.

Pode-se criar novas permissões para que um aplicativo possa proteger seus próprios componentes (atividades, serviços, receptores de broadcast e provedores de conteúdo) ou utilizar a declarada por outros aplicativos. Para declarar uma nova permissão é utilizado o elemento `<permission>`. Um exemplo de como é possível proteger uma atividade com o uso deste elemento pode ser observada no Código 4.39:

```

1 <manifest . . . >
2     <permission android:name="com.example.project.COMPRAR" . . . />
3     <uses-permission android:name="com.example.project.COMPRAR" />
4     .
5     <application . . . >
6         <activity android:name="com.example.project.ComprasActivity"
7             android:permission="com.example.project.COMPRAR"
8             . . . >
9             .
10            </activity>
11        </application>
12    </manifest>

```

Código 4.39: Utilizando permissões em uma atividade.

No exemplo anterior é possível observar que além de criar uma nova permissão utilizando o elemento `<permission>`, a aplicação precisa solicitar a utilização desta através do elemento `<uses-permission>`, mesmo sendo esta a responsável pela criação da permissão.

4.2.4 Armazenamento de Dados

O Android oferece várias opções para persistência de dados. A escolha de qual utilizar depende das necessidades específicas da aplicação e quanto de memória é requerida. As opções são as seguintes: (ANDROID, 2012f)

- Preferências Compartilhadas (*Shared Preferences*)

A classe *Shared Preferences* é utilizada para armazenar dados primitivos, em pares de chave-valor, sempre que a *activity* for fechada. Isto possibilita a recuperação destes dados após esta ser reiniciada.

- Armazenamento interno

Armazena dados em arquivo diretamente na memória interna do dispositivo, que só pode ser acessado pelo aplicativo que o criou. Este arquivo é removido com a desinstalação do aplicativo.

- Armazenamento externo

Armazena dados públicos em mídia móvel (como em cartão SD) ou na própria memória interna. Por estes dados estarem disponíveis, deve-se tomar cuidado com este tipo de armazenamento, já que os dados em questão podem ser apagados. Além disso, caso estejam armazenados em uma mídia móvel, estes podem ficar indisponíveis após esta ser removida.

- Banco de dados SQLite

Até o presente momento, o Android oferece total suporte apenas a bancos de dados SQLite, armazenando dados estruturados em um banco privado.

- Conexão de rede

O Android também permite armazenar e recuperar dados em servidores *web*, desde que haja conexão com a rede, utilizando os pacotes *Java.net* e *Android.net*.

Dentre as opções listadas anteriormente, será utilizado neste trabalho o banco de dados SQLite, um banco relacional que utiliza a sintaxe do SQL.

A Figura 4.6 apresenta o diagrama de classe que representa uma visão geral da interação entre as classes fundamentais para o funcionamento do banco de dados criado. Estas classes são implementadas seguindo os conceitos de orientação objeto, onde há um escopo bem definido e a estrutura possibilita sua reutilização pelo sistema.

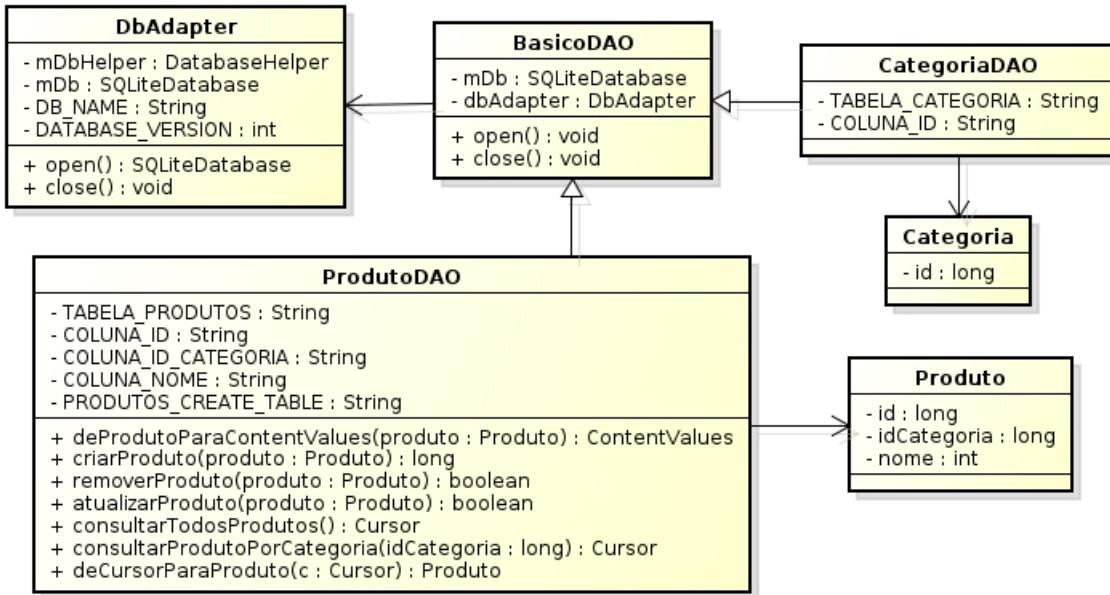


Figura 4.6: Diagrama de classe do banco de dados

Utilizando o diagrama da Figura 4.6 como base, neste trabalho será demonstrado como criar um banco contendo uma tabela de produtos. A seguir, as classes criadas serão apresentadas:

- *DbAdapter* - Classe responsável por garantir o acesso ao banco e por criar, atualizar e remover tabelas deste.
- *BasicoDAO* - Classe responsável por adquirir uma instância de um SQLiteDatabase a partir da classe *DbAdapter*. Esta instância será utilizada por suas classes filhas para acessar o banco.
- *ProdutoDAO* - Classe filha de *BasicoDAO* responsável por definir métodos que criam e manipularão dados da tabela de produtos.
- *Produto* - Classe que define o escopo de um objeto *Produto*.

Por padrão, as classes responsáveis por lidar com uma determinada tabela recebem o nome da tabela seguido de DAO. Por exemplo, o nome da classe responsável por lidar com a tabela de categorias é chamada de *CategoriaDAO*. Para manter o encapsulamento do sistema, caso seja necessário lidar com mais de uma tabela por vez, deve-se criar uma classe para este fim. Esta deve receber o nome das tabelas envolvidas seguido de DAO, como *CategoriaProdutoDAO*.

O Código 4.40 apresenta a classe que define o modelo de um *Produto*.

```

1 package iff.tcc.obrafacil.model;
2
3 public class Produto {
4     private Long id;
5     private String nome;
6     private Long idCategoria;
7
8     //Getter e Setters
9 }
```

Código 4.40: Classe Produto

Nas próximas seções serão descritas as classes *DbAdapter*, *BasicoDAO* e *ProdutoDAO*.

4.2.4.1 Classe *DbAdapter*

O trecho de Código 4.41 a seguir apresenta a declaração das variáveis utilizadas nesta classe. Na linha 5, podemos destacar a *string* que armazenará o nome do banco de dados. Já na linha 6 é definida a versão do banco de dados.

```

1 public class DbAdapter {
2     private static final String TAG = "DbAdapter";
3     private DatabaseHelper mDbHelper;
4     private SQLiteDatabase mDb;
5     private static final String DB_NAME = "NOME_DO_BANCO";
6     private static final int DATABASE_VERSION = 1;
7     private final Context mCtx;
```

Código 4.41: Variáveis da classe *DbAdapter*

Na linha 3 do Código 4.41 é criado um objeto do tipo *DatabaseHelper*. Esta é uma classe privada que será criada dentro da classe *DbAdapter* que manipulará o banco.

A classe *DatabaseHelper* deve implementar os seguintes métodos: *onOpen*, *onCreate* e *onUpdate*. A criação desta classe será dividida em três partes, vistas a seguir.

O Código 4.42 apresenta a primeira parte da criação da classe *DatabaseHelper*. O método *onOpen* é definido. Este abre o banco e habilita a utilização de chaves estrangeiras, que não são suportadas em versões anteriores a 2.1. Em seguida seu construtor é sobrescrito, recebendo o nome do banco e sua versão.

```

1     private static class DatabaseHelper extends SQLiteOpenHelper {
2         @Override
3         public void onOpen(SQLiteDatabase db) {
4             super.onOpen(db);
5             if (!db.isReadOnly()) {
6                 db.execSQL("PRAGMA foreign_keys=ON;");
7             }
8         }
9         DatabaseHelper(Context context) {
```

```

10         super(context, DB_NAME, null, DATABASE_VERSION);
11     }

```

Código 4.42: Classe privada *DatabaseHelper* - Parte 1

No Código 4.43 é definido o método *onCreate*, que cria as tabelas do banco na primeira vez que uma conexão com este for aberta. Na linha 3, o método *execSQL* recebe como parâmetro uma *string*, presente na classe *ProdutoDAO*, que guarda o *script* de criação do banco. Este será visto posteriormente.

```

1     @Override
2     public void onCreate(SQLiteDatabase db) {
3         db.execSQL(ProdutoDAO.PRODUTOS_CREATE_TABLE);
4     }

```

Código 4.43: Classe privada *DatabaseHelper* - Parte 2

O Código 4.44 demonstra uma atualização das tabelas do banco. Neste caso, o método simplesmente apaga as tabelas do banco e solicita ao método *onCreate* que as recrie. Porém, é possível utilizar dos parâmetros *oldVersion* e *newVersion* para basear possíveis alterações do banco.

```

1     @Override
2     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
3         db.execSQL("DROP TABLE IF EXISTS " + ProdutoDAO.TABELA_PRODUTOS);
4         onCreate(db);
5     }
6 }

```

Código 4.44: Classe privada *DatabaseHelper* - Parte 3

A classe *DbAdapter* deve definir seu construtor, que receberá um objeto do tipo *context*, e os métodos *open* e *close*, responsáveis por abrir e fechar o banco. O Código 4.45 apresenta a implementação deste construtor e destes métodos.

```

1     public DbAdapter(Context ctx) {
2         this.mCtx = ctx;
3     }
4     public SQLiteDatabase open() throws SQLException {
5         mDbHelper = new DatabaseHelper(mCtx);
6         mDb = mDbHelper.getWritableDatabase();
7         return mDb;
8     }
9     public void close() {
10        mDbHelper.close();
11        mDb.close();
12    }

```

Código 4.45: Métodos *open* e *close*

4.2.4.2 Classe *BasicoDAO*

O Código 4.46 mostra a criação da classe *BasicoDAO* que conterá os métodos que serão utilizados por todas as classes do tipo DAO para gerenciar o acesso direto ao banco.

```

1 package iff.tcc.obrafacil.dao;
2
3 import android.content.Context;
4 import android.database.sqlite.SQLiteDatabase;
5 public class BasicoDAO {
6     protected static SQLiteDatabase mDb;
7     protected static Context context;
8     private static DbAdapter dbAdapter;
9     private static int contador;
10
11    public BasicoDAO(Context ctx) {
12        context = ctx;
13    }
14    private synchronized static int numeroConexoes(int i) {
15        contador = contador + i;
16        return contador;
17    }
18
19    public synchronized void open() {
20        if (mDb == null || (mDb != null && !mDb.isOpen())) {
21            dbAdapter = new DbAdapter(context);
22            mDb = dbAdapter.open();
23        }
24        numeroConexoes(+1);
25    }
26
27    public synchronized void close() {
28        if (mDb != null && mDb.isOpen() && (numeroConexoes(0) == 1)) {
29            dbAdapter.close();
30            mDb.close();
31        }
32        numeroConexoes(-1);
33    }
34 }
```

Código 4.46: Classe BasicoDAO

Os métodos de *BasicoDAO* servem para adquirir uma instância de um *SQLiteDatabase* a partir do *DbAdapter*. Esta instância será utilizada pelas classes filhas de *BasicoDAO* para abrir e fechar a conexão com o banco.

É importante ressaltar que sempre que uma conexão com o banco for aberta, através do método *open*, esta deverá ser fechada assim que não for mais necessária, através do método *close*, para evitar uso desnecessário de recursos do sistema.

4.2.4.3 Classe *ProdutoDAO*

O Código 4.47 apresenta o início da implementação da classe *ProdutoDAO*. Neste são definidas constantes responsáveis por armazenar o nome da tabela e de seus atributos. Além disso, também é definido seu construtor.

```

1 public class ProdutoDAO extends BasicoDAO {
2
3     public static final String TABELA_PRODUTOS = "PRODUTOS";
4     public static final String COLUNA_ID = "_id";
5     public static final String COLUNA_NOME = "NOME";
6     public static final String COLUNA_ID_CATEGORIA = "ID_CATEGORIA";
7
8     public ProdutoDAO(Context ctx) {
9         super(ctx);
10    }

```

Código 4.47: Cabeçalho da classe *ProdutoDAO*

O *script* de criação da tabela de produtos no banco é exibido no Código 4.48. Este script será utilizado pela classe *DbAdapter* para criar esta tabela no banco, como pode ser observado no Código 4.43.

```

1     public static final String PRODUTOS_CREATE_TABLE = "CREATE TABLE "
2             + TABELA_PRODUTOS + " (" + COLUNA_ID
3             + " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUNA_NOME
4             + " TEXT NOT NULL," + COLUNA_ID_CATEGORIA + " INTEGER NOT NULL,"
5             + " FOREIGN KEY (" + COLUNA_ID_CATEGORIA + " ) REFERENCES "
6             + CategoriaDAO.TABELA_CATEGORIA + " (" + CategoriaDAO.COLUNA_ID
7             + " ) ON DELETE RESTRICT ON UPDATE CASCADE);";

```

Código 4.48: *Script* de criação da tabela *Produtos*

No Código 4.48 também é possível observar como utilizar uma chave estrangeira. Para fins de exemplo, deve-se considerar que a tabela *Categoria*, a qual a chave estrangeira faz referência, foi criada baseada na tabela *Produtos*.

O método *deProdutoParaContentValues* é responsável por converter o objeto *Produto* em um objeto *ContentValues*, como mostrado pelo Código 4.49. A classe *ContentValues* é usada para armazenar um conjunto de valores que podem ser processados pelo banco. Esse objeto funciona como um *HashMap*, com chaves associadas a valores, onde a *string* que representa o nome da coluna deve ser a chave.

```

1     public static ContentValues deProdutoParaContentValues(Produto produto) {
2         ContentValues values = new ContentValues();
3         values.put(COLUNA_NOME, produto.getNome());
4         values.put(COLUNA_ID_CATEGORIA, produto.getIdCategoria());
5         return values;
6     }

```

Código 4.49: Conversão de objeto *Produto* para objeto *ContentValues*

O Código 4.50 apresenta o método *criarProduto*. Este recebe um objeto *Produto* que, após ser convertido em um *ContentValues*, é adicionado ao banco pelo método *insert*.

```

1  public long criarProduto(Produto produto) {
2      ContentValues values = deProdutoParaContentValues(produto);
3      return mDb.insert(TABELA_PRODUTOS, null, values);
4 }
```

Código 4.50: Cadastro de *Produto*

É possível observar como remover um item do banco no Código 4.51. O método *remove* recebe como parâmetro o nome da tabela e a condição para remoção.

```

1  public boolean removerProduto(long idProduto) {
2      return mDb.delete(TABELA_PRODUTOS, COLUNA_ID + "=?",
3                         new String[] { String.valueOf(idProduto) }) > 0;
4 }
```

Código 4.51: Remoção de *Produto*

O Código 4.52 mostra como atualizar os dados de uma tabela do banco de dados, utilizando o método *update*. Este recebe como parâmetro o nome da tabela, um objeto *ContentValues* que contém os valores a serem alterados e a condição para atualização.

```

1  public boolean atualizarProduto(Produto produto) {
2      ContentValues values = deProdutoParaContentValues(produto);
3      return mDb.update(TABELA_PRODUTOS, values, COLUNA_ID + "=?",
4                         new String[] { String.valueOf(produto.getId()) }) > 0;
5 }
```

Código 4.52: Atualização de *Produto*

Os dados da tabela *Produtos* podem ser consultados de acordo com o Código 4.53. Toda consulta ao banco de dados deve ser efetuada pelo método *query*, que retorna um objeto do tipo *Cursor* que contém os dados coletados.

```

1  public Cursor consultarTodosProdutos() {
2      return mDb.query(TABELA_PRODUTOS, null, null, null, null, null, null);
3 }
```

Código 4.53: Consultar todos os *Produtos*

O Código 4.54 exemplifica uma consulta que retorna o *id* e o *nome* de todos os *Produtos* de uma determinada *Categoria*.

```

1  public Cursor consultarProdutosPorCategoria(long idCategoria)
2      throws SQLException {
3
4      Cursor mCursor = mDb.query(true, TABELA_PRODUTOS, new String[] { COLUNA_ID,
5          ,COLUNA_NOME }, COLUNA_ID.CATEGORIA + "=?", new String[] { String.
6          valueOf(idCategoria) }, null, null, null);
7
8      if (mCursor != null) {
9          mCursor.moveToFirst();
```

```

8     }
9
10    return mCursor;
11 }
```

Código 4.54: Consultar Produtos por Categoria

O objeto *Cursor* pode ser utilizado para alimentar diretamente a *interface* do sistema. Porém, este objeto não pode ser manipulado. Caso haja esta necessidade, é possível converter este objeto conforme demonstrado no Código 4.55. O método *deCursorParaProduto* converte um *Cursor* em um objeto *Produto*.

```

1  public static Produto deCursorParaProduto(Cursor c) {
2      if (c == null || c.getCount() < 1) {
3          return null;
4      }
5      Produto produto = new Produto();
6      produto.setId(c.getLong(c.getColumnIndex(COLUNA_ID)));
7      produto.setIdCategoria(c.getLong(c.getColumnIndex(COLUNA_ID_CATEGORIA)));
8      produto.setNome(c.getString(c.getColumnIndex(COLUNA_NOME)));
9      return produto;
10 }
```

Código 4.55: Conversão de objeto *Cursor* para objeto *Produto*

4.3 Depuração e testes automatizados

Apesar dos recursos de depuração não terem sido utilizados neste trabalho, nas próximas seções são abordados como depurar e como testar um aplicativo Android. Inicialmente, é demonstrado como funciona o processo de depuração. Em seguida, é exemplificado como implementar testes automatizados.

4.3.1 Depuração (*Debugging*)

O Android SDK fornece a maioria das ferramentas necessárias para se depurar um aplicativo, porém precisa de um depurador JDWP (*Java Debug Wire Protocol*) compatível, uma ferramenta capaz de percorrer o código, analisar os valores das variáveis e pausar a execução de um aplicativo.

O Eclipse já possui um depurador JDWP compatível e que não necessita ser configurado. Caso seja utilizada outra IDE torna-se necessário um depurador para comunicação com a máquina virtual.

Conforme mencionado anteriormente, no Android cada aplicativo é executado em seu próprio processo, e este possui sua própria máquina virtual que habilita apenas uma

porta para que o depurador possa se anexar. A Figura 4.7 apresenta o fluxo de um ambiente de depuração.

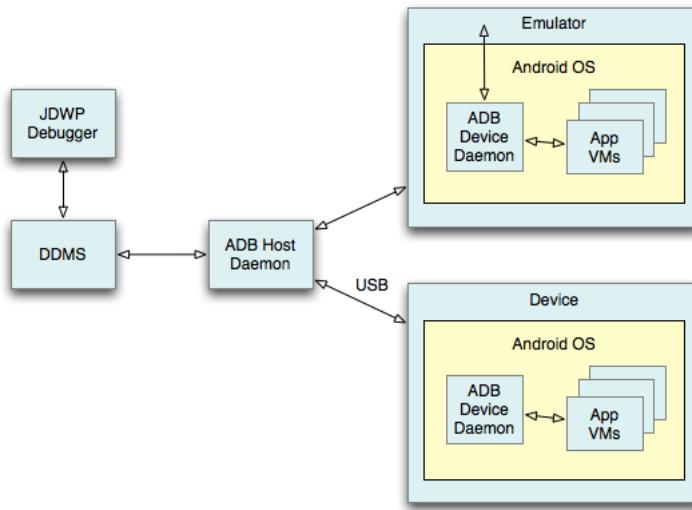


Figura 4.7: Fluxo da depuração

Quando se inicia o DDMS, este se conecta ao ADB (*Android Debug Bridge*) e um serviço de monitoramento de máquina virtual é criado, que notifica o DDMS quando uma VM é iniciada ou terminada. No momento em que a VM está em execução, o DDMS recupera a identificação desta e cria uma conexão com o depurador, através do ADB. Cada depurador pode ser anexado a uma única porta, mas o DDMS pode gerenciar vários depuradores.

4.3.1.1 O ambiente de depuração

Os principais componentes de um ambiente de depuração típico do Android são:

- ADB - age como um intermediário entre o dispositivo e o sistema de desenvolvimento, agindo como um gestor;
- *Dalvik Debug Monitor Server* (DDMS) - é um programa gráfico que se comunica com os dispositivos em execução;
- Dispositivos ou dispositivo Android Virtual (*Device or Android Virtual Device*) - o aplicativo deve ser executado em um dispositivo ou no emulador para que este possa ser depurado;
- JDWP - Depurador suportado pela *Dalvik Virtual Machine*, para que esse seja anexado a uma máquina virtual.

Além das ferramentas principais de depuração, o Android SDK fornece outras adicionais como:

- Visualizador de hierarquia e layout - Programa gráfico que permite a depuração e otimização da interface do usuário;
- *Trace View* - Visualizador gráfico que permite traçar o perfil de desempenho do aplicativo;
- Ferramentas Dev de aplicação Android - faz parte do SDK, sendo instalado por padrão em todas as imagens do sistema para ser utilizado pelo emulador. Permite ativar uma série de configurações no dispositivo que facilita o teste e a depuração do aplicativo.

4.3.1.2 Depurando com o Eclipse e o plugin ADT

O DDMS é uma ferramenta que funciona de forma autônoma que pode ser integrado ao Eclipse através do plugin ADT. Este fornece alguns recursos importantes que rapidamente lhe darão uma idéia de onde seu aplicativo está com problemas.

A Figura 4.8 apresenta uma tela básica do DDMS. Observe que o processo em destaque *com.android.email* está rodando no emulador tem atribuída a porta de depuração 8700, além da 8606. Isto significa que o DDMS está encaminhando os dados da porta 8606 para a porta de depuração padrão 8700.

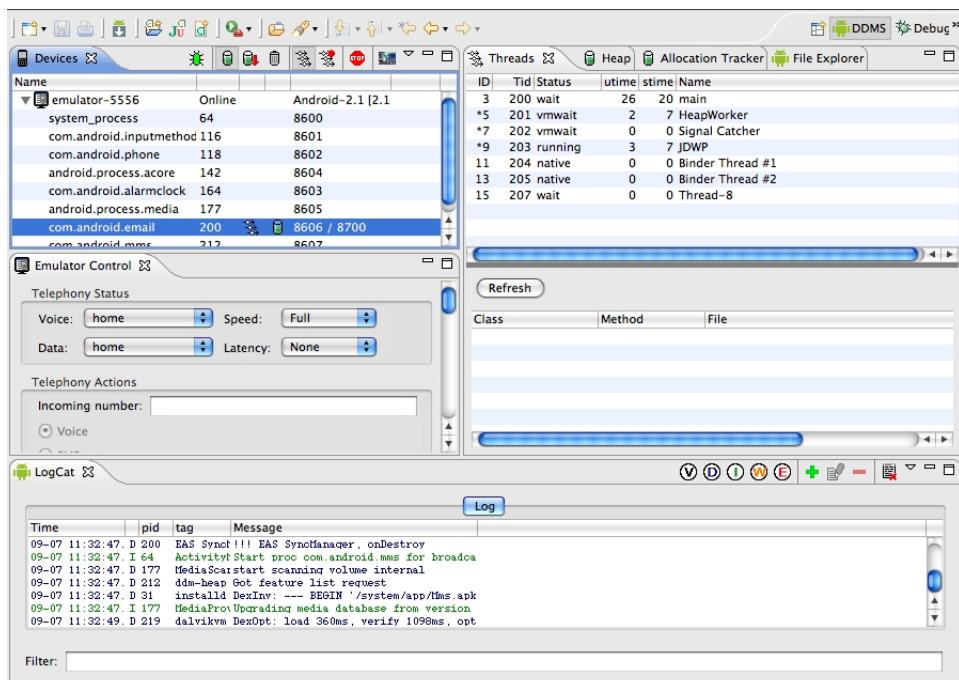


Figura 4.8: Depuração DDMS

- A perspectiva de depuração no Eclipse

Permite o acesso as seguintes guias:

- Depuração - mostra as aplicações depuradas anterior e atualmente, e seus segmentos em execução;
- Variáveis - quando os pontos de interrupção são definidos, exibe os valores das variáveis durante a execução do código;
- *Breakpoints* - exibe uma lista dos pontos de interrupção definidos no código do aplicativo;
- *LogCat* - permite visualizar mensagens de log do sistema em tempo real.

- A perspectiva DDMS

Permite que todos os recursos sejam acessados por dentro da IDE Eclipse. Estes recursos são os seguintes:

- Dispositivos - mostra a lista de aplicativos e AVDs que estão ligadas a ADB;
- Controle do Emulador - permite realizar funções do dispositivo;
- *LogCat* - permite visualizar mensagens de log do sistema em tempo real;
- *Threads* - mostra as *threads* em execução no momento dentro das VM;
- *Heap* - mostra a pilha de processos para uma VM;
- Rastreador de alocação - mostra a alocação de memória de objetos;
- Explorador de arquivos - permite explorar o sistema de arquivos do dispositivo.

4.3.2 Testes

O Android possui um *framework* de testes que é parte integrante do ambiente de desenvolvimento. Este fornece a arquitetura e ferramentas que permitem testar a aplicação no nível de unidade. Os testes são realizados utilizando o Eclipse com as ferramentas do SDK, com o *plugin* ADT, e o *JUnit*.

O SDK disponibiliza uma ferramenta de linha de comando para testes de interface, chamada de MonkeyRunner, conforme descrito em Android Developer (ANDROID, 2012l). Além disso há no mercado diversos outros *frameworks* e ferramentas para automatizar testes no Android como, por exemplo, Robotium e Robolectric.

Existe também a possibilidade de se utilizar a ferramenta JUnit para testar as classes do projeto que não dependem da API do Android. Para testar componentes Android existe o *Android JUnit Extensions*, que disponibiliza classes específicas para testar determinados componentes da plataforma. Estas permitem controlar um componente Android

independente de qual estágio se encontra o seu ciclo de vida, ou seja, este *framework* permite que se percorra todo o ciclo de um componente, passo a passo, como se este estivesse em execução.

A Figura 4.9 apresenta um resumo do processo de teste padrão. Podemos observar que existem dois pacotes, um guarda as classes do aplicativo e o outro as classes de teste. Estes são intermediados pelo *InstrumentationTestRunner*, que utiliza da ferramenta *MonkeyRunner* para testar a aplicação com os testes definidos no pacote de teste.

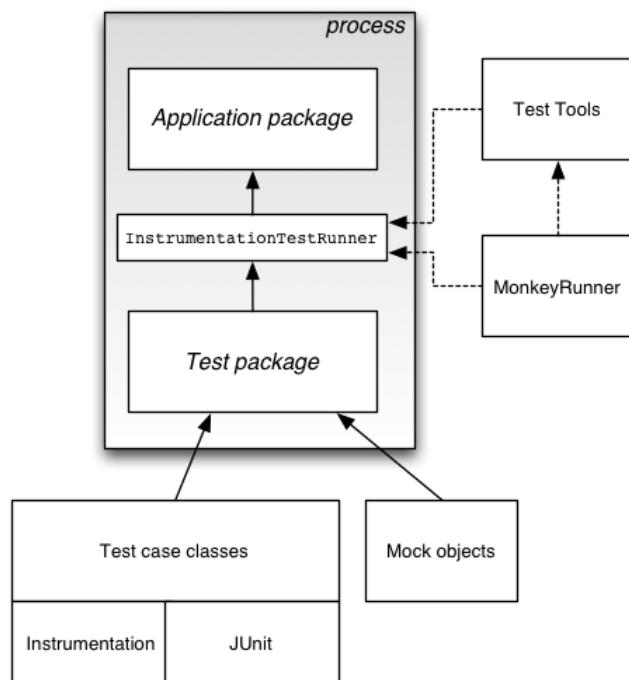


Figura 4.9: Processo de teste padrão

Geralmente é criado um novo projeto onde as classes de teste serão definidas. Este projeto tem total visibilidade do projeto que contém os códigos do aplicativo. Porém, esse procedimento não é obrigatório, podendo o desenvolvedor definir seus testes dentro do projeto principal, atitude que influencia diretamente no tamanho final do aplicativo.

Ao se utilizar do Eclipse para criação deste projeto, esse configura automaticamente o vínculo com o projeto que se pretende testar, já que este é definido no momento de sua criação. Este mantém a estrutura do projeto principal.

Os métodos de teste são organizados em classes de teste. Cada método representa um módulo de teste isolado que é executado individualmente.

A utilização de classes que simulam o comportamento de outras, conhecidas como *Mock*, facilita o isolamento dos testes. Estas classes podem simular objetos do tipo *context*, *Content Provider*, *Content Resolver*, *Service*, e, em alguns casos, podem simular uma *Intent*.

4.3.2.1 Criando o Projeto de Teste

Nesta seção será demonstrado como criar um projeto de teste utilizando o Eclipse aliado ao *plugin* ADT, baseado no conteúdo do site oficial do Android (ANDROID, 2012n). Deve-se considerar a pré-existência de um projeto que contém uma atividade chamada *SpinnerActivity*, que implementa um Spinner que será testado.

Para criar um projeto de teste para o projeto principal deve-se seguir os passos descritos abaixo.

1. No menu ***File*** do Eclipse abrir submenu ***New*** e clicar na opção ***Project...***;
2. Selecionar ***Android Test Project*** na pasta ***Android***. Em seguida, clicar no botão ***Next***;
3. Na tela seguinte, informar o nome do projeto de teste no campo ***Project Name*** e clicar em ***Next***;
4. Na próxima tela, marcar a opção ***An existing Android Project***, escolher o projeto que será testado na lista abaixo e clicar em ***Finish***;

Observe que o Eclipse gera a configuração básica do arquivo *AndroidManifest.xml* do projeto de teste, na qual encontra-se a definição do *instrumentation* apontando para o pacote do projeto principal. O Código 4.56 a seguir apresenta-o.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.android.example.spinner.test"
4     android:versionCode="1"
5     android:versionName="1.0">
6     <uses-sdk android:minSdkVersion="3" />
7     <instrumentation
8         android:targetPackage="com.android.example.spinner"
9         android:name="android.test.InstrumentationTestRunner" />
10    <application android:icon="@drawable/icon" android:label="@string/app_name">
11        <uses-library android:name="android.test.runner" />
12        ...
13    </application>
14 </manifest>

```

Código 4.56: O arquivo *AndroidManifest.xml* do projeto de teste

4.3.2.2 Classe de Teste

Para cada atividade do aplicativo a ser testada, deve-se criar uma classe de teste que implementará os métodos que definem os testes dessa atividade. Isto é feito com intuito de manter os testes organizados de forma a facilitar seu processo de implementação.

Na próxima seção será descrito como criar a classe de teste. Já nas seções posteriores, será exemplificado como implementar um teste nesta classe.

4.3.2.2.1 Adicionando a classe Caso de Teste .

Para se criar a classe de teste de uma atividade deve-se seguir os seguintes passos:

1. No menu **File** do Eclipse abrir submenu **New** e clicar na opção **Class**;
2. Na tela seguinte, editar os campos em evidência na Figura 4.10. Em seguida, clicar no botão **Finish**;
 - (a) *Package* - Verificar se o pacote corresponde ao mesmo do projeto de teste;
 - (b) *Name* - Definir o nome da classe;
 - (c) *Superclass* - Deve preencher com o pacote ”*android.test.ActivityInstrumentationTestCase2<?>*”, onde ? deve ser substituído pelo nome da classe do projeto principal que contém a atividade que será testada pela classe de teste.

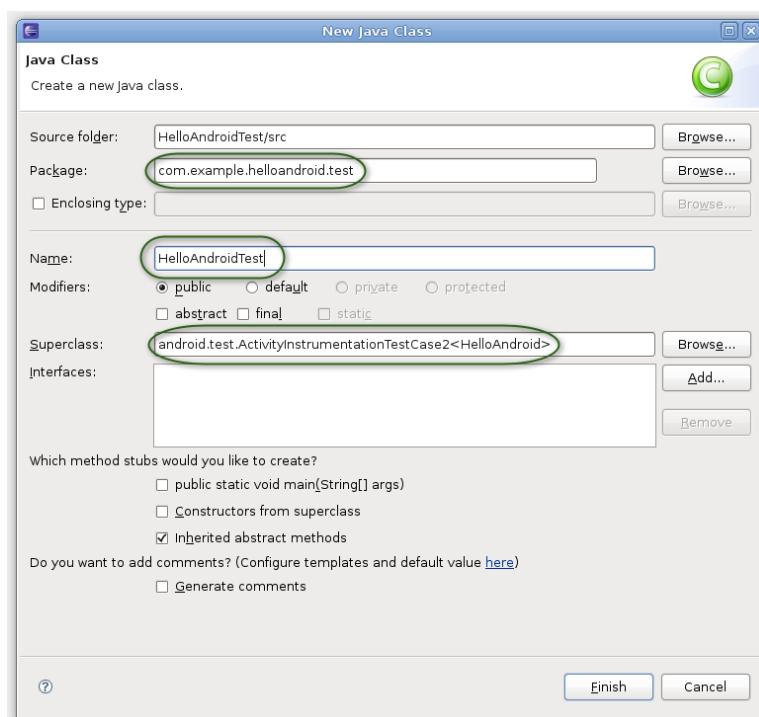


Figura 4.10: Cadastro de classe de teste

3. Por fim, a classe de teste será criada. Deve-se adicionar o *import* da classe do projeto principal para que esta seja reconhecida pela classe de teste.

4.3.2.2.2 Adicionando o construtor da classe de teste .

O construtor da classe de teste possui a função de enviar para o construtor da superclasse o nome do pacote e da classe que será testada, como pode ser observado no Código 4.57.

```
1 public SpinnerActivityTest() {
2     super("com.android.example.spinner", SpinnerActivity.class);
3 }
```

Código 4.57: Construtor da classe de teste

4.3.2.2.3 Adicionando o método *setUp()* .

O método *setUp()* é executado antes de cada teste, afim de preparar o ambiente, inicializando variáveis e desfazendo modificações dos testes anteriores. O Código 4.58 apresenta a definição do método *setUp()* da classe *SpinnerActivityTest* após a definição do construtor.

```
1 @Override
2 protected void setUp() throws Exception {
3     super.setUp();
4
5     setActivityInitialTouchMode(false);
6
7     mActivity = getActivity();
8
9     mSpinner =
10        (Spinner) mActivity.findViewById(
11            com.android.example.spinner.R.id.Spinner01
12        );
13
14     mPlanetData = mSpinner.getAdapter();
15
16 }
```

Código 4.58: Método de pré-configuração de variáveis de teste

Na linha 05 do Código 4.58 é realizada a chamada do método *setActivityInitialTouchMode(false)*, que é responsável por desativar a sensibilidade da tela, evitando que o processo de teste da interface seja comprometido. Já o método *getActivity()* resgata a referencia da atividade a ser testada e a inicializa. Em seguida, é criado um vínculo entre o *Spinner* presente na interface da atividade principal com um objeto que será utilizado para manipulá-lo.

Para que o Código 4.58 funcione, será necessário criar os objetos utilizados pelo método *setUp()* no corpo da classe de teste, conforme o Código 4.59.

```

1 private SpinnerActivity mActivity;
2 private Spinner mSpinner;
3 private SpinnerAdapter mPlanetData;
```

Código 4.59: Variáveis da classe de teste

4.3.2.2.4 Adicionando os testes de pré-condições .

O método *testPreConditions()* é responsável por verificar se a atividade a ser testada foi devidamente inicializada, implementando testes responsáveis por fazer esta verificação. O Código 4.60 é apresenta este método implementado, seguindo o exemplo.

```

1 public void testPreConditions() {
2     assertTrue(mSpinner.getSelectedItemListener() != null);
3     assertNotNull(mPlanetData);
4     assertEquals(mPlanetData.getCount(), ADAPTER_COUNT);
5 }
```

Código 4.60: Teste de pré-condições

Como é possível observar no Código 4.60, os testes automatizados são implementados utilizando os métodos do *JUnit*. Neste exemplo, foram executados três testes responsáveis, respectivamente, por:

- Verificar se o *Spinner* foi inicializado;
- Verificar se o adaptador que fornece valores para o *Spinner* é inicializado;
- Verificar se o adaptador contém o número certo de entradas.

A constante *ADAPTER_COUNT* deve ser inicializada com o valor correspondete no corpo da classe, conforme Código 4.61.

```

1 public static final int ADAPTER_COUNT = 9;
```

Código 4.61: Inicialização da constante *Adapter_Count*

4.3.2.2.5 Adicionando um teste de interface de usuário .

Nesta seção será criado um teste de interface que seleciona um item do *Spinner*, através de comandos que simulam a interação do usuário, permitindo avaliar se o que foi selecionado corresponde ao resultado esperado, que é informado em um *TextView* na mesma interface. Este teste visa demonstrar o poder do uso do *instrumentation* em testes automatizados de interface para aplicativos Android.

Como dito anteriormente, o teste simula uma interação entre um usuário e o sistema. Porém, é possível durante o teste solicitar a atividade testada que pratique alguma ação que facilita o processo do teste, como solicitar que o foco da aplicação esteja em um determinado elemento da tela. Esse tipo de solicitação a atividade deve ser feito por meio do método *runOnUiThread()*, conforme demonstrado no Código 4.62. Porém, é necessário tomar cuidado ao se utilizar deste artifício, já que este pode prejudicar a integridade do teste.

```

1 public void testSpinnerUI() {
2
3     mActivity.runOnUiThread(
4         new Runnable() {
5             public void run() {
6                 mSpinner.requestFocus();
7                 mSpinner.setSelection(INITIAL_POSITION);
8             }
9         }
10    );

```

Código 4.62: Começo da criação do método de teste *TestSpinnerUI*

No Código 4.62 é criado um teste chamado *testSpinnerUI()*. Sua primeira ação é solicitar o foco do *Spinner* na aplicação e, em seguida, que este selecione uma posição em sua lista. Para este exemplo, a constante *INITIAL_POSITION* equivale a zero.

Para simular a interação do usuário com a interface do aplicativo deve-se utilizar do método *sendKeys()*. Este recebe como parâmetro um *KeyEvent* ou uma *String* que representa uma ou mais ações executadas na tela do aplicativo. Caso o parâmetro seja uma *String*, para que esta represente mais de um comando por vez, estes devem ser separados por espaços. No Código 4.63 pode ser visto um exemplo de como utilizar este comando.

```

1 this.sendKeys(KeyEvent.KEYCODE_DPAD_CENTER);
2 for (int i = 1; i <= TEST_POSITION; i++) {
3     this.sendKeys(KeyEvent.KEYCODE_DPAD_DOWN);
4 } // end of for loop
5
6 this.sendKeys(KeyEvent.KEYCODE_DPAD_CENTER);

```

Código 4.63: Simulação de interação do usuário com o *Spinner*

O Código 4.63 demonstra como simular o uso dos botões direcionais de um dispositivo para navegar pelo *Spinner*. Considerar a constante *TEST_POSITION* igual a cinco.

A última etapa do método de teste é verificar se os resultado obtido no *TextView* equivale ao esperado após a seleção do *Spinner*, conforme demonstrado no Código 4.64.

```

1 mPos = mSpinner.getSelectedItemPosition();
2 mSelection = (String)mSpinner.getItemAtPosition(mPos);
3 TextView resultView =
4     (TextView) mActivity.findViewById(

```

```

5     com.android.example.spinner.R.id.SpinnerResult
6 );
7
8     String resultText = (String) resultView.getText();
9
10    assertEquals(resultText, mSelection);
11
12 } // end of testSpinnerUI() method definition

```

Código 4.64: Verificação dos dados obtidos

Para que o Código 4.64 funcione, será necessário criar os objetos utilizados pelo método *testSpinnerUI()* no corpo da classe de teste, conforme o Código 4.65.

```

1 private String mSelection;
2 private int mPos;

```

Código 4.65: Criação de variáveis para o método *testSpinnerUI()* funcionar

4.3.2.3 Executando os testes e visualizando os resultados

Para executar o projeto de teste do aplicativo, deve-se clicar na pasta do projeto com o botão direito do mouse e, em seguida, selecionar o botão **Run As -> Android JUnit Test**.

O emulador será aberto, caso ainda não esteja, e será possível observar a execução dos testes de interface através dele. Com a inicialização do teste, uma aba chamada *JUnit* poderá ser visualizada ao lado da guia do Gerenciador de Pacotes.

Esta aba apresenta dois sub-painéis, um superior, que resume os testes que foram executados, e um inferior, que apresenta as falhas dos testes em destaque. A Figura 4.11 apresenta a interface superior desta aba para um teste bem sucedido.

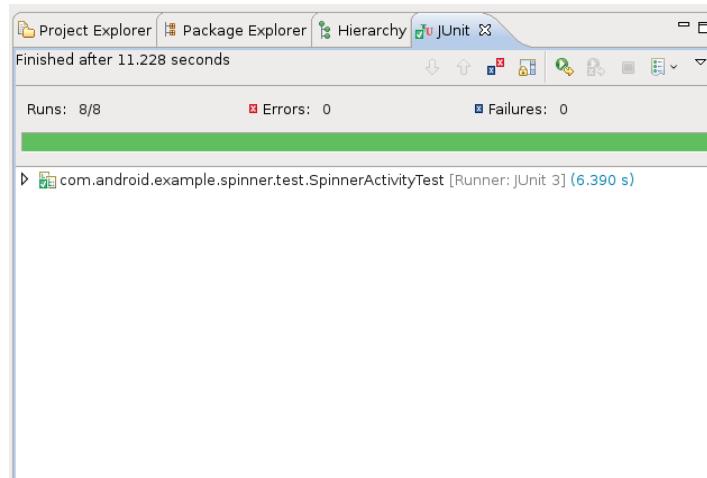


Figura 4.11: Aba *JUnit* para testes bem sucedidos

O painel acima resume o teste apresentando as seguintes informações:

- *Finished after x seconds* - Tempo transcorrido no teste;
- *Runs* - Número de testes executados;
- *Errors* - Número de erros encontrados durante a execução;
- *Failures* - Número de falhas encontradas durante a execução.

Uma barra de progresso se estende da esquerda para a direita durante a execução dos testes. Se todos os testes forem bem sucedidos a barra ficará verde, caso algum apresente problema esta ficará vermelha. Um exemplo de falha de um teste pode ser observado na Figura 4.12.

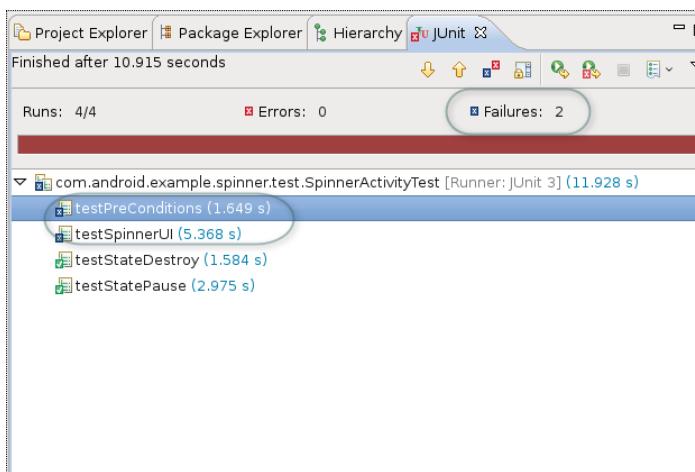


Figura 4.12: Aba *JUnit* em caso de falha de um ou mais testes

Abaixo da barra de progresso é apresentada uma lista contendo as classes testadas. É possível observar os resultados individuais de cada método destas classes clicando na seta em frente ao nome da classe. À direita do nome da classe ou do método, é possível visualizar o tempo decorrido na sua execução.

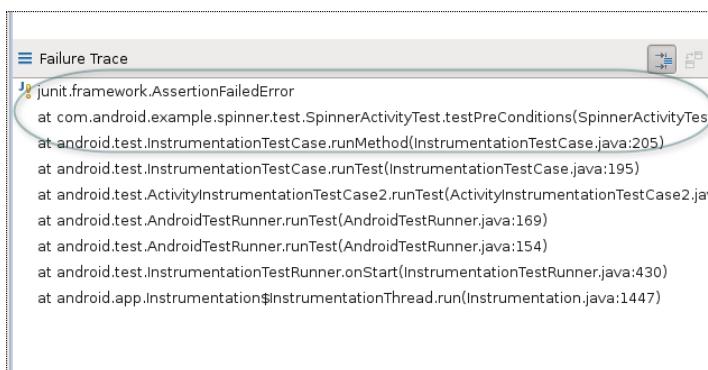


Figura 4.13: Painel *Failure Trace* detalhando uma falha no teste

Abaixo desta lista se encontra um painel, chamado *failure trace*, responsável por mostrar algumas informações sobre as falhas ocorridas durante o teste, se houver, conforme Figura 4.13.

4.4 Publicação

Nos Capítulos 3 e 4 deste trabalho, foram apresentadas de forma detalhada as fases pelas quais o desenvolvedor deve passar para criar uma aplicação para o sistema operacional Android. Estas fases culminam na publicação, como pode ser observado na Figura 4.14, abordada nesta seção.



Figura 4.14: Modelo sintético do processo de desenvolvimento da aplicação

A publicação é a última fase do processo de desenvolvimento de um aplicativo para o Android, e tem por finalidade tornar disponível a outros usuários os aplicativos desenvolvidos. A Google disponibiliza aos desenvolvedores um local onde estes podem lançar seus aplicativos, chamado de Google Play (PLAY, 2012). Porém, o desenvolvedor pode optar por hospedá-los em qualquer outro local, como em seu próprio site.

Uma das vantagens de se utilizar o Google Play é que, como este é integrado ao Android, sempre que uma nova atualização do aplicativo for lançada o usuário é avisado de imediato. Outra vantagem é que o aplicativo fará parte de uma base amplamente utilizada por usuários para busca de aplicativos, o que pode ajudar na disseminação deste. Como desvantagem do uso do ambiente do Google, podemos destacar a necessidade de o desenvolvedor pagar uma taxa de inscrição para criar uma conta onde poderá publicar seus aplicativos. Além disso, quando um aplicativo não for gratuito é cobrada uma taxa de transação em cima do valor total da venda, que atualmente corresponde a 30%.

Nas próximas seções são descritos os principais procedimentos para a publicação de um aplicativo no Google Play.

4.4.1 Procedimentos para publicação no Google Play

Antes de poder publicar um aplicativo no Google Play, é necessário criar um registro de desenvolvedor no site deste. O registro é efetuado após o pagamento de uma taxa de inscrição. Para que um aplicativo seja vendido, também é necessário ter uma conta no Google Checkout Merchant.

Feita a inscrição, o aplicativo deve ser preparado para ser publicado. A seguir são descritos os requisitos necessários para isto.

- **Configuração da Aplicação**

Nesta etapa deverão ser realizadas as seguintes tarefas:

- Retirar todas as linhas de códigos utilizados para gerar logs no sistema;
- Remover do arquivo *AndroidManifest.xml* o atributo `android:debuggable`, caso esteja sendo utilizado, ou trocar seu valor para `false`;
- Definir no arquivo *AndroidManifest.xml* os valores dos `android:versionCode` e `android:versionName`. O primeiro é utilizado para controle de atualização do aplicativo no Google Play e o segundo guarda o nome visualizado pelos usuários;
- Definir os ícones da aplicação no arquivo *AndroidManifest.xml*.

- **Construção e assinatura da versão do aplicativo**

O sistema Android requer que todas as aplicações instaladas sejam assinadas digitalmente com um certificado cuja chave privada é mantida pelo desenvolvedor. O Android usa o certificado como um meio de identificar o autor da aplicação e estabelecer uma relação de confiança entre as aplicações. Utilizando o Eclipse com o *plugin ADT*, é possível usar o Assistente de Exportação para executar os procedimentos de compilação e assinatura do aplicativo. Este assistente também permite que seja gerada uma nova chave privada no processo.

Além disso, também é necessário definir o Acordo de Licença de Usuário Final (*End User License Agreement*), visando a proteção do desenvolvedor, seja ele uma pessoa física ou uma organização, quanto à propriedade intelectual.

- **Teste da versão do aplicativo**

Antes da distribuição, o aplicativo deve ser testado no aparelho para o qual foi desenvolvido.

- **Verificação dos recursos do aplicativo**

Há a necessidade de averiguar se todos os recursos do aplicativo, tais como, arquivos de multimídia e gráficos estão atualizados e inclusos no aplicativo ou nos servidores de produção.

- **Serviços e servidores remotos**

Caso o aplicativo dependa de serviços ou servidores externos deve-se certificar que estes estejam seguros e prontos para uso.

Após a finalização destas tarefas será possível publicar o aplicativo no Google Play e este poderá ser instalado pelos usuários. Mais detalhes podem ser encontrados no site oficial Desenvolvimento Android (ANDROID, 2012h) e no livro Desenvolvimento de Aplicações Android (ROGERS et al., 2009).

5 ESTUDO DE CASO

Nos capítulos anteriores foram apresentados conceitos sobre a plataforma Android, como instalar e configurar o ambiente de desenvolvimento, como criar um projeto desde sua concepção até o momento de sua publicação.

Neste capítulo, será apresentado um estudo de caso idealizado com o objetivo de aplicar os conceitos estudados na criação de um aplicativo que possa trazer benefícios para o usuário e, também, que ajude a identificar particularidades presentes no processo de desenvolvimento para um dispositivo móvel que utiliza o Sistema Operacional Android. As particularidades encontradas serão descritas posteriormente.

5.1 A Aplicação

O estudo de caso deste trabalho visa a criação de um aplicativo, denominado Obra Fácil, que proporcionará ao usuário o controle de gastos atribuídos a compra de material em sua obra, podendo esta ser uma construção ou uma reforma.

O aplicativo possibilitará o cadastro, visualização, atualização e remoção de produtos, categorias de produtos, obras e listas de compras feitas durante a obra. Nas seções posteriores serão descritas as funcionalidades da aplicação.

5.1.1 A Interface Inicial

O aplicativo Obra Fácil possui uma tela inicial onde o usuário poderá escolher qual funcionalidade irá utilizar (Figura 5.1). Nesta é possível observar que existem quatro botões na parte inferior. Destes, os três botões são responsáveis por abrir uma nova interface para controle de **Categoria**, **Produto** e **Obra**, respectivamente. Ao pressionar o botão **Sair**, o aplicativo é fechado.



Figura 5.1: A tela inicial do aplicativo Obra Fácil

5.1.2 Gerência de Categorias para Produtos

Ao clicar no botão **Categoria**, na tela inicial, a interface de controle de categorias será aberta. Nesta é possível gerenciar o cadastro, atualização e remoção de categorias onde os produtos do sistema deverão ser alocados. Na Figura 5.2 é possível observar as três telas responsáveis por este gerenciamento.



Figura 5.2: Telas para gerência de categorias de produtos

A primeira e terceira tela da Figura 5.2 mostram as interfaces onde os dados das categorias poderão ser cadastrados e atualizados, respectivamente. A segunda tela apresenta a interface aberta pelo evento do botão pressionado na interface inicial. Nesta aparecerá uma lista contendo as categorias cadastradas. Caso uma categoria nesta lista seja pressionada por um segundo, um menu onde o usuário poderá escolher editar ou deletar tal categoria aparecerá, conforme Figura 5.3. Caso o usuário pressione o botão

menu do dispositivo, um menu onde este poderá escolher criar uma nova categoria ou retornar a tela inicial irá aparecer.



Figura 5.3: Menu que aparecerá caso uma categoria seja pressionada na lista

5.1.3 Gerência de Produtos

Ao clicar no botão **Produto**, na tela inicial, a interface de controle de produtos será aberta. Nesta é possível gerenciar o cadastro, atualização e remoção de produtos que irão compor as listas de compras elaboradas pelo usuário.

A Figura 5.4 apresenta duas telas. A primeira é a interface aberta pelo evento do botão pressionado na tela principal do aplicativo. Inicialmente, nessa serão listados todos os produtos cadastrados. Porém, ao selecionar uma categoria no *spinner*, presente na parte superior desta tela, a lista passará a conter apenas os produtos da categoria escolhida, exemplificado na segunda tela desta figura.



Figura 5.4: A lista de produtos cadastrados

As telas de cadastro e atualização de um produto podem ser vistas na Figura 5.5. O acesso a estas telas é feito de forma similar as de cadastro e atualização de categoria.



Figura 5.5: Interface de cadastro e atualização de produtos

5.1.4 Gerência de Obras

A interface de controle de Obras pode ser acessada ao clicar no botão **Obra** na tela inicial do aplicativo. Nesta é possível gerênciar o cadastro, atualização e remoção de obras na aplicação, além de proporcionar o acesso a interface onde as listas de compras destas obras poderão ser gerenciadas. As interfaces podem ser observadas na Figura 5.6.

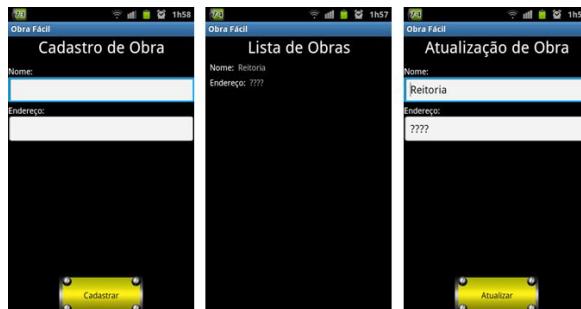


Figura 5.6: Telas para gerência de obras

Como é possível observar na Figura 5.6, as interfaces de gerência de Obra possuem muitas semelhanças com as de gerência de Categoria. De fato, as funcionalidades para o controle de Categoria são executadas da mesma forma que no controle de Obra. Nesta última, porém, o menu que é ativado ao selecionar uma obra na listagem de obras, como na Figura 5.7, possui uma nova função responsável por abrir uma tela onde serão listadas as listas de compras da obra selecionada.



Figura 5.7: Nova função presente no menu ao selecionar uma obra na lista

5.1.5 Gerência das Listas de Compras da Obra

Ao clicar no botão **Abrir Listas de Compras**, conforme explicado na seção anterior, serão apresentadas as listas de compras da obra escolhida (Figura 5.8). Ao pressionar a opção **Adicionar Novo**, no menu do dispositivo, uma nova interface será aberta para o cadastro de itens a uma nova lista.

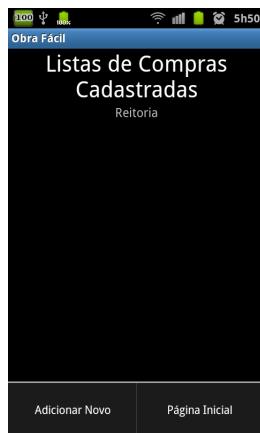


Figura 5.8: Listagens de Compras de uma Obra

A interface da tela de cadastro de produtos na lista de compras pode ser vista na Figura 5.9. No topo da tela existe um campo que oferecerá opções de produtos cadastrados de acordo com o que for digitado. Feita a escolha do produto, o usuário deverá clicar no botão **Adicionar** para continuar o processo de cadastro.

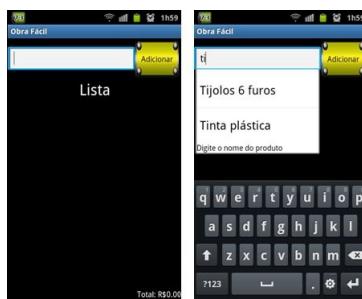


Figura 5.9: Interface para adição de produtos a uma lista de compras

Após clicar no botão **Adicionar** uma nova interface aparecerá onde será possível especificar a quantidade e o valor do produto escolhido. Para finalizar a adição deste na lista, o usuário deverá clicar no botão **Adicionar**. O aplicativo retornará a interface anterior, onde o produto adicionado agora será listado. Este processo irá se repetir até que todos os produtos pertencentes a lista sejam devidamente adicionados. Para efetuar o cadastro da lista de compras o usuário deverá clicar no botão **Salvar Lista**, no menu do dispositivo. Feito isso, a aplicação retornará a tela que contem a listagem das listas de compras da obra, onde a nova lista deverá constar. Este processo pode ser visto na Figura 5.10

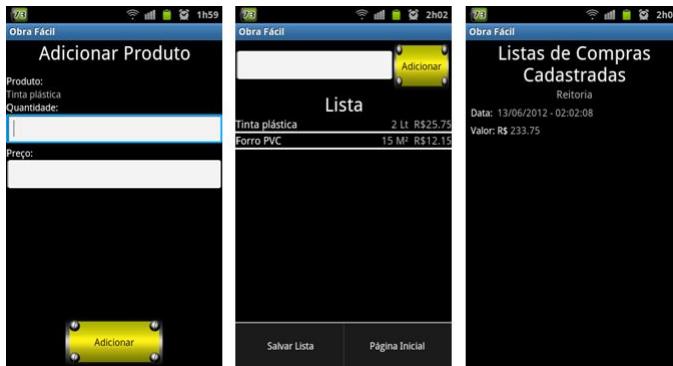


Figura 5.10: Telas para adição de produtos à lista de compras e finalização desta

5.2 O processo de desenvolvimento

O processo pelo qual um aplicativo passa desde sua concepção até a sua conclusão é chamado de processo de desenvolvimento.

Baseado no desenvolvimento do aplicativo descrito nas seções anteriores pretende-se descrever o processo de desenvolvimento e apresentar possíveis diferenças entre desenvolver para dispositivos móveis que utilizam a plataforma Android e para desktops e/ou para internet.

Neste projeto, o aplicativo foi dividido em partes de acordo com a função que estas exercem. Em seguida, foi definida a ordem de implementação de cada uma das partes, considerando as dependências de cada uma delas pelas demais. Foi definido então implementar o aplicativo na seguinte ordem: gerência de categorias para produtos, gerência de produtos, gerência de obras e, por fim, gerência de listas de compras.

O desenvolvimento de cada uma dessas partes passou pelo seguinte processo: criação da base de dados e de métodos para gerência dos mesmos, criação da interface gráfica, implementação das *activitys* e testes. A seguir, cada uma destas partes será descrita.

5.2.1 A base de dados

A criação da base de dados utilizada pela aplicação é uma parte importante do processo de desenvolvimento e merece atenção do desenvolvedor. A forma em que esta base será estruturada e a forma como seus dados serão manipulados influenciará diretamente na eficiência da aplicação, visto que os recursos dos dispositivos móveis são comumente mais limitados.

Devido a pouca quantidade de material disponível, tanto impresso como eletrônico, inclusive no site oficial do Android (ANDROID, 2012c), que expusesse com clareza a manipulação da base de dados, vale destacar o trabalho desenvolvido no blog Android

Brasil Projetos (COTTA, 2011), cuja leitura é recomendada por ser de fácil compreensão, bem explicado, com exemplos úteis e amplos. Por este motivo, o assunto banco de dados deste projeto foi baseado nesta documentação.

Um dos maiores desafios encontrados nesta parte do desenvolvimento foi como lidar com os dados provenientes de uma consulta ao banco de dados. Cada consulta ao banco gera seu resultado em variável do tipo *Cursor*. Este tipo de variável funciona de forma similar a um vetor, onde em cada uma de suas posições são armazenados uma ou mais informações. Porém, não é possível manipular estes dados, sendo necessário transferí-los para outro tipo de estrutura.

Por este motivo, um conjunto de classes reperesentando cada uma das tabelas do banco de dados foi criado. Assim, o resultado das consultas são passados para seus respectivos objetos sempre que for necessário um tratamento destes dados pela acticity. O Código 5.1 mostra um método responsável por fazer a conversão dos dados de uma *Cursor* para um objeto do tipo **Categoria**.

```

1  public static Categoria deCursorParaCategoria(Cursor c) {
2
3      if (c == null || c.getCount() < 1) {
4          return null;
5      }
6
7      Categoria categoria = new Categoria();
8      categoria.setId(c.getLong(c.getColumnIndex(COLUNA_ID)));
9      categoria.setNome(c.getString(c.getColumnIndex(COLUNA_NOME)));
10
11     return categoria;
12 }
```

Código 5.1: Conversão de *Cursor* para um objeto **Categoria**

5.2.2 A interface gráfica

A implementação da interface gráfica merece atenção por esta estar em contato direto com o usuário da aplicação. Uma aplicação bem organizada, com seus componentes gráficos dispostos de forma clara, simples e intuitiva se torna atraente aos olhos do usuário.

O Android proporciona ao desenvolvedor duas formas distintas para implementação da interface gráfica, como foi visto no Capítulo 2. Dentre estas duas opções, foi escolhida a forma que cria um arquivo XML onde o layout de cada interface é definido.

Durante o desenvolvimento, ficou claro que este método facilita a criação do layout do aplicativo. Além de não necessitar de um grande conhecimento sobre XML por parte do desenvolvedor, a implementação se mostrou simples tendo como resultado códigos claros.

O Código 5.2 demonstra o layout de cadastro de categoria:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent">
6
7     <TextView android:text="Cadastro de Categoria"
8         android:textSize="27sp"
9         android:gravity="center"
10        android:layout_height="wrap_content"
11        android:layout_width="fill_parent"/>
12
13     <LinearLayout android:orientation="vertical"
14         android:layout_width="fill_parent"
15         android:layout_height="wrap_content">
16
17         <LinearLayout android:orientation="horizontal"
18             android:layout_width="fill_parent"
19             android:layout_height="20px"/>
20
21         <TextView android:text="Nome:"
22             android:layout_width="wrap_content"
23             android:layout_height="wrap_content"/>
24
25         <EditText android:id="@+categoria_cadastro/nome"
26             android:text=""
27             android:layout_width="fill_parent"
28             android:layout_height="wrap_content"/>
29
30     </LinearLayout>
31
32     <LinearLayout android:orientation="horizontal"
33         android:layout_width="fill_parent"
34         android:layout_height="wrap_content"
35         android:baselineAligned="true"
36         android:gravity="center">
37
38         <Button android:id="@+categoria_cadastro/btnCadastrarCategoria"
39             android:text="Cadastrar"
40             android:layout_height="wrap_content"
41             android:layout_width="wrap_content"
42             android:background="@drawable/meu_botaو"/>
43
44     </LinearLayout>
45 </LinearLayout>
```

Código 5.2: Arquivo XML com a definição do layout de cadastro de Categoria

O layout gerado pelo XML acima, Código 5.2, pode ser visto na Figura 5.11.

Outro fator que contribuiu para o desenvolvimento do projeto foi a possibilidade de visualizar as alterações feitas neste código sem a necessidade de executar o aplicativo. O Eclipse junto ao plugin ADT proporcionam essa função durante a implementação do código XML, exclusivamente. Sendo assim, esta se torna outra vantagem da utilização do XML ao invés da implementação em Java.

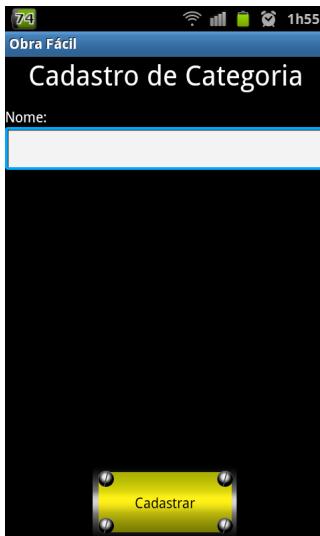


Figura 5.11: Layout gerado pelo XML de cadastro de Categoria

5.2.3 Activity

Cada tela possui uma Activity que é responsável por estabelecer a interface entre o banco de dados e o interface gráfica do aplicativo. Esta deve gerenciar as funções da tela, os dados que serão apresentados ao usuário e, também, os informados pelo mesmo.

A primeira coisa a se considerar na construção de uma activity é seu ciclo de vida (descrito no Capítulo 2). Ao contrário de aplicações Desktop, um aplicativo Android necessita estar preparado para mudanças de estado do dispositivo. Por exemplo, se uma ligação é recebida durante a utilização do aplicativo, este perde o foco e consequentemente é pausado. O aplicativo precisa saber como lidar com esta situação, que, neste caso, será tratada no estado chamado *onPause()*. Porém, nem todos os estados deste ciclo necessitam ser implementados, o que vai variar de aplicação para aplicação. No aplicativo Obra Fácil foram implementados três destes estados: *onCreate()*, *onResume()* e *onPause()*.

O estado *onCreate()* é o único estado de implementação obrigatória, pois é neste onde a activity ganhará vida. Como exemplo, parte do código deste método criado na activity de cadastro de categoria pode ser visto no Código 5.3:

```

1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4
5      setContentView(R.layout.categoria_cadastro);
6
7      txtTitulo = (TextView) findViewById(R.layout.categoria_cadastro.titulo);
8      txtNome = (EditText) findViewById(R.layout.categoria_cadastro.nome);
9      btnCadastrarCategoria = (Button) findViewById(R.layout.categoria_cadastro.
    btnCadastrarCategoria);

```

Código 5.3: *onCreateCategoria Teste*

Como pode ser observado no Código 5.3, a linha 5 define o layout desta activity. Entre as linhas 7 e 9 são criadas conexões entre campos e botões do layout e variáveis Java para que a activity consiga manuseá-los.

Desconsiderando a questão do ciclo de vida da atividade, tanto a manipulação de dados e o controle da interface são feitos de forma similar aos programas para desktops e para internet.

5.3 Testes automatizados

Dentre as ferramentas de automação de testes existentes para aplicativos Android, foi escolhido utilizar o *framework* baseado no *JUnit*, *framework* para testes Java. Seu uso terá por objetivo simular a utilização de uma *activity* e avaliar se esta responde de forma adequada aos estímulos do usuário.

Uma das principais diferenças na criação de testes para o Android é que estes devem ser criados em um projeto a parte que terá esta finalidade. O Eclipse facilita a criação deste projeto disponibilizando uma interface onde é possível criar tal projeto e sua estrutura base de forma simples. Além disto, o vincula ao projeto principal onde o aplicativo é implementado.

Cada classe de teste deve ser vinculada a uma *activity* do projeto principal para que esta possa ser testada. Um exemplo disto pode ser visto no Código 5.4.

```

1 public class ObraListaActivityTest extends
2 ActivityInstrumentationTestCase2<ObraCadastroActivity> {
3
4     public ObraListaActivityTest() {
5         super("iff.tcc.obrafacil", ObraCadastroActivity.class);
6     }
7     ...
8 }
```

Código 5.4: Exemplo de vinculo da classe de teste com a classe do projeto principal.

A implementação dos testes automatizados pode ser feita de forma fácil por aqueles que estiverem familiarizados com o *JUnit*. O Código 5.5 exemplifica um teste.

```

1 public void testPreConditions() {
2     assertTrue(textViewTitulo != null);
3     assertTrue(titulo != null);
4     assertTrue(editTextNome.getText().toString().equals(""));
5     assertTrue(editTextEndereco.getText().toString().equals(""));
6 }
```

Código 5.5: Código de teste implementado.

O exemplo presente no Código 5.5 demonstra a utilização do método *assertTrue*

que verifica se a condicional informada a este é verdadeira. Caso alguma destas condições seja falsa o teste irá falhar.

```

1 public void testDePreenchimentoDeEditTexts() {
2
3     obraCadastroActivity.runOnUiThread(new Runnable() {
4         @Override
5         public void run() {
6             editTextNome.requestFocus();
7         }
8     });
9
10    this.sendKeys("N O M E" + " DPAD_DOWN " + "E N D E R E C O");
11
12    assertEquals("nome", editTextNome.getText().toString());
13    assertEquals("endereco", editTextEndereco.getText().toString());
14 }
```

Código 5.6: Simulação de preenchimento de campos.

O Código 5.6 objetiva avaliar se os dados informados pelo usuário estão sendo preenchidos corretamente. Entre as linhas 3 e 8 o teste é iniciado colocando foco no *EditText* chamado **EditTextNome**. Na linha 10 o comando *sendKeys* simula o preenchimento dos *EditText* **EditTextNome** e **EditTextEndereco** pelo usuário. Já nas linhas 13 e 14 o método *assertEquals* verifica se o conteúdo dos *EditText* é igual ao que foi preenchido anteriormente.

5.4 Publicação

O aplicativo desenvolvido neste estudo de caso ainda é um protótipo e, além disso, este ainda não possui todos os testes implementados. Por estes motivos, o aplicativo não foi submetido a publicação.

Porém, os resultados obtidos ao final deste estudo de caso estão disponibilizados no site de compartilhamento de projetos GitHub (VIANA; CASTRO, 2012). Neste é possível encontrar uma cópia deste trabalho e o código fonte do estudo de caso apresentado, além do instalador do aplicativo gerado (.apk). Para instalar este aplicativo basta o copiar para o dispositivo móvel e executá-lo.

O código fonte será acompanhado da Licença GNU GPL (Licença Pública Geral) que possibilitará qualquer usuário copiar, utilizar, alterar e distribuir este aplicativo. Como qualquer licença de software, esta também exige a execução dos termos dispostos a fim de assegurar os direitos do autor ou desenvolvedor e do usuário final.

CONCLUSÃO

Este trabalho apresentou o Sistema Operacional Android, focando nas informações que serão necessárias àqueles que desejam desenvolver aplicativos para esta plataforma. Neste processo, a estrutura do Android foi descrita e ferramentas de desenvolvimento foram apresentadas. Além disso, os principais recursos Java disponíveis para tal desenvolvimento foram devidamente descritos e exemplificados, culminando na criação de um aplicativo. Muitos foram os desafios encontrados durante este processo, como a carência de material com informações detalhadas sobre determinados assuntos e também as mudanças de paradigma para desenvolvimento nesta plataforma. Dentre estas pode-se destacar a necessidade de tratamento do aplicativo em função de ações específicas dos dispositivos móveis, como, por exemplo, recebimento de mensagens de texto e de voz, que possuem prioridade e interferem no funcionamento deste.

Como trabalhos futuros pretende-se aprimorar a interface do aplicativo, além de implementar novos recursos que tornarão a utilização deste pelo usuário melhor. Dentre os novos recursos destacamos a implementação de um cadastro de lojas, onde estas poderão ser mapeadas por meio do uso de GPS. A criação deste cadastro possibilitará o aplicativo informar ao usuário o local mais próximo onde este poderá encontrar um produto específico, quais lojas possuem um determinado produto com o menor preço, dentre outros. Além disso, pode-se implementar diversos relatórios que retornarão informações encontradas através de uma mineração dos dados até então cadastrados. Por fim, pretende-se publicar o aplicativo desenvolvido na loja virtual do Android, Google Play.

REFERÊNCIAS BIBLIOGRÁFICAS

- ABLESON, F. *Introdução ao Desenvolvimento do Android*. 2009. Disponível em: <<http://www.ibm.com/developerworks/br/library/os-android-devel/>>.
- ALLIANCE, O. H. *Open Handset Alliance*. 2011. Disponível em: <<http://www.openhandsetalliance.com>>.
- ANDROID, D. *Android Architecture*. 2012. Disponível em: <<http://developer.android.com/images/system-architecture.jpg>>.
- ANDROID, D. *Android Hello Views*. 2012. Disponível em: <<http://developer.android.com/guide/topics%20ui/declaring-layout.html>>CommonLayouts>.
- ANDROID, D. *Developer Android Fundamentals*. 2012. Disponível em: <<http://developer.android.com/guide/components/fundamentals.html>>.
- ANDROID, D. *Developer Android FundamentalsActivities*. 2012. Disponível em: <<http://developer.android.com/reference/android/app/Activity.html>>.
- ANDROID, D. *Developer Android Guide*. 2012. Disponível em: <<http://developer.android.com/guide/components/index.html>>.
- ANDROID, D. *Developer Android Guide Data*. 2012. Disponível em: <<http://developer.android.com/guide/topics%20data/data-storage.html>>.
- ANDROID, D. *Developer Android Reference Widget*. 2012. Disponível em: <<http://developer.android.com/guide/topics%20appwidgets/index.html>>.
- ANDROID, D. *Developer Android Resources*. 2012. Disponível em: <<http://developer.android.com/guide/topics%20resources/available-resources.html>>.
- ANDROID, D. *Developer Android SDK*. 2012. Disponível em: <<http://developer.android.com/sdk/index.html>>.
- ANDROID, D. *Developer Bundle*. 2012. Disponível em: <<http://developer.android.com/reference/java/util/ResourceBundle.html>>.
- ANDROID, D. *The Development Process for Android Applications*. 2012. Disponível em: <<http://developer.android.com/tools/workflow/index.html>>.
- ANDROID, D. *Monkey Runner/UI Application*. 2012. Disponível em: <<http://developer.android.com/guide/developing/tools/monkey.html>>.
- ANDROID, D. *Plataforma versões*. 2012. Disponível em: <<http://developer.android.com/about/dashboards/index.html>>.
- ANDROID, D. *Teste*. 2012. Disponível em: <http://developer.android.com/tools/testing/activity_test.html>.

ANDROIDOPENSOURCE. *A história do Android*. 2012. Disponível em: <<http://source.android.com/about/index.html>>.

BIRK, J. *Desenvolvendo Soluções com Android*. 2010. Disponível em: <<http://www.slideshare.net/jgbirk/desenvolvendo-solucoes-com-android>>.

BLOOMBERGBUSINESSWEEK. *Business Week*. ago. 2005. Disponível em: <<http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal>>.

celeiroandroid.blogspot celeiroandroid.blogspot celeiroandroid.blogspot celeiroandroid.blogspot. *Content Provider*. 2011. Disponível em: <<http://www.brighthub.com/mobile/google/articles/29340.aspxixzz1BO7NjEJN>>.

COTTA, J. C. B. *Introdução ao banco de Dados Android*. 2011. Disponível em: <<http://www.androidbrasilprojetos.org/category/tutoriais/>>.

DALVIK. *Dalvik Virtual Machine*. 2008. Disponível em: <<http://dalvikvm.com>>.

DEVELOPERS, G. *Android Developer Challenge*. 2007. Disponível em: <<https://developers.google.com/android/adc/?hl=pt-BR>>.

EXAME.COM. *Android será plataforma móvel mais usada em 2015*. 2010. Disponível em: <<http://exame.abril.com.br/tecnologia/android/noticias/android-sera-plataforma-movel-mais-usada-2015-597901>>.

FRAGA, R. *Android e Mercado*. jul. 2012. Disponível em: <<http://googlediscovery.com/2012/07/14/android-e-ios-dominam-90-dos-smartphones-nos-eua/>>.

GIVOCE. *A ascensão do android*. 2012. Disponível em: <<http://qivoce.com/android-o-que-e-e-sua-ascensao/>>.

GLOBO.COM, G. *Google da premio de 10 milhoes*. nov. 2007. Disponível em: <<http://g1.globo.com/Noticias/Tecnologia/0,,MUL179123-6174,00-GOOGLE+DA+PREMIO+DE+US+MILHOES+PARA+APLICATIVOS+DE+CELULAR-.html>>.

INTERESSANTE, S. *A história do Android*. 2009. Disponível em: <<http://super.abril.com.br/galerias-fotos/conheca-historia-android-sistema-operacional-mobile-google-688822.shtml0>>.

JOBSTRAIBIZER, F. *Criação de Aplicativos para Celulares - Com Google Android*. [S.l.]: Digerati Books, 2009.

LABS, X. *Infográfico*. 2011. Disponível em: <<http://www.xcubelabs.com/the-android-story.php>>.

LECHETA, R. R. *Google Android - Aprenda a Criar Aplicações para Dispositivos Móveis com o Android Sdk*. [S.l.]: Novatec, 2009.

MORIMOTO, C. E. *Entendendo o Google Android*. abr. 2008. Disponível em: <<http://www.hardware.com.br/artigos/google-android/>>.

- MURPH, D. *Android 4.1 Jelly Bean*. 2012. Disponível em: <<http://www.engadget.com/2012/06/28/android-4-1-jelly-bean-review-a-look-at-whats-changed-in-googl/>>.
- OGLIARI, R. da S. *Começando a criar interfaces gráficas com Android*. 2010. Disponível em: <<http://www.mobilidadetudo.com>>.
- PEREIRA, L. C. O.; SILVA, M. L. da. *Android para Desenvolvedores*. [S.l.]: Brasport, 2009.
- PLAY, G. *Site Google Play*. 2012. Disponível em: <<https://play.google.com>>.
- ROGERS, R. et al. *Desenvolvimento de Aplicações Android*. [S.l.]: Novatec, 2009.
- ROQUEIROJP. *Imagen do Ice cream sandwich*. 2012. Disponível em: <<http://rokeirojp.blogspot.com.br/2012/02/tutorial-android-4-ice-cream-sandwinch.html>>.
- SILVA, L. A. da. *Apostila de Android*. 2010. Disponível em: <<http://www.apostilaandroid.ueuo.com>>.
- SILVEIRA, F. *Desenvolvendo para Android*. 2010. Disponível em: <<http://www.felipesilveira.com.br/desenvolvendo-para-android/>>.
- STARCK, D. *Instalar e testar Android*. 2010. Disponível em: <<http://www.tecmundo.com.br/5005-como-instalar-e-testar-o-android-no-computador.htm>>.
- TEAM, A. *Android Widget*. 2011. Disponível em: <<http://www.ibm.com/developerworks;br/opensource/tutorials/os-eclipse-androidwidget/>>.
- TELETIME. *Android no mercado brasileiro*. jul. 2012. Disponível em: <<http://www.teletime.com.br/18/07/2012%20-%20android-cresce-mas-symbian-ainda-e-lider-de-mercado-no-brasil-segundo-pesquisa/tt/289697/news.aspx>>.
- TSUHARESU. *A história do Android*. 2010. Disponível em: <<http://www.euandroid.com.br/geral/2010/11/historia-do-android/>>.
- VIANA, E.; CASTRO, I. de. *Código Fonte - Obra Fácil*. 2012. Disponível em: <<https://github.com/igorcpontes/tccsi>>.