

# **Desenvolvimento Android Básico**

(por Maurício Salamon - <http://mauriciosalamon.com.br>)



## Conteúdo

1. Introdução à plataforma Java .....	4
1.1. Comandos básicos .....	6
1.2. Tipos de dados.....	7
1.3. Operadores .....	8
1.4. Operadores relacionais .....	9
1.5. Comentários .....	10
1.6. Pacotes, Classes e Objetos .....	11
1.7. Modificadores de acesso e encapsulamento.....	17
1.8. Coleções .....	19
1.9. Padrão de projeto Observer, Eventos e Listeners.....	21
1.10. Threads e Exceções .....	25
2. Introdução ao sistema operacional Android.....	28
2.1. Desenvolvimento Android .....	30
2.2. Preparação do Ambiente de Desenvolvimento .....	32
2.3. HelloWorld! .....	34
2.4. Estrutura de um projeto e XML .....	36
2.5. Compilação, execução e depuração .....	39
2.6. Empacotamento, publicação e instalação do app.....	45
3. Conceitos Básicos .....	49
3.1. Activity, Intent e IntentFilter.....	50
3.2. Service.....	54
3.3. ContentProvider .....	58
3.4. Processo e Thread .....	61
3.5. Manifesto.....	63
4. Interface Gráfica.....	65
4.1. Layouts.....	67
4.2. Componentes de tela .....	74
4.3. Menus.....	81
4.4. Janelas de Diálogo.....	84
4.5. Notificações.....	86
4.6. Drag and Drop.....	91
4.7. Recursos .....	94
5. Conceitos Avançados .....	96
5.1. Permissões e Segurança .....	97
5.2. Preferências de Usuário.....	100

5.3. Persistência em Banco de Dados (SQLite) .....	102
5.4. Câmera e Arquivos.....	105
5.5. GPS e Mapas .....	108
5.6. Bluetooth.....	111
5.7. Sensores.....	114
5.8. OpenGL, games, redes sociais e tendências.....	117
4. Exercícios.....	120
Anotações.....	123

## 1. Introdução à plataforma Java

*Java é mais que uma linguagem de programação, é uma plataforma. Partindo deste conceito, consegue-se analisar o histórico, as principais utilidades, vantagens e desvantagens de trabalhar com o Java, utilizando exemplos de código e explorando os principais comandos.*

*Criada no início dos anos 90 pela Sun Microsystems com o objetivo de ser uma linguagem independente de plataforma de execução, Java segue os paradigmas de linguagem Orientada a Objetos (OO), o que era uma tendência no desenvolvimento de software na época de seu surgimento. Diferentemente de outras linguagens, cujos códigos eram compilados para código-nativo, Java é compilado em bytecode, sendo este interpretado e executado por uma máquina virtual, no caso a JVM (Java Virtual Machine). Isto permite que o mesmo programa desenvolvido em Java seja executado em sistemas operacionais diferentes (como Windows e Linux, por exemplo).*



*A Sun disponibiliza a maioria das extensões e distribuições Java de forma gratuita, obtendo receita com softwares especializados e mais robustos. Dentre as diversas extensões e distribuições Java, pode-se destacar:*

- *JSE (Standard Edition): é a plataforma Java de utilização mais comum, possui todos os recursos para o desenvolvimento de uma aplicação de ambiente de trabalho completa, como bibliotecas de código, compilador, máquina virtual e ferramentas e APIs auxiliares;*
- *JEE (Enterprise Edition): é a plataforma Java mais completa, possui todos os recursos da JSE e mais ferramentas e recursos específicos para o ambiente corporativo (servidores) e web, como novas APIs de acesso a dados e segurança;*

- *JME (Micro Edition): é a plataforma Java mais reduzida, específica para utilização em sistemas embarcados e móveis, contém menos recursos que a JSE, mas com APIs específicas, como de gerenciamento de MIDIs, Bluetooth, Wireless, GPS, etc;*
- *JSP (Server Pages): é a tecnologia utilizada para a criação de páginas web, tem um comportamento semelhante ao ASP da Microsoft e o PHP e normalmente é utilizada em combinação com o JEE (com Servlets, componentes de integração Java, HTML e XML);*
- *JRE (Runtime Edition): é o conjunto de ferramentas necessárias para a execução de aplicações Java, como a JVM (máquina virtual), algumas bibliotecas de execução e outros utilitários;*

*Em 2006, Java foi parcialmente liberado como software livre sob a licença GNU (General Public License), sendo completamente licenciado como livre em 2007. A Sun foi comprada pela Oracle em 2009, o que não interferiu na licença de distribuição Java. Dessa forma, Java possui uma comunidade muito grande de desenvolvedores, o que dá à linguagem a vantagem de possuir uma enorme gama de frameworks (alguns pagos, mas a maioria gratuita), úteis para diversas ocasiões de desenvolvimento, e IDEs (Integrated Development Environment), ou seja, ambientes de desenvolvimento. A ferramenta mais utilizada para o desenvolvimento Java até 2011 é o IDE Eclipse.*

*Orientado ao desenvolvimento baseado em plugins, o Eclipse é uma ferramenta de código-fonte livre robusta, capaz de executar as principais operações de produção de software (codificação, compilação, depuração, modelagem, persistência, teste, publicação, etc.). Os diversos plugins disponíveis dão ao Eclipse a capacidade de produzir códigos não só Java, mas também PHP, Python, Ruby, Action Script, dentre outros, além de agregar funcionalidades diversas na codificação dessas linguagens. Um dos plugins mais conhecidos do Eclipse é o ADT (Android Development Tools), que combinado com o SDK (Standard Development Kit), biblioteca de desenvolvimento Android disponibilizada de forma gratuita pela Google, dá ao desenvolvedor o poder de não só programar Java para Android, mas também emular dispositivos para testes e depuração.*

*As versões mais robustas de Java possuem bibliotecas de suporte à interface gráfica, como a “Swing”, que possui elementos de controle de layout (em tabela, em camadas, etc.) e componentes de tela, além de suporte a controle de eventos para tratamento desses componentes.*

## 1.1. Comandos básicos

*A capacitação Android é focada no desenvolvimento a partir de Java. Assim, é importante o conhecimento dos principais comandos utilizados na linguagem, sendo aqui apresentados de forma sintetizada.*

*Ao estudar Java, fala-se de Classes e Objetos. Resumidamente, Classe é a generalização de uma espécie e objeto é um representante único desta espécie. Classes possuem atributos (informações/propriedades) e métodos (ações/funções).*

*Exemplo de código Java:*

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

*Toda classe Java, quando executada diretamente, é iniciada através do método “main”. Desta forma, no exemplo acima, ao executar a classe “HelloWorld”, o programa exibirá em console, através do método “System.out.println();”, a frase “Hello, World!”.*

## 1.2. Tipos de dados

<i>Tipo</i>	<i>Bits</i>	<i>Bytes</i>	<i>Mínimo</i>	<i>Máximo</i>	<i>Ex.:</i>
<i>byte</i>	8	1	$-2^7$	$2^7-1$	99
<i>short</i>	16	2	$-2^{15}$	$2^{15}-1$	9999
<i>int</i>	32	4	$-2^{31}$	$2^{31}-1$	9999999999
<i>long</i>	64	8	$-2^{63}$	$2^{63}-1$	9999999999999999
<i>float</i>	32	4	-	-	99999999.99
<i>double</i>	64	8	-	-	9999999999999999.99
<i>boolean</i>	<i>verdadeiro/falso</i>	<i>true/false</i>	-	-	<i>True</i>
<i>char</i>	<i>caractere</i>	1	-	-	'a'
<i>String</i>	<i>caracteres</i>	-	-	-	"abc"

*O tipo String não é um tipo primitivo, mas sim uma Classe. Diferentemente de outras Classes, "String" não necessita de instanciação (uso do comando "new"), apenas atribuição normal. Porém, ao ser atribuído, um "String" pode ser modificado apenas pela utilização de métodos ou alguns operadores específicos.*

## 1.3. Operadores

<i>Operador</i>	<i>Descrição</i>	<i>Ex.:</i>
=	<i>Atribuição</i>	<i>int x = 1; // x é 1</i>
+	<i>Adição</i>	<i>int x = 9+1; // x é 10</i>
-	<i>Subtração</i>	<i>int x = 4-3; // x é -1</i>
/	<i>Divisão</i>	<i>int x = 4/2; // x é 2</i>
*	<i>Multiplificação</i>	<i>int x = 1*5; // x é 5</i>
%	<i>Módulo</i>	<i>int x = 4%2; // x é 0</i>
+=	<i>Adição seguida de atribuição</i>	<i>int x = 1; x+= 10; // x é 11</i>
-=	<i>Subtração seguida de atribuição</i>	<i>int x = 10; x-= 5; // x é 5</i>
++	<i>Adição própria</i>	<i>int x = 1; x++; // x é 2</i>
--	<i>Subtração própria</i>	<i>int x = 1; x--; // x é 0</i>



## 1.4. Operadores relacionais

<i><b>Operador</b></i>	<i><b>Descrição</b></i>	<i><b>Ex.:</b></i>
<code>==</code>	<i>Igual</i>	<i>if(x == 1)</i>
<code>!=</code> ou <code>&lt;&gt;</code>	<i>Diferente</i>	<i>if(x != 1) ou if(x &lt;&gt; 1)</i>
<code>&amp;&amp;</code>	<i>E</i>	<i>if(x != 1 &amp;&amp; x != 0)</i>
<code>//</code>	<i>Ou</i>	<i>if(x != 1 // x == 0)</i>
<code>&lt;</code>	<i>Menor</i>	<i>if(x &lt; 1)</i>
<code>&gt;</code>	<i>Maior</i>	<i>if(x &gt; 1)</i>
<code>&lt;=</code>	<i>Menor ou igual</i>	<i>if(x &lt;= 1)</i>
<code>&gt;=</code>	<i>Maior ou igual</i>	<i>if(x &lt;= 1)</i>

*Para comparação de “String”, é necessária a utilização de métodos específicos da própria classe, como o “equals”. Ex.:*

```
String x = "abc";

if (x.equals("abc")) {
    System.out.println("É abc.");
}
```

## 1.5. Comentários

*Os comentários em Java podem ser escritos em linha:*

```
// isto é um comentário
```

*Ou em blocos:*

```
/* este é um comentário em blocos
 * utilizado para comentários longos
 * ou geração de documentação automática */
```

*Os comentários em Java podem ser úteis para documentação automática de código (Javadoc). Para isso, deve ser respeitado um modelo de comentário em blocos e este adicionado antes de declarações de Classes, atributos e métodos. Ex.:*

```
/**
 *
 * Valida a inclusão de um pedido.
 *
 * @param numeroPedido Número do pedido a ser finalizado
 * @param valor Valor total to pedido
 * @param codigoCliente Referencia ao cliente do pedido
 * @param data Data do pedido
 *
 * @return verdadeiro se o pedido foi incluído e falso se houve erro
 *
 * @author João Primeiro
 * @author Zé Segundo
 */

boolean finalizaPedido (int numeroPedido, double valor, int
codigoCliente, Date data)
{
}
```

## 1.6. Pacotes, Classes e Objetos

*O conceito de Classes e Objetos define que uma Classe é define um conjunto de elementos com características em comum. Cada um desses elementos possui uma identidade própria, assim como suas características peculiares, sendo chamados de Objetos. Cada Classe possui, além das características (atributos), métodos, que representam o comportamento, as ações e funções realizadas pelos Objetos.*

*Objetos, então, são a interpretação em programação de um elemento do mundo real, sendo ele representante pertencente de uma Classe em específico. Cada objeto possui valores de características próprios, conforme o conjunto de características definido em sua Classe. Por exemplo, seguindo a análise do código abaixo:*

```
public class Pessoa {
    public String nome;
    public int idade;
    public float altura;
    public float peso;

    // construtor padrão
    public Pessoa(){}

    // construtor atribuindo "nome"
    public Pessoa(String nome) {
        this.nome = nome;
    }

    public void falar(){
        System.out.println("Olá, meu nome é " + this.nome + ".");
    }
}
```

*A classe “Pessoa” define um conjunto de atributos e ações em comum entre todos os representantes deste grupo (Objetos). Além disso, estão definidos dois métodos construtores. Para criar uma nova instância de objeto do tipo “Pessoa”, é utilizado o comando “**new**”. Este commando realiza uma chamada ao método construtor definido conforme os parâmetros passados a ele. Caso não sejam passados parâmetros, é realizada a chamada ao construtor padrão.*

```
public class Main {
    public static void main(String[] args) {
        Pessoa pessoal = new Pessoa();
        pessoal.nome = "Jão";

        pessoal.falar();
        // exibe no console "Olá, meu nome é Jão."
    }
}
```

*ou*

```
public class Main {
    public static void main(String[] args) {
        Pessoa pessoal = new Pessoa("Jão");

        pessoal.falar();
        // exibe no console "Olá, meu nome é Jão."
    }
}
```

*É importante observar que Java é case-sensitive, ou seja, há diferenciação de código entre letras minúsculas e maiúsculas. O padrão de escrita de código utilizado deve ser o adotado pela equipe de desenvolvimento do programa. Nesta apostila, foi adotada a convenção sugerida pela Sun:*

**Métodos e Atributos:** *iniciam com letra minúscula e palavras compostas utilizam letra maiúscula para troca. Não devem ser utilizados espaços nem acentos e caracteres especiais, além de não ser comum utilizar artigos e preposições. Ex.: nome, idade, nomePai, quantidadeItens, realizaCompra(), finalizaPedido();*

**Variáveis e Classes:** *todas palavras iniciam com letra maiúscula, inclusive na troca para palavras compostas. Não devem ser utilizados espaços nem acentos e caracteres especiais, além de não ser comum utilizar artigos e preposições. Ex.: Pessoa, Animal, Pedido, ItemPedido, CarinhoCompras.*

*Pacotes, por sua vez, são coleções de Classes e códigos responsáveis por conceitos em específico. Por exemplo, alguns dos Pacotes utilizados em Java são:*

**java.lang;** - inclui no código referências às Classes principais para codificação, como “String”, “Math”, “Thread”, “Object”, etc.;

**java.util;** - inclui no código referências às principais Classes auxiliares para codificação, como “Collection”, “List”, “Date”, etc.;

A chamada de inclusão de Pacotes é feita sempre no início do arquivo de código, nas primeiras linhas, podendo ser incluídas Classes e divisões específicas de um Pacote, ou o Pacote inteiro. Ex.:

```
import java.io.*; // importa todo o pacote "java.io";

import java.util.regex.*;
// importa toda a divisão "regex" do pacote "java.util";
```

Dentre as principais Classes Java pode-se considerar:

**String:** classe responsável pela construção e manipulação de textos. Seus principais métodos são:

- *charAt()*: Obtém o caracter localizado na posição especificada;
- *startsWith()*: informa se o texto inicia com o prefixo fornecido;
- *endsWith()*: informa se o texto termina com o sufixo fornecido;
- *equals()* : Compara conteúdo, caracter-a-caracter;
- *equalsIgnoreCase()* : Compara conteúdo, caracter-a-caracter ignorando o aspecto maiúsculo/minúsculo
- *indexOf()*: obtém a posição da primeira ocorrência do argumento;
- *intern()*: retorna uma referência para um objeto String correspondente armazenado no "pool";
- *lastIndexOf()*: obtém a posição da última ocorrência do argumento;
- *length()*: obtém a quantidade de caracteres;
- *substring()*: obtém o trecho desejado;
- *split()*: quebra o texto em pedaços;
- *toLowerCase()*: obtém o texto equivalente em letras minúsculas;
- *toUpperCase()*: obtém o texto equivalente em letras maiúsculas;
- *trim()*: obtém o texto equivalente sem espaços laterais;
- *valueOf()*: obtém o valor textual do dado de tipo primitivo fornecido.

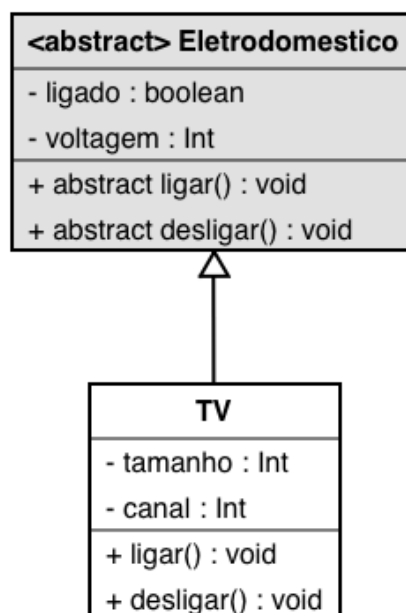
**Math:** classe responsável pela construção e manipulação de equações e cálculos matemáticos;

**Array:** principal classe responsável pela construção e manipulação de coleções de objetos (funcionamento semelhante a listas e vetores);

**Thread:** classe responsável pela construção e manipulação de processamentos paralelos;

**System:** classe responsável pela manipulação e exibição de comportamentos de sistema, como trabalhos com data e hora, propriedades de sistema, etc.;

Existe em Java, também, a abstração de Classes. Uma Classe Abstrata é uma classe que serve de modelo para as classes filhas na sua hierarquia, com estas herdando métodos e atributos da Abstrata pai. Por exemplo, pode-se ter uma Classe abstrata “Eletrrodomestico” com os atributos básicos em comum entre todos os eletrodomésticos, assim como seus métodos, e ter as classes respectivas a cada tipo de eletrodoméstico, como “TV”, contendo dados referentes aos mesmos. Classes abstratas não podem ser instanciadas diretamente (através do “new”). O mesmo vale para métodos abstratos, valem as declarações realizadas nas classes filhas.



```

public abstract class Eletrrodomestico {
    public boolean ligado;
    public int voltage;

    // obrigatoriamente implementados nas classes derivadas
  
```

```

        public abstract void ligar();
        public abstract void desligar();
    }

    public class TV extends Eletrodomestico {
        public int tamanho;
        public int canal;

        public void ligar(){
            this.ligado = true;}
        public void desligar(){
            this.ligado = false;}
    }

```

*Java conta, também, com **Interface**, um tipo de Classe que não possui conteúdo implementado, apenas chamadas de declarações. Interfaces são, em partes, uma solução encontrada para a emulação de herança múltipla, recurso não implementado em Java. Dessa forma, uma Classe Java pode herdar itens da Classe pai e ainda possuir itens de uma outra Classe modelo. A Interface mais comum utilizada em Java é a Runnable, utilizada em threads e que possui a chamada de método “run()”, implementado na classe implementada. Por exemplo:*

```

public class Relogio implements Runnable {
    // método de implementação obrigatória derivado do Runnable
    public void run(){
        // passa as horas
    }
}

```

*Por fim, existem ainda as Classes **Internas**, declaradas internamente a outras classes, com acesso a atributos e métodos da externa e não instanciáveis diretamente.*

*Por exemplo:*

```
public class Externa {  
    public class Interna {  
        public void mostraReferencia() {  
            System.out.println("A referência é " + this);  
        }  
    }  
}
```

```
public static void main(String [] args)  
{  
    Externa ex = new Externa();  
    Externa.Interna i = ex.new Interna();  
    i.mostraReferencia();  
}
```



## 1.7. Modificadores de acesso e encapsulamento

*Trabalhar com Classes na Orientação a Objetos torna-se mais interessante quando utiliza-se o encapsulamento. Esta técnica consiste em agrupar e proteger trechos de código a fim de adicionar maior integridade de dados e do programa, além de manter uma organização de código e dados conforme padrões de desenvolvimento.*

*O exemplo mais comum de encapsulamento em Java é a utilização de métodos “get” e “set” para acesso a atributos com modificadores não públicos. Os principais modificadores de atributos são:*

***public:*** utilizado de forma livre para atribuição ou chamada pelas instâncias da Classe;

***protected:*** tem atribuição ou chamada realizadas apenas nas classes derivadas;

***private:*** tem atribuição ou chamada realizadas apenas na instância da própria classe ou através da utilização de métodos “get” e “set”;

***static:*** define um único armazenamento em memória de um valor ou chamada de método, não podendo ser alterado;

***final:*** define o primeiro valor atribuído como o ultimo, não podendo ser alterado, mais utilizado como definição de constantes;

```
public class Pessoa {
    public String nome;
    private float peso;

    // construtor padrão
    public Pessoa(){}

    // construtor atribuindo "nome"
    public Pessoa(String nome) {
        this.nome = nome;
    }

    public void setPeso(float peso) {
        this.peso = peso;
    }

    public float getPeso(){
```

```
        return this.peso;}
    }
    public class Main {
        public static void main(String[] args) {
            Pessoa pessoal = new Pessoa();
            pessoal.nome = "Jão";
            //pessoal.peso = 70; - não funciona
            pessoal.setPeso(70); // funciona

            System.out.println("Olá, meu peso é " +
pessoal.getPeso().toString() + ".");
        }
    }
```

## 1.8. Coleções

<i>Tipo de Coleção</i>	<i>Classe</i>	<i>Tipo de Método</i>	<i>Método</i>
<i>Genérico</i>	<i>Collection</i>	<i>Adição</i>	<i>add(Object elemento)</i>
		<i>Remoção</i>	<i>remove(Object elemento)</i>
			<i>clear()</i>
		<i>Acesso</i>	<i>iterator()</i>
		<i>Pesquisa</i>	<i>contains(Object elemento)</i>
		<i>Atributos</i>	<i>size()</i>
<i>Lista</i>	<i>Vector</i> <i>ArrayList</i> <i>LinkedList</i>	<i>Adição</i>	<i>add(int index, Object elemento)</i>
		<i>Remoção</i>	<i>remove(int index)</i>
		<i>Acesso</i>	<i>get(index)</i>
			<i>listIterator</i> <i>(iterador que pode andar para trás)</i>
		<i>Pesquisa</i>	<i>indexOf(Object elemento)</i>
<i>Conjunto</i>	<i>HashSet</i> <i>TreeSet</i>		<i>Nenhum método a mais</i>
<i>Mapa</i>	<i>HashMap</i> <i>TreeMap</i>	<i>Adição</i>	<i>put(Object key, Object value)</i>
		<i>Remoção</i>	<i>remove(Object key)</i>
			<i>clear()</i>
		<i>Acesso</i>	<i>get(Object key)</i> <i>(Isso também é "pesquisa")</i>
			<i>Set entrySet()</i> <i>(retorna os itens como conjunto)</i>
			<i>Set keySet()</i> <i>(retorna as chaves como conjunto)</i>
			<i>Collection values()</i> <i>(retorna os valores como Collection)</i>
		<i>Pesquisa</i>	<i>get(Object key)</i> <i>(Isso também é "acesso")</i>

*Existem alguns tipos de coleções em Java que se aplicam melhor a situações específicas. Pode-se destacar as Listas, os Conjuntos e os Mapas.*

**Listas:** *uma lista é uma arrumação de elementos em uma ordem linear, podendo os elementos serem facilmente ordenados e reorganizados. Pode ser utilizado em formato de vetor ou de lista encadeada.*

**Conjuntos:** são arrumações de elementos em forma não linear e não ordenadas, porém sem elementos duplicados. As formas mais comuns de utilização são como tabelas hash ou árvores.

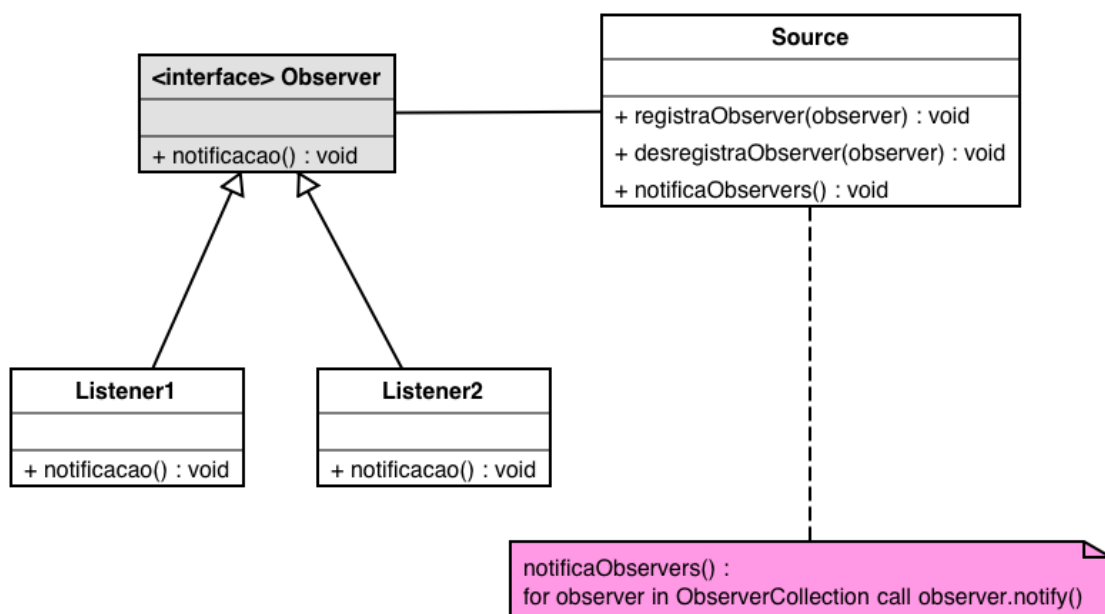
**Mapas:** são arrumações de elementos em forma de pares (chave e valor), chamados itens. O grande destaque dos Mapas em Java é que tanto os valores quanto as chaves podem ser de qualquer tipo. Mapas são normalmente utilizados para pesquisa rápida em grande quantidade de dados.

## 1.9. Padrão de projeto Observer, Eventos e Listeners

A utilização do padrão “Observer” tem como objetivo realizar a ”conversação” de objetos em tempo real (tempo de execução) sem a necessidade de prever tal comportamento em tempo de compilação. Para isso, é necessária a utilização de alguns elementos em Java.

**Source:** é o Objeto responsável pela alteração a ser notificada para os Objetos interessados (listener);

**Listener:** são os objetos interessados nas modificações ocorridas no Objeto de interesse (source).



Como exemplo, identificar uma caixa de e-mails. Imagina-se que um remetente deseja saber quando seu e-mail chegou na caixa de entrada do destinatário e este deseja ser avisado quando há novos e-mails não lidos. Identifica-se um “Source” (e-mail) e dois “Listeners” (remetente e destinatário). Primeiramente, deve ser criada uma Classe para o evento de modificação.

```
public class EmailEvent extends EventObject {
    public EmailEvent(Object source){
        super(source);
    }
}
```

*Em seguida, é necessária a criação de uma Classe que irá receber as notificações de alteração e será implementada pelos interessados, no caso remetente e destinatário.*

```
public interface EmailListener {
    public void emailRecebido (EmailEvent emailEvent){}
    public void emailLido (EmailEvent emailEvent){}
}
```

*Depois, é criada a Classe do Objeto a ser modificado.*

```
public class Email {
    private Collection<EmailListener> emailListener = new
    ArrayList<EmailListener>();

    public void enviar(){
        disparaEmailRecebido();
        disparaEmailLido();
    }

    public void emailRecebido(){
        this.disparaEmailRecebido();
    }

    public void emailLido(){
        this.disparaEmailLido();
    }

    public synchronized void addEmailListener(EmailListener
emailListener) {
        this.emailListeners.add(emailListener);
    }

    public synchronized void removeEmailListener(EmailListener
emailListener) {
        this.emailListeners.remove(emailListener);
    }
    private void disparaEmailLido(){
```

```

        EmailEvent event = new EmailEvent(this);
        for(EmailListener listener : emailListeners) {
            listener.emailLido(event);
        }
    }

    private void disparaEmailRecebido(){
        EmailEvent event = new EmailEvent(this);
        for(EmailListener listener : emailListeners) {
            listener.emailRecebido(event);
        }
    }
}

```

*Em seguida, as classes responsáveis pelo remetente e pelo destinatário devem implementar a classe de “escuta” das modificações.*

```

public class Destinatario implements EmailListener {
    public void emailLido (EmailEvent emailEvent){
        Email email = (Email) emailEvent.getSource();
        System.out.println("Você acabou de ler o e-mail!");
    }

    public void emailRecebido (EmailEvent emailEvent){
        Email email = (Email) emailEvent.getSource();
        System.out.println("Você acabou de receber o e-mail!");
    }
}

```

```

public class Remetente implements EmailListener {
    public void emailLido (EmailEvent emailEvent){
        Email email = (Email) emailEvent.getSource();
        System.out.println("O e-mail " + email + " acabou de ser
lido!");
    }

    public void emailRecebido (EmailEvent emailEvent){
        Email email = (Email) emailEvent.getSource();
        System.out.println(("O e-mail " + email + " acabou de ser
recebido!"));
    }
}

```

*Para simular a execução, cria-se uma classe gerenciadora da caixa de entrada.*

```
public class CaixaEntrada {
    public static void main(String[] args){
        Email email1 = new Email();
        Email email2 = new Email();

        Destinatario destinatario = new Destinatario();
        Remetente remetente = new Remetente();

        email1.addEmailListener(destinatario);
        email1.addEmailListener(remetente);
        email2.addEmailListener(destinatario);
        email2.addEmailListener(remetente);

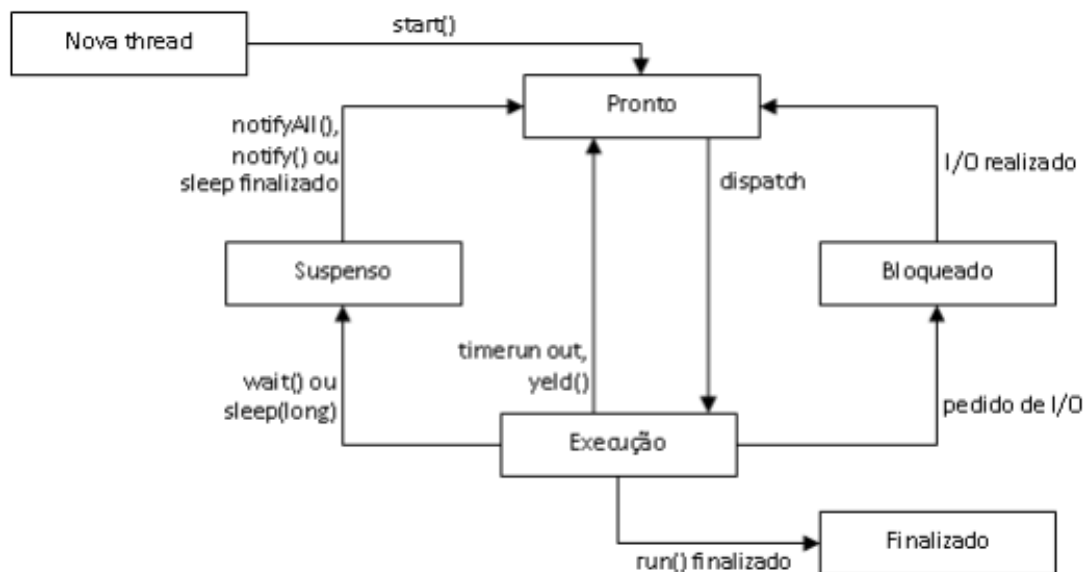
        // ações de modificação a serem notificadas aos
interessados
        email1.enviar();
        email2.emailRecebido();
    }
}
```

*O tratamento de **Eventos** em Java funciona de maneira semelhante. Através de *Listeners*, ou seja, *handlers* que ficam esperando por notificações de evento, estes são disparados e executam as ações previstas no código.*



## 1.10. Threads e Exceções

*Um dos grandes diferenciais do Java é a qualidade do trabalho com programação concorrente, as “Threads”. Em Java é muito simples criar concorrência com bom gerenciamento de memória e rápido processamento.*



*Basicamente, existem duas formas de se trabalhar com “Threads” em Java.*

*A primeira, consiste em criar uma Classe filha da Classe nativa “Thread”. Em seguida, devem ser criados métodos (ou utilizados comandos avulsos) a serem chamados dentro do método derivado “run()”.*

*A segunda forma consiste em, ao invés de estender a Classe nativa “Thread”, implementar a Interface nativa “Runnable”. Desta forma, é permitido que seja realizado um trabalho com Classes derivadas de uma hierarquia e estas possuam trabalhos concorrentes (lembrando que Java não possui herança múltipla).*

*A grande diferença de codificação é que, após instanciar uma “Runnable”, deve ser criado um objeto do tipo “Thread” para execução, passando como parâmetro a referência àquela.*

```

public class ExemploRunnable implements Runnable {
    public void run() {
        int total = 0;

        while ((int)(Math.random()*100) != 50){
            total++;
        }
    }
}

public class Main {
    public static void main(String[] args) {
        ExemploRunnable r = new ExemploRunnable();

        Thread t = new Thread(r, "Thread 1");

        t.start();
    }
}

```

*Dessa forma, a API Java referente a “Threads” se encarregará de fazer a execução concorrente a partir de chamadas como “start()”, “stop()”, “interrupt()”, “suspend()”, “yield()” e “resume()”.*

*O tratamento de **Exceções** em Java também é realizado de forma simples. Baseado no conceito “try-catch”, onde um comando executado com erro dispara uma informação de exceção e esta é recuperada por um trecho de código, é possível criar tratamentos de exceção específicos para cada caso ou utilizar os tratamentos genéricos.*

```

public class Calculadora {
    public int divide(int a, int b) throws DivisaoZero {
        if (b == 0) throw new DivisaoZero();
        return a/b;
    }
}

```

*No exemplo acima, foi implementada a função de divisão de uma calculadora. No método responsável pelo cálculo, é testado se o segundo parâmetro é zero e, caso positivo, dispara a exceção “DivisaoZero”. Esta, é uma Classe personalizada, que deriva da Classe nativa “Exception” e sobrescreve o método “toString()”.*

```
public class DivisaoZero extends Exception {  
    public String toString() {  
        return "Divisão por zero.";  
    }  
}
```

*Para utilizar esta exceção, basta realizar a chamada ao método dentro do bloco “try” e o código de tratamento dentro do bloco “catch”.*

```
...  
Calc calc = new Calc();  
try {  
    int div = calc.div(x, y);  
    System.out.println(div);  
} catch (DivisaoZero e) {  
    System.out.println(e);  
}  
...
```

*Java dispõe, também, do comando “finally”, responsável por executar o bloco mesmo após o tratamento da exceção.*

## 2. Introdução ao sistema operacional Android

*A Android Inc. é uma empresa adquirida pela **Google** em 2005. Naquela época, a empresa possuía a ideia em papel de um sistema operacional para dispositivos móveis, posteriormente realizado pela compradora.*

*Android é o Sistema Operacional para smartphones e tablets colocado em prática e continuado pela Google, em comunidade com a Open Handset Alliance, conjunto de mais de trinta empresas, dentre elas operadoras de telefonia e fabricantes de dispositivos. Além disso, a Google fornece uma biblioteca gratuita para desenvolvimento de aplicativos Android em Java, cuja comunidade é muito grande, o que possibilita uma grande variedade de aplicativos no Google Play. Esta, por sua vez, é a loja e distribuidora oficial de aplicativos Android, também gerenciada pela Google.*

*Em dezembro de 2011, o Android chegou à incrível marca de mais de 700 mil aparelhos ativados por dia, ou seja, está em uma crescente de utilizadores, números que devem aumentar a partir da finalização da compra parcial da Motorola (uma das principais fabricantes de aparelhos com Android) pela Google, iniciada em agosto de 2011. Esta presença de uma comunidade forte e o crescente interesse das fabricantes no sistema operacional é o grande diferencial do Android contra os fortes concorrentes da atualidade (iOS e Windows Phone). Mais aplicativos diferenciados, interfaces customizadas por operadora e fabricante agradam novos consumidores e até mesmo os “hardcore”.*

*Tecnicamente falando, Android é construído sobre o núcleo Linux. Por ser uma plataforma em código aberto, possui distribuições em kernel diferentes, o que deixa o sistema operacional mais personalizável. Além disso, é um sistema muito robusto, possuindo recursos nativos como a máquina virtual Dalvik (baseada na JVM), banco de dados SQLite, touch screen, GPS, gráficos 3D, acelerômetro, giroscópio e outros.*



*Até dezembro de 2011, foram desenvolvidas 7 versões de Android, tendo o percentual de ativações conforme o quadro a seguir:*

<i>1.5 Cupcake</i>	<i>1,1%</i>
<i>1.6 Donut</i>	<i>1,4%</i>
<i>2.1 Eclair</i>	<i>11,7%</i>
<i>2.2 Froyo</i>	<i>45,3%</i>
<i>2.3.x Gingerbread</i>	<i>38,7%</i>
<i>3.x.x Honeycomb</i>	<i>1,8%</i>
<i>4.0 Ice Cream Sandwich</i>	<i>0%</i>

*As versões 3 e 4 são oficialmente licenciadas para tablets.*

*O mercado tende à mobilidade e esta age cada vez mais integrada com a web, com produtos e serviços de empresas, atuando como um mediador e uma nova ferramenta de comunicação. É cada vez mais comum aplicativos com interfaces gráficas melhores, compatíveis com diversos dispositivos com recursos de tela diferentes, e a arte para aplicações móveis é uma das áreas de tecnologia e comunicação que mais crescem.*

## 2.1. Desenvolvimento Android

*Por ser uma plataforma de código aberto codificada em C++ e Java, Android possui bibliotecas de desenvolvimento oficiais nessas linguagens e outras formas de desenvolvimento criadas pela comunidade. A grande parte dos aplicativos Android hoje são desenvolvidos em Java através do **SDK** (Standard Development Kit) e do plugin oficial para o IDE Eclipse, o ADT, fornecidos gratuitamente pela Google.*

*<http://developer.android.com/sdk/index.html>*

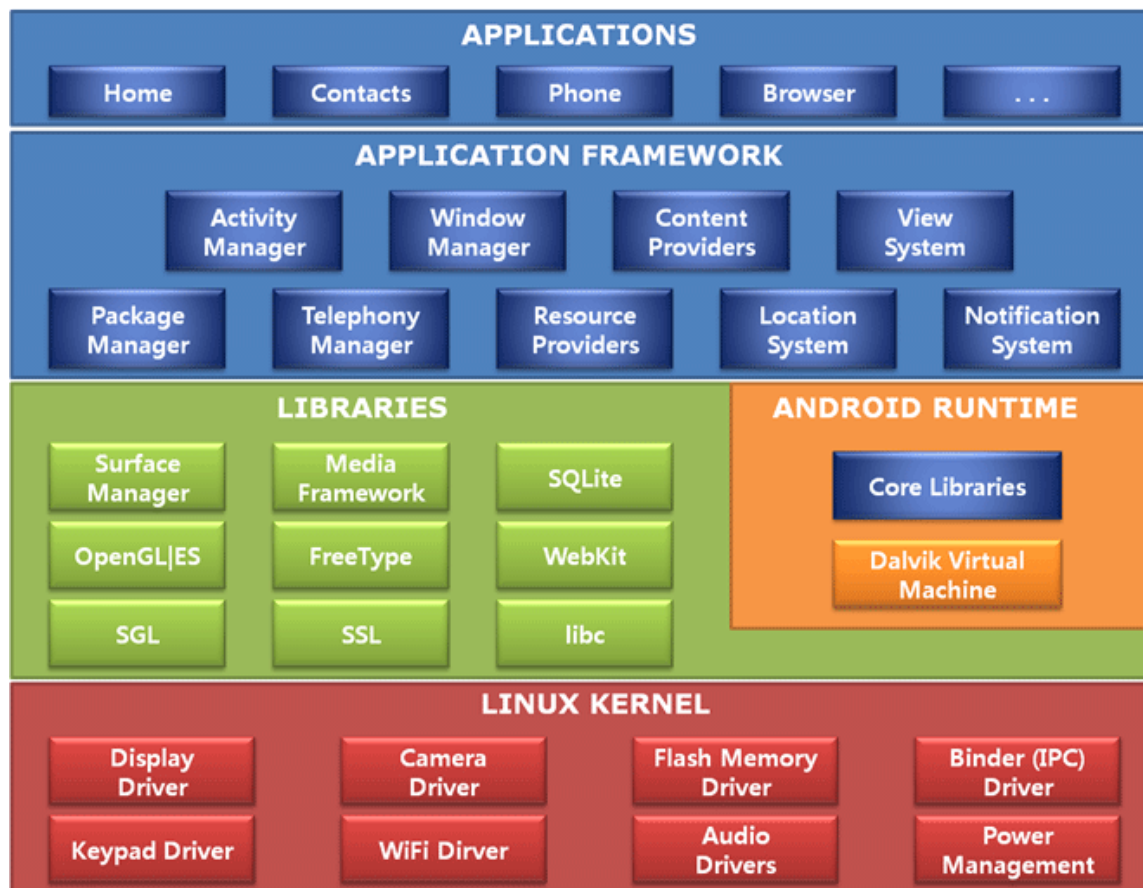
*Apesar disso, a Google oferece, ainda, o **NDK** (Native Development Kit), biblioteca de desenvolvimento em código nativo (C/C++), utilizado para desenvolver aplicativos que requerem maior manipulação dos recursos fornecidos pela GPU Android e que não são suportados no SDK (como OpenGL).*

*<http://developer.android.com/sdk/ndk/index.html>*

*Aplicativos desenvolvidos para Android são compilados e convertidos em um pacote **APK** (Android Application Package). Este pacote é um arquivo compactado (como ZIP) baseado no formato JAR (Java Archive, pacote de execução Java), que possui arquivos de referência, de estrutura e arquivos DEX (Dalvik Executable), capazes de serem executados pela máquina virtual do Android (a Dalvik).*

*O Android possui, ainda, um navegador de Internet nativo, construído em cima da plataforma Webkit (Chrome, Safari). Por isso, é possível construir aplicativos em **HTML5**, executados diretamente no navegador. Existem alguns frameworks, como o PhoneGap que utilizam tags HTML e as converte em elementos gráficos do próprio SDK, tornando, assim, o desenvolvimento HTML para Android mais robusto e compatível com o Google Play (o framework é capaz de gerar um APK com um elemento navegador web interno na aplicação). No entanto, este desenvolvimento ainda não é homologado pela Google e não é oferecido suporte pela mesma.*

*Nos links acima, está incluída a documentação completa do SDK Android, que é usado como base para o estudo nesta apostila.*



## 2.2. Preparação do Ambiente de Desenvolvimento

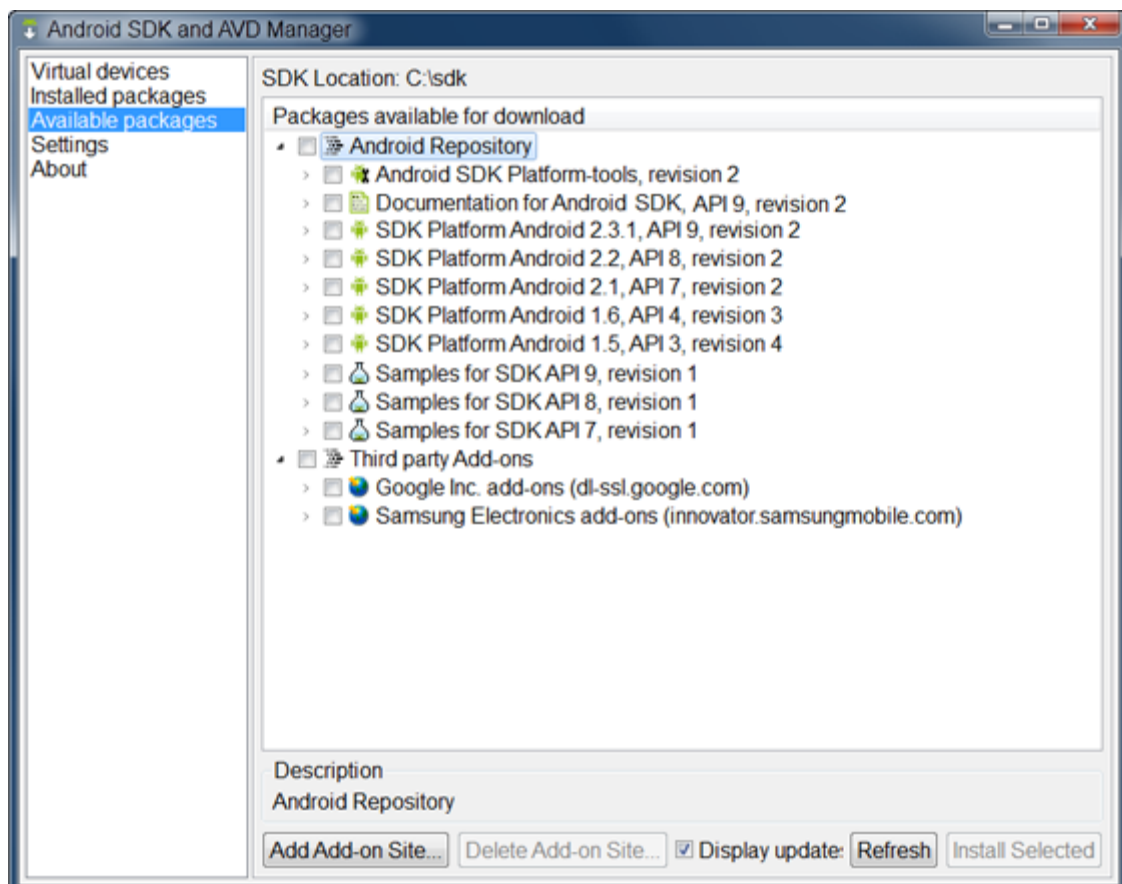
*Para iniciar o desenvolvimento Android em Java em ambiente Windows são necessárias três ferramentas. É importante observar que, ao atualizar cada ferramenta, é pode ser necessário atualizar as demais.*

*IDE Eclipse: <http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/indigor>*

*Android SDK: <http://developer.android.com/sdk/index.html>*

*Plugin ADT: <http://developer.android.com/sdk/eclipse-adt.html>*

*Primeiramente, é necessário instalar o SDK. Após a instalação, é necessário entrar na pasta onde foi instalado e executar o “SDK Manager.exe”.*



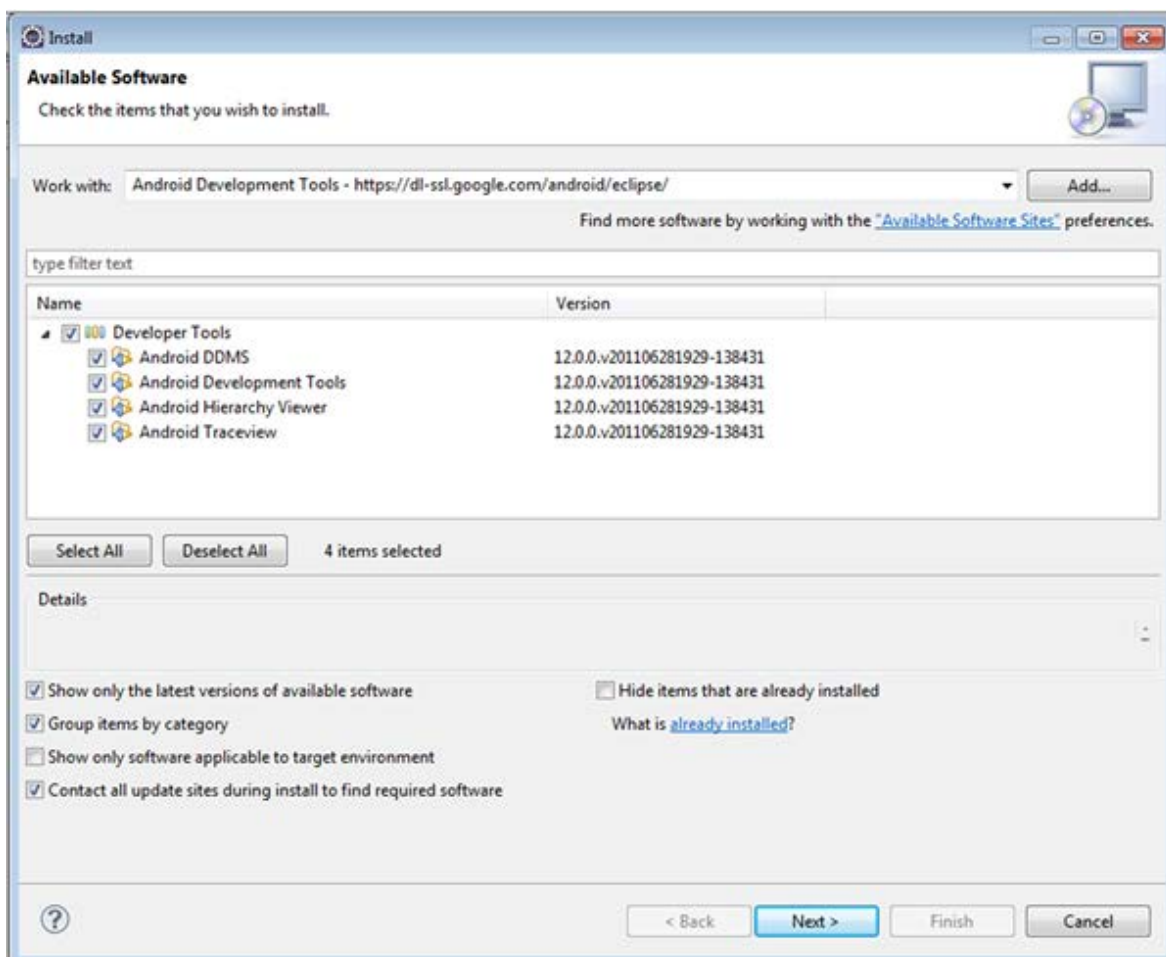


*Nesta tela é possível escolher quais pacotes serão baixados e instalados. É recomendável baixar todos, pois em momentos específicos pode ser necessário desenvolver aplicações para aparelhos mais antigos, que não suportam as API novas. Além do que, existem pacotes de exemplos e add-ons de outros fornecedores.*

*Após esta instalação é possível, no menu “Tools -> Manage AVDs”, configurar um emulador, selecionando opções de configuração como tipo e tamanho de tela e outros componentes hardware, ou selecionando um emulador pré-definido.*

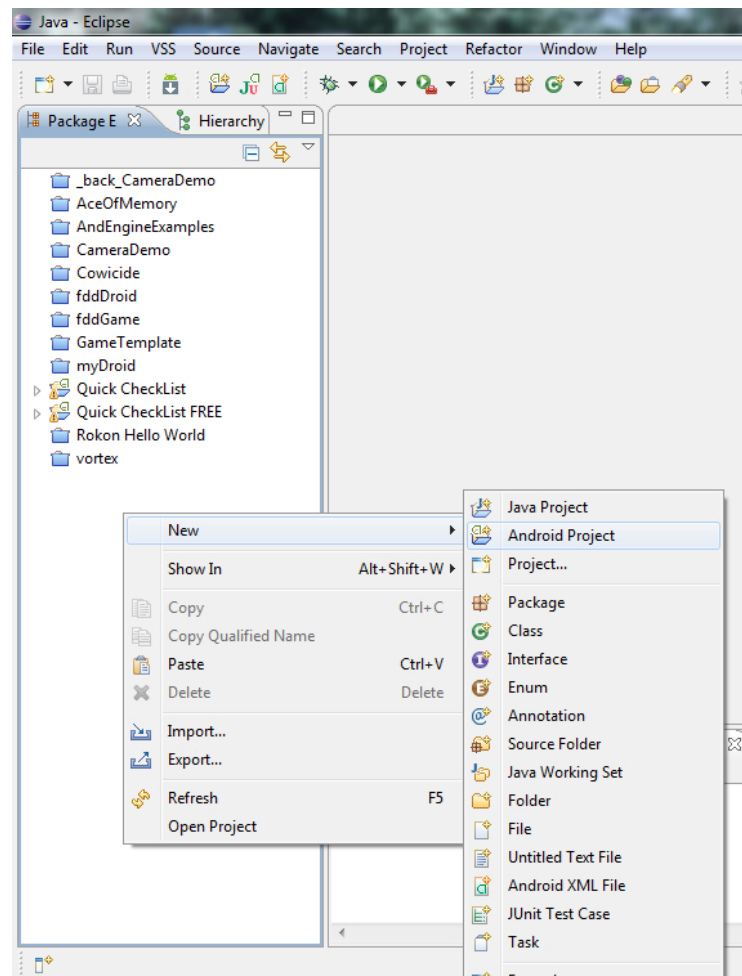
*No Eclipse, no menu de ajuda, existe a opção de instalar novos recursos. Basta inserir o link para procura disponibilizado pela Google e instalar o pacote completo.*

*<https://dl-ssl.google.com/android/eclipse/>*



## 2.3. HelloWorld!

*Como primeiro projeto e para ambientação no Eclipse, cria-se o projeto padrão “Hello World”. Na área esquerda do ambiente existe o explorador de projetos. Basta abrir o menu flutuante do projeto e selecionar a opção “New -> Android Project”.*



*Em seguida, coloca-se o nome do projeto (no caso “HelloWorld”) e seleciona-se, em “Build Target” a versão de API a ser utilizada (no caso, mais atual). É interessante observar que para cada versão de API existe uma opção básica e uma opção “Google API”, que inclui códigos de integração entre serviços Google.*

*Na seção “Properties”, é necessário preencher o nome da aplicação (“HelloWorld”), o nome do pacote (“hello.world”), a principal “Activity” (“HelloWorldActivity”) e a versão mínima de API (a mesma selecionada no “Build Target”, como por exemplo “8”). Por fim, basta finalizar a criação.*

Conforme configurado, o próprio SDK se encarrega de criar a atividade principal (“HelloWorld Activity”), cujo código pode-se conferir em “src -> hello -> world -> HelloWorldActivity.java”.



```
package hello.world;

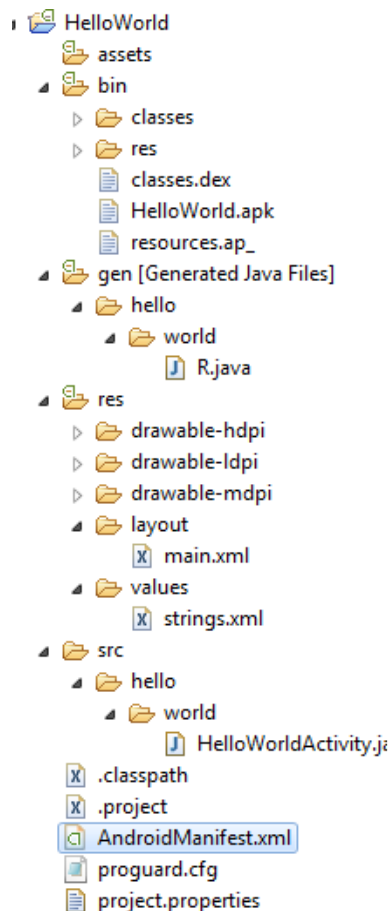
import android.app.Activity;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Uma “Activity” (atividade) é, basicamente, uma Classe com escopo de execução. Possui o método “onCreate” que é responsável por executar o primeiro código da Classe. Dessa forma, quando define-se uma “Activity” principal de início, executa-se sua criação em primeiro lugar.

## 2.4. Estrutura de um projeto e XML

*Conforme pode-se analisar no navegador de projetos do Eclipse, uma aplicação Java para Android é dividida em diversos componentes. Cada pasta e sub-pacotes do projeto é responsável por alguma função na aplicação e pode-se destacar alguns elementos:*



**assets:** é a pasta utilizada para armazenamento de mídias diversas, como imagens, vídeos, músicas, que não são utilizadas como elementos de layout, mas de conteúdo;

**bin:** pasta onde fica armazenado o código compilado das classes Java;

**gen:** pasta de arquivos gerados automaticamente pelo SDK. Esta pasta, assim como seus arquivos, não deve ser modificada, pois qualquer alteração pode comprometer a integridade do código gerado automaticamente. A Classe “**R**” é muito importante, pois ela armazena referências hexadecimais a todos componentes globais da aplicação, como elementos de layout;

**res:** pasta de recursos. Possui subpastas “drawable-\*” responsável por armazenar as imagens para elementos de tela (layout) em diversas resoluções (“ldpi”, “mdpi” e “hdpi” referenciam baixa, média e alta qualidade, respectivamente). A pasta “layout” possui os arquivos **XML** de layout, onde cada um destes arquivos é formado por tags que representam os elementos de tela (como campos de texto, botões e grades de alinhamento) e são utilizados em Classes “Activity”. Por fim, há a pasta “values”, que por sua vez possui arquivos XML diversos, mais utilizados para definições de constantes globais, como o “strings.xml”;

**src:** pasta responsável por armazenar todas as classes do projeto, podendo ser separadas por pacotes secundários ou não.

Existe, ainda, o arquivo “AndroidManifest.xml”, responsável pelas principais definições da aplicação, como número da versão de código, versão de SDK, referências a ícone e nome da aplicação, permissões de acesso e recursos necessários, declaração de atividades, serviços e demais componentes da aplicação, etc.

Para melhor interpretação da estrutura de arquivos de uma aplicação Java para Android, é interessante conhecer o funcionamento da linguagem XML. Extensible Markup Language (Língua de Marcação Extensível) é um recurso recomendado pela W3C (consórcio internacional regulamentador da web) para manipulação de dados de forma específica. Com o XML, é possível montar estruturas de dados baseadas em tags, realizando agrupamentos de tags e adicionando atributos e valores a elas. Dessa forma, é possível compartilhar dados específicos, com uma quantidade ilimitada de tags diferentes, em uma linguagem compreensível tanto por computadores quanto por humanos.

#### Exemplo de código XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<receita nome="pão" tempo_de_preparo="5 minutos"
tempo_de_cozimento="1 hora">
    <titulo>Pão simples</titulo>
    <ingredientes>
        <ingrediente quantidade="3"
unidade="xícaras">Farinha</ingrediente>
        <ingrediente quantidade="7"
unidade="gramas">Fermento</ingrediente>
        <ingrediente quantidade="1.5" unidade="xícaras"
estado="morna">Água</ingrediente>
        <ingrediente quantidade="1" unidade="colheres de
chá">Sal</ingrediente>
```

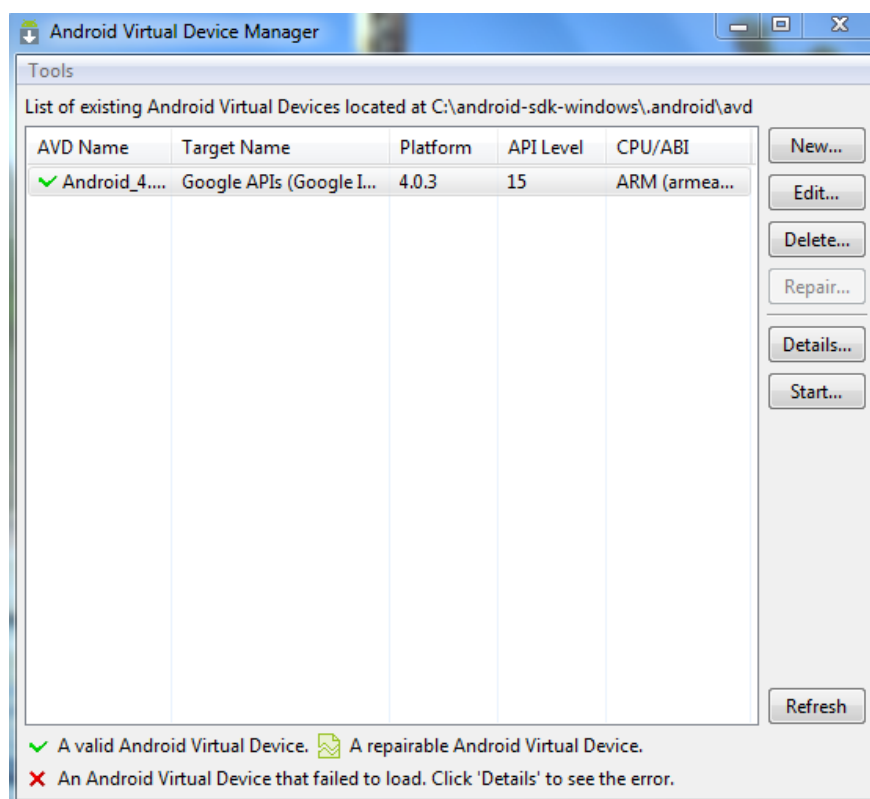
```
        </ingredientes>
        <instrucoes>
            <passo>Misture todos os ingredientes, e dissolva
bem.</passo>
            <passo>Cubra com um pano e deixe por uma hora em um local
morno.</passo>
            <passo>Misture novamente, coloque numa bandeja e asse num
forno.</passo>
        </instrucoes>
    </receita>
```

*A partir de um arquivo XML é possível construir um interpretador em diversas linguagens, muitas das quais possuem API com suporte específico a leitura deste tipo de arquivo. Assim, diversas aplicações podem partilhar de uma mesma fonte de dados. O grande exemplo é a utilização de arquivos XML em “WebServices” (recursos de compartilhamento de informações e integrações entre serviços na web), como o fornecimento de dados do Twitter.*

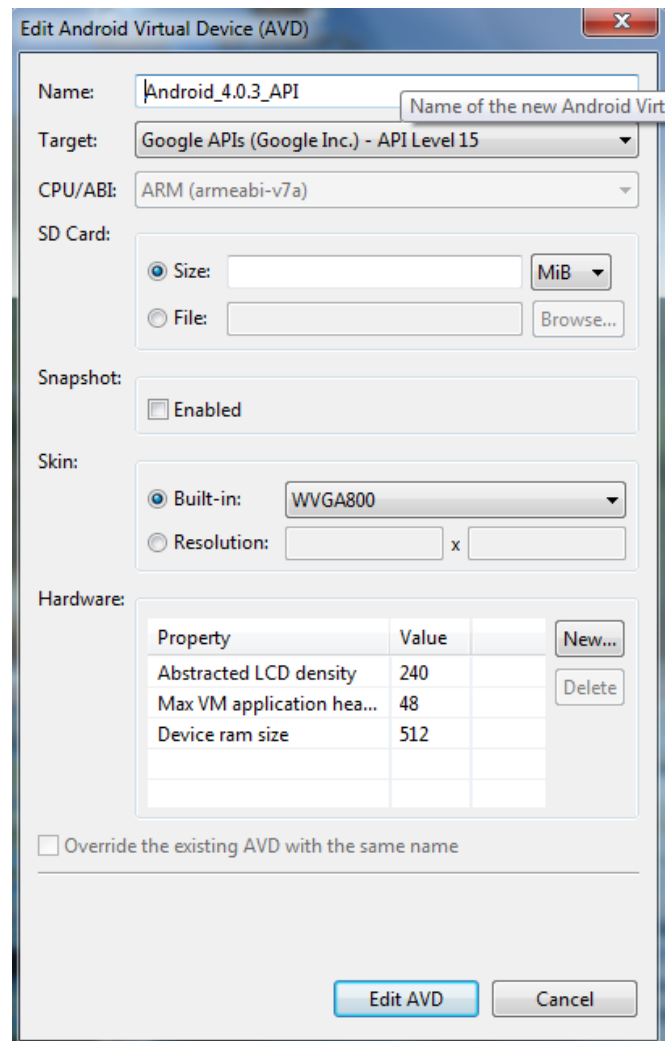
## 2.5. Compilação, execução e depuração

*O SDK Android conta com um emulador de plataformas Android, chamado de AVD (Android Virtual Device). No diretório de instalação do SDK existe um programa chamado “AVD Manager.exe”, onde é possível configurar diversos emuladores, para as diversas versões do sistema Android.*

*Na tela principal, é possível visualizar, modificar, remover ou inicializar um dispositivo.*



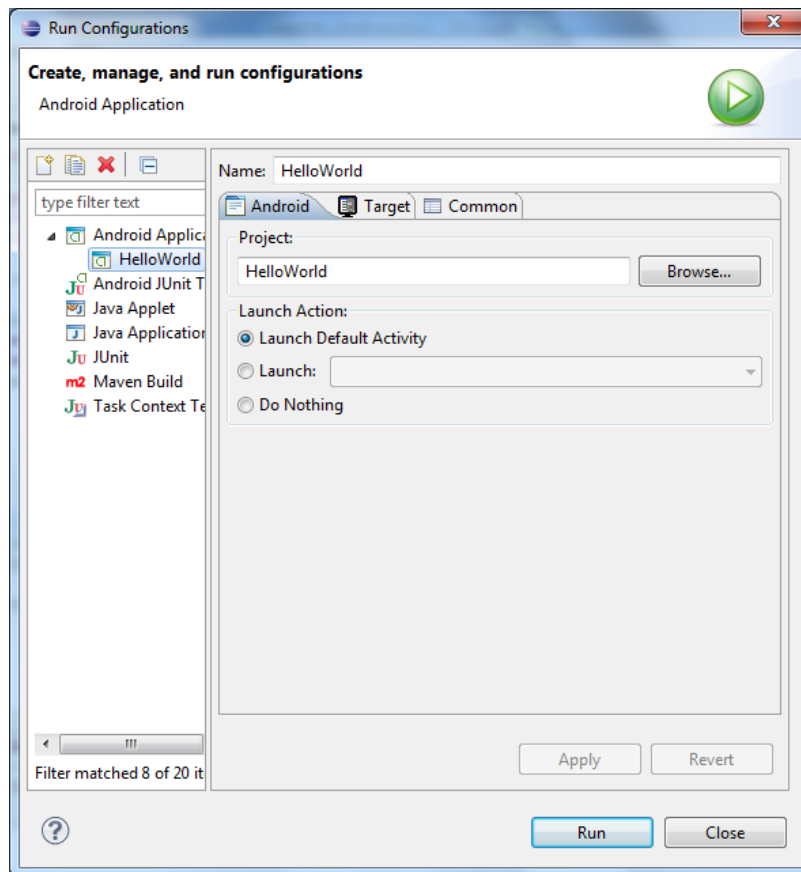
*Nas opções de criação ou modificação de um dispositivo existente, é possível configurar opções bastante flexíveis para emulação, como tamanho da tela, tamanho de espaço para o cartão SD (cartão de memória), modelo do processador, etc.*



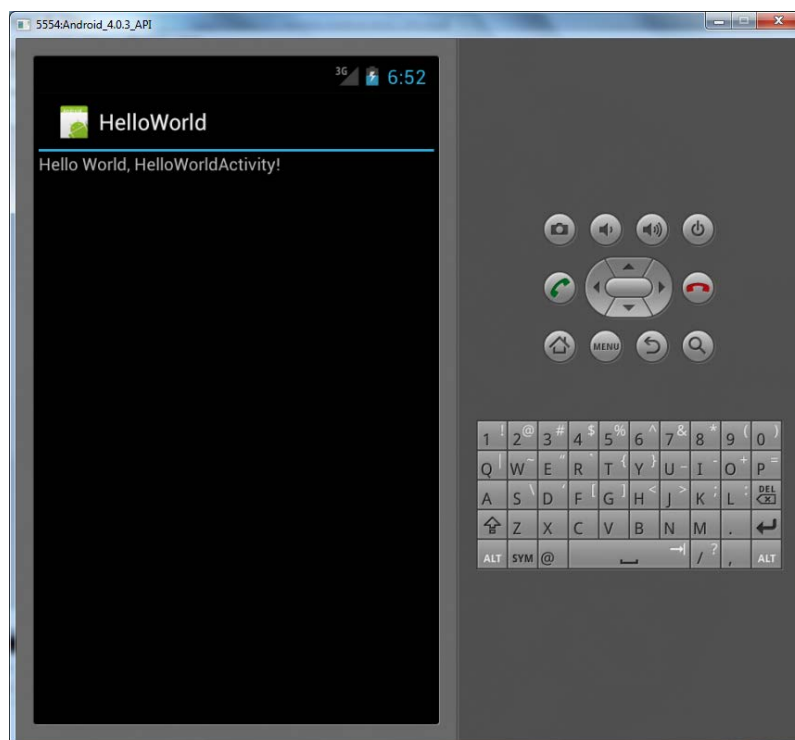
*Assim, é possível emular praticamente qualquer dispositivo lançado oficialmente no mercado, tanto smartphones quanto tablets.*

*No ambiente Eclipse, é possível compilar e executar uma aplicação Android diretamente nos emuladores criados pelo AVD Manager. No menu flutuante do projeto, em “Run As -> Run Configurations...” é possível gerenciar a execução da aplicação. Na opção de criar uma nova configuração, é possível selecionar na aba “Android” o projeto a ser executado e a “Activity” a ser executada (inicialmente é selecionada a “Activity” padrão, definida no “AndroidManifest.xml” ou na criação do projeto). Na aba “Target”, é selecionado qual o dispositivo emulado a ser executado e algumas opções para ele, como a alocação de banda de Internet disponível para ele. O ideal é manter o padrão para selecionar o dispositivo manualmente no momento da execução do app (aplicativo). Por fim, na aba “Commons” existem configurações diversas, como o endereço de armazenamento da configuração, configurações de I/O e execução.*





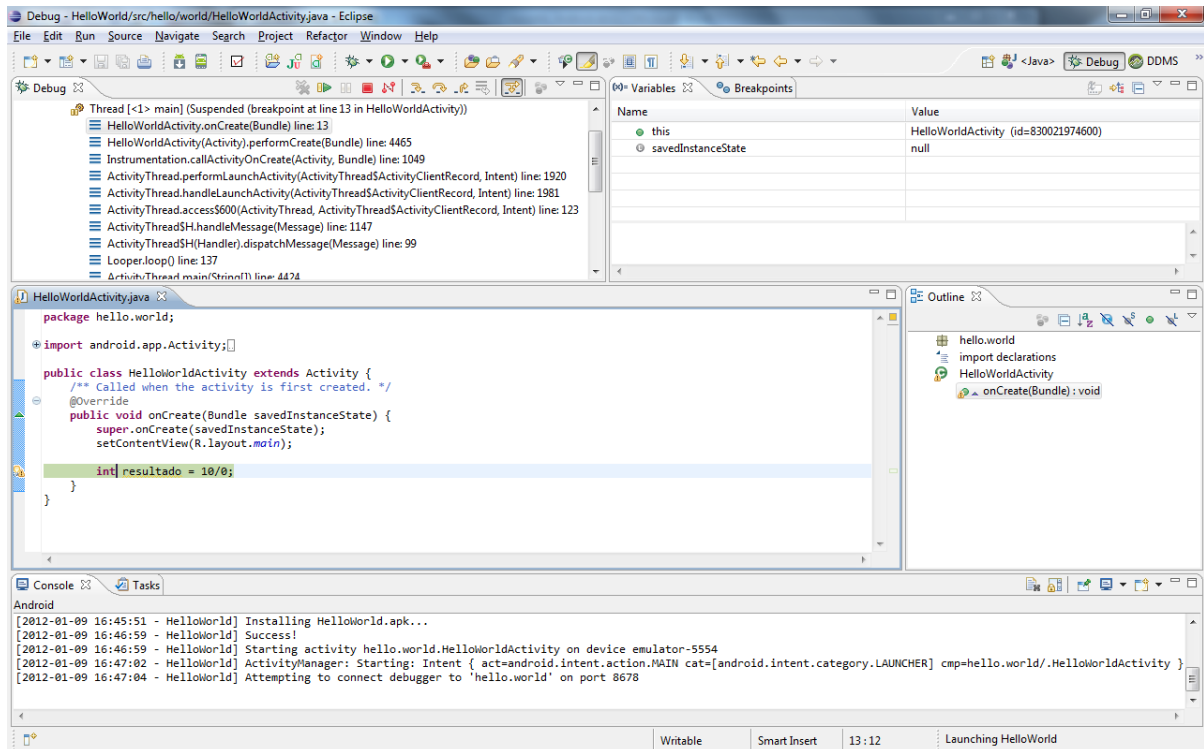
*Feito isso, basta utilizar a opção “Run” ou finalizar e selecionar a partir do menu suspenso, em “Run As”, a configuração recém criada.*



*Além do dispositivo emulado, pode-se trabalhar com um dispositivo real conectado via porta USB. Para isso, o dispositivo deve estar configurado com a opção “Depuração USB” ativada e ele aparecerá na tela de seleção de destino de execução (mesma dos emuladores) do app.*

*O projeto é compilado automaticamente no momento da execução e o IDE acusará no console uma informação de erro no código, quando houver. Para apenas compilação manual, é necessário utilizar a opção do menu principal “Project -> Build Project”. É interessante observar que, quando ocorrem muitas inclusões de arquivos internos e modificações de código seguidas sem compilação, pode ocorrer uma má formação dos arquivos gerados automaticamente. Para corrigir isso, basta utilizar a opção do menu principal “Project -> Clean...” e, logo após, executar ou utilizar o Build no projeto.*

*O processo de depuração é muito semelhante ao de execução. Ao invés de utilizar o menu “Run As”, utiliza-se o “Debug As”. O processo de depuração monta a máquina virtual com opções para execução passo-a-passo, o que ajuda muito no diagnóstico de problemas runtime (em tempo de execução). Os erros de código são filtrados em tempo de compilação, ao realizar o Build do projeto, porém os erros lógicos não tratados podem gerar falhas em tempo real de execução (como o caso da divisão por zero). Neste caso, pode-se observar no console a linha de código responsável pela mensagem de falha. Com o cursor marcado nesta linha de código da Classe específica, pode-se utilizar os comandos de debug, encontrados no menu principal “Run -> Toggle Breakpoint”. Ao executar a aplicação, na parte referente ao trecho de código sinalizado, a máquina virtual realizar uma pausa e o IDE redireciona para o ambiente de depuração, onde é possível realizar alterações ou sinalizar novas paradas.*



Neste ambiente, é possível analisar os processos realizados pela máquina virtual na interpretação do programa, analisar variáveis e propriedades atribuídas da mesma maneira que em um teste de mesa, visualizar o console e visualizar passo a passo a interpretação do código.

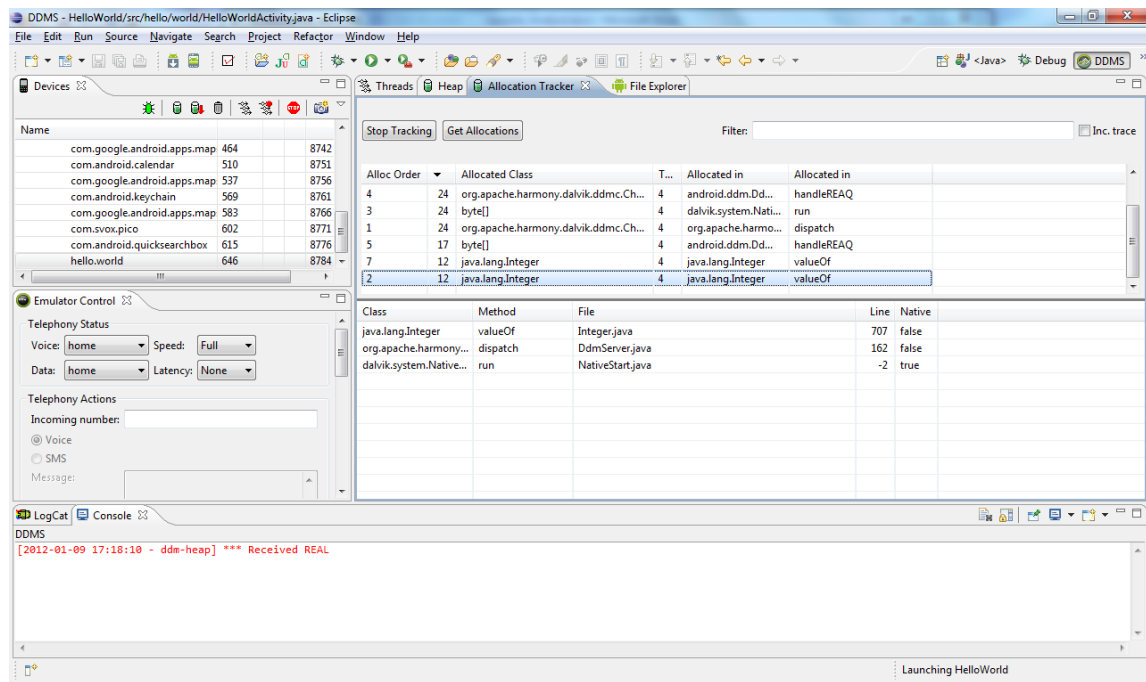
Os principais comandos de depuração são encontrados no menu principal “Run”:

**Step Into:** avança para a primeira linha de processamento após a linha atual, considerando a entrada em métodos, ou seja, caso a linha atual seja uma chamada de método, o depurador avança para a primeira linha do escopo deste método;

**Step Return:** avança para a primeira linha de processamento após sair do método atual, frequentemente utilizado para sair de um “Step Into”;

**Step Over:** avança para a primeira linha de processamento após a linha atual, sem entrar no método chamado, quando for o caso.

Para depuração mais avançada, como análise de gerenciamento de memória, armazenamento e processamento, existe o ambiente **DDMS** (Dalvik Debug Monitor Service). Todos ambientes disponibilizados no IDE podem ser alternados pelos seus botões de atalho (por definição no canto superior direito) ou através do menu principal “Window -> Open Perspective”.

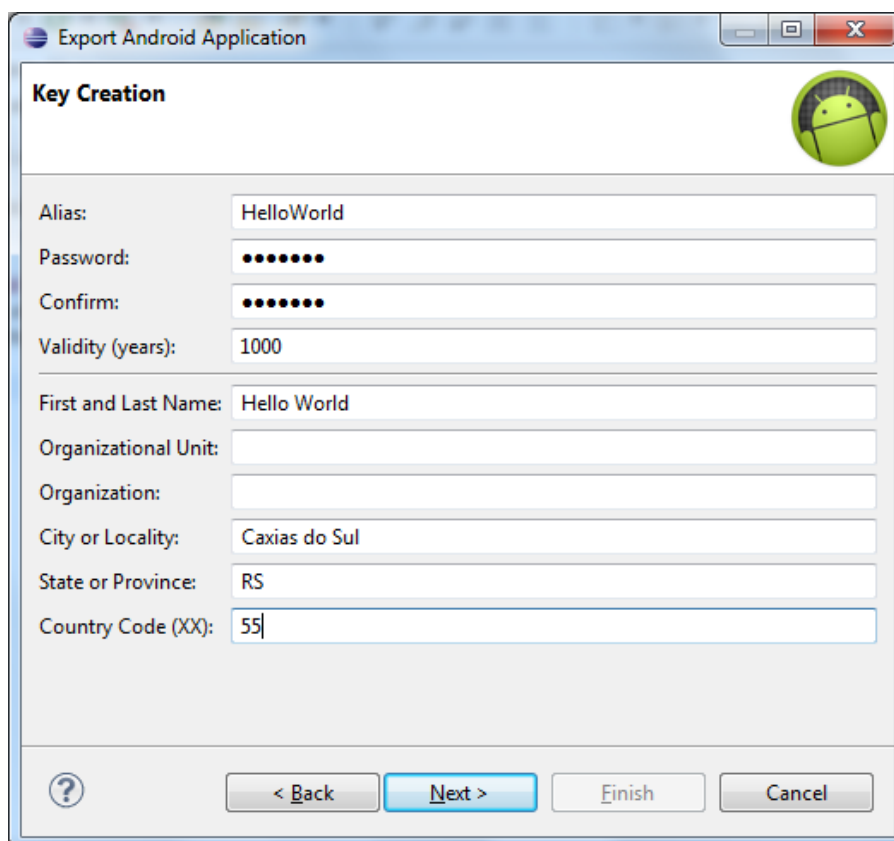


Com a utilização do DDMS é possível enviar para o emulador, mudanças de sinal (como, por exemplo, a perda de conexão de dados 3G), mudança de localização GPS, simulação de entrada de chamada, etc. É importante entender que este funcionamento está relacionado apenas à dispositivos emulados, e não funciona com dispositivos reais conectados e depurados. Além disso, na guia “Allocation Tracker”, é possível monitorar a alocação de objetos em memória, selecionando na guia “Devices” qual o aplicativo a ser monitorado (no caso, o “HelloWorld”). Através do “File Explorer”, é possível controlar a entrada e saída de elementos no sistema de arquivo do emulador, assim como do seu cartão SD simulado, o que pode ser muito útil em aplicações que gerenciam arquivos (por exemplo, uma câmera). Por fim, pode-se analisar o “LogCat”, responsável pela exibição de mensagens de mais baixo nível, diretamente da máquina virtual, com a possibilidade de realizar filtros de busca.

## 2.6. Empacotamento, publicação e instalação do app

*Após uma aplicação Android ser bem arquitetada, codificada, enfim produzida, é necessário realizar o empacotamento para instalação nos dispositivos, além da publicação no Google Play. Um arquivo “executável” de uma aplicação Android é exportado em formato APK, que compacta as informações necessárias para interpretação do programa.*

*Para gerar um APK a partir do Eclipse é relativamente simples. Com o menu suspenso “Android Tools -> Export Application Package” é possível escolher entre duas opções, “Signed” ou “Unsigned”. Para eventuais testes e distribuição não oficial (como versões beta), é possível gerar um pacote não assinado (unsigned). Já para publicação no Google Play, é obrigatório que o aplicativo seja digitalmente assinado (signed). Para isso, é necessário criar uma chave de assinatura digital criptografada, contendo um encapsulamento por usuário e senha e dados informativos, como validade do certificado e nome e localização da empresa produtora do app.*



The screenshot shows the 'Export Android Application' dialog box with the 'Key Creation' tab selected. The dialog contains the following fields and values:

- Alias: HelloWorld
- Password: (masked with dots)
- Confirm: (masked with dots)
- Validity (years): 1000
- First and Last Name: Hello World
- Organizational Unit: (empty)
- Organization: (empty)
- City or Locality: Caxias do Sul
- State or Province: RS
- Country Code (XX): 55

At the bottom, there are four buttons: a help icon (?), '< Back', 'Next >' (highlighted), and 'Finish'. A 'Cancel' button is also present.

*Com o APK gerado e digitalmente assinado, é possível enviá-lo para publicação no Google Play.*

<https://play.google.com/apps/publish/>

*Acessando o endereço acima, devidamente autenticado com uma conta Google, é possível enviar um arquivo APK, informando dados de publicação, como nome do app, versão atual, descritivo do produto, categoria, vídeos e imagens de divulgação, logo, etc.*

Detalhes do produto	Arquivos APK
<b>Enviar recursos</b>	
Capturas de tela pelo menos duas	Adicionar uma captura de tela: <input type="button" value="Escolher arquivo"/> Nenhum a...cionado <input type="button" value="Enviar"/>
	<b>Capturas de tela:</b> 320 x 480, 480 x 800, 480x854, 1.280x720, 1.280x800 24 bits no formato PNG ou JPEG (sem alfa) Sem margens nem bordas É possível fazer upload das capturas de tela em orientação paisagem. As miniaturas parecerão invertidas, porém, as imagens reais e suas orientações serão preservadas.
Ícone do aplicativo em alta resolução <a href="#">[Saiba mais]</a>	Adicionar um ícone de aplicativo de alta resolução: <input type="button" value="Escolher arquivo"/> Nenhum a...cionado <input type="button" value="Enviar"/>
	<b>Ícone do aplicativo em alta resolução:</b> 512 x 512 PNG ou JPEG de 32 bits Máximo: 1024 Kb
Gráfico promocional opcional	Adicionar gráfico promocional: <input type="button" value="Escolher arquivo"/> Nenhum a...cionado <input type="button" value="Enviar"/>
	<b>Gráfico promocional:</b> 180 L x 120 A PNG ou JPEG de 24 bits (sem alfa) Sem bordas na arte
Gráfico de recursos opcional <a href="#">[Saiba mais]</a>	Adicionar gráfico de recursos: <input type="button" value="Escolher arquivo"/> Nenhum a...cionado <input type="button" value="Enviar"/>
	<b>Gráfico de recursos:</b> 1024 x 500 PNG ou JPEG de 24 bits (sem alpha) Será diminuído para tamanho mini ou micro
Vídeo promocional opcional	Adicionar link de vídeo promocional: <input type="text" value="http://"/>
	<b>Vídeo promocional:</b> Insira o URL do YouTube
Desativação de publicidade	<input checked="" type="checkbox"/> Não promover meu aplicativo, exceto no Android Market e em qualquer propriedade on-line ou para celular pertencente ao Google. Entendo que quaisquer alterações nesta preferência podem demorar seis dias para entrar em vigor

## Detalhes da entrada

Idioma | \*English (en) |  
[adicionar idioma](#) O asterisco (\*) indica o idioma padrão.

Title (Inglês)   
10 caracteres (máx. 30)

Description (Inglês)   
15 caracteres (máx. 4000)

Recent Changes (Inglês)   
[\[Saiba mais\]](#)  
14 caracteres (máx. 500)

Promo Text (Inglês)   
15 caracteres (máx. 80)

Tipo de aplicativo

Categoria

## Opções de publicação

Proteção contra cópia ☒ Desligado (O aplicativo pode ser copiado do dispositivo)  
☐ Ligado (Ajuda a impedir a cópia deste aplicativo do dispositivo. Aumenta a quantidade de memória no telefone necessária para instalação do aplicativo.)  
O recurso de proteção contra cópia será desativado em breve; em vez disso, use o [serviço de licenciamento](#).

Avaliação do conteúdo [\[Saiba mais\]](#)  
☐ Alto nível de maturidade  
☐ Médio nível de maturidade  
☐ Baixo nível de maturidade  
☒ Todos

Preço ☒ Gratuito ☐ Pago  
Dispositivos compatíveis [\[Saiba mais\]](#)  
A configuração do preço como "Gratuito" é permanente, e você não poderá alterar para um preço. [\[Saiba mais\]](#)

Definir um preço para cada país/região

Preço padrão USD

☒ Todos os países

☒ [África do Sul](#)

☒ [Alemanha](#)

☒ [Argentina](#)

☒ [Austrália](#)

☒ [Áustria](#)

☒ [Bélgica](#)

☒ [Brasil](#)

☒ [Bulgária](#)

☒ [Canadá](#)

☒ [Chipre](#)

☒ [Índia](#)

☒ [Irlanda](#)

☒ [Islândia](#)

☒ [Israel](#)

☒ [Itália](#)

☒ [Japão](#)

☒ [Letônia](#)

☒ [Lituânia](#)

☒ [Luxemburgo](#)

☒ [Malta](#)

*É interessante observar algumas opções disponíveis na publicação. A principal delas é a opção de proteger o aplicativo contra cópias, o que faz com que o aplicativo tenha que ser baixado através do Play, não podendo ser copiado para outros dispositivos. Em uma aplicação gratuita onde a necessidade é a distribuição em larga escala, pode ser um bom recurso a desativação da proteção contra cópias.*

*É possível, ainda, definir um valor de venda do aplicativo. O recebimento dos valores é realizado através da conta Google configurada no “Google Checkout”, o mesmo sistema de recebimentos utilizado pelo “Google AdSense”. A Google realiza pagamentos acima de U\$ 100,00, através de transferência por bancos internacionais, portanto é necessário ter uma conta que suporte tal recebimento.*

*Após a aprovação dos regulamentos e diretrizes impostas pela Google, o aplicativo está imediatamente disponível nos países optados através do cadastro. Outro diferencial do sistema da Google sobre os concorrentes é a publicação, onde a avaliação de integridade é realizada após a publicação, fazendo com que esta seja instantânea.*



### 3. Conceitos Básicos

*Para construir um aplicativo além do “HelloWorld” é necessário conhecer um pouco mais sobre os componentes fundamentais do Android.*

**Activities:** classes (“Activity”) responsáveis pelo código back-end de uma interface gráfica, é onde tudo começa na aplicação;

**Services:** classes (“Service”) responsáveis por códigos a serem executados por tempo maior, em segundo plano, sem a necessidade de intervenção por parte do usuário;

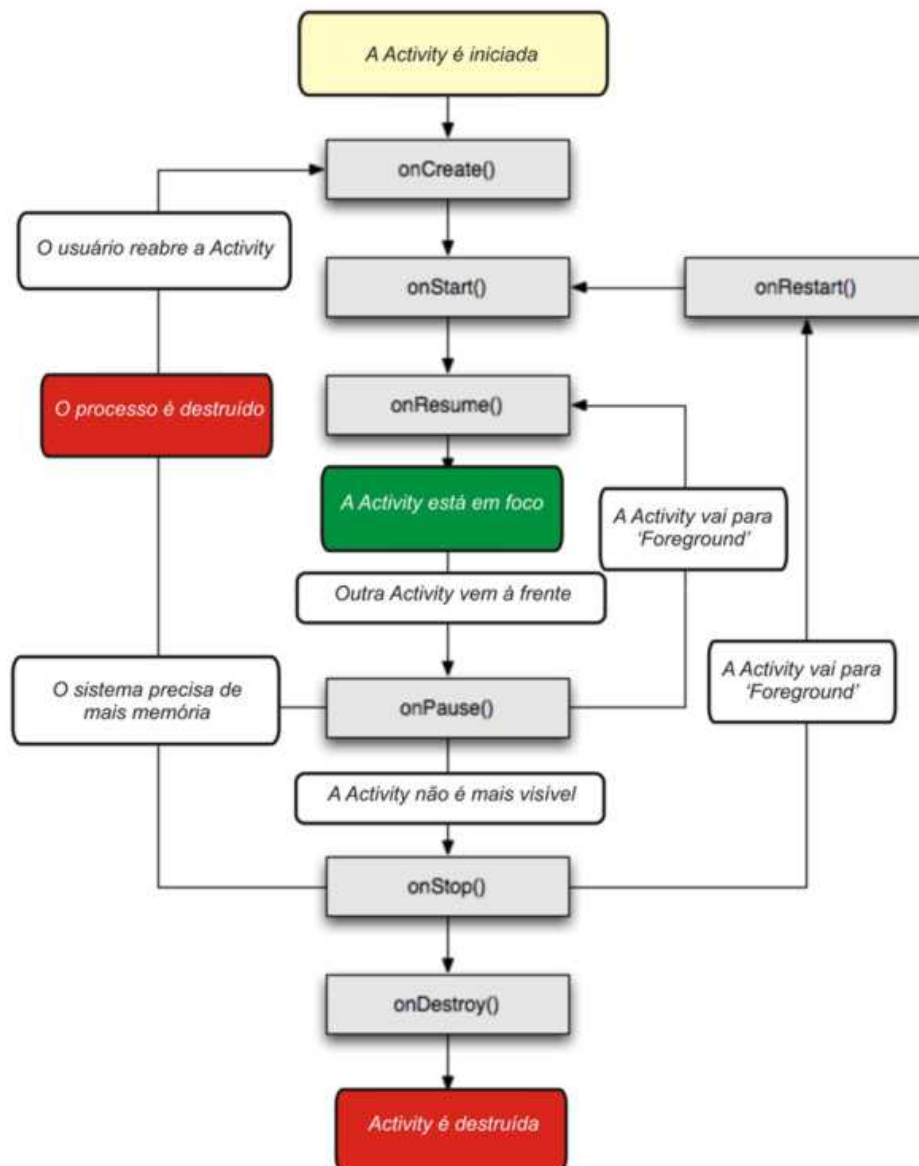
**Content Provider:** no Android, não é possível acessar conteúdo de outros aplicativos diretamente a partir de um aplicativo terceiro. Para resolver isso, existe esta classe que fornece os conteúdos como, por exemplo, o acesso à galeria de fotos ou agenda de contatos, através de chamadas e URI (Uniform Resource Identifier) específicos. É possível, ainda, implementar provedores de conteúdo próprios para acesso externo;

**Intent e IntentFilter:** a classe “Intent” funciona (como nome sugere, intenção) como uma chamada à permissão de execução de ação. Toda ação solicitada ao Android para executar é um Intent, como a transição entre “Activity”, realização de uma chamada telefônica ou acesso a um “ContentProvider”. “IntentFilter”, por sua vez, serve para predefinir chamadas para “Intents” específicos através de constantes, indicando quais recursos a atividade tem interesse em utilizar;

**Processos e Threads:** por padrão, o sistema Android gerencia toda a aplicação em um único processo. Em alguns momentos pode ser interessante criar processos separados para algumas atividades, a fim de que a quebra dela não interfira na quebra do sistema. Já a concorrência de código (“Thread”) funciona como uma única concorrência de interface, podendo ser necessário separar alguns processamentos dela.

### 3.1. Activity, Intent e IntentFilter

Partindo do conceito de que uma atividade (“**Activity**”) Android funciona como um “gerenciador” back-end de uma interface gráfica, pode-se entender que esta possui um ciclo de vida, e que cada fase deste ciclo possui suas próprias funções e características.



*Pode-se notar que a utilização de chamadas de espera ("pause") de uma atividade para exibição de outra é um processo muito comum e prático, porém é importante observar que o gerenciamento de memória em uma aplicação móvel é bastante delicado. É necessário possuir a certeza de que o processo não será interrompido ao executar uma espera, caso contrário o conteúdo do processo anterior poderá ser perdido.*

*Quando realiza-se a transição entre atividades (e esta pode ser feita com suporte a transições gráficas de UI, User Interface), a atividade nova passa ao estado de "execução", enquanto a anterior passa ao estado de "espera", caso não haja despejo de memória. Desta forma, a atividade em execução consegue se comunicar com a atividade em espera. A maneira mais comum de realizar transição entre atividades é a partir da criação de um "Intent".*

```
public class Main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button addButton = (Button)findViewById(R.id.bAdd);
        addButton.setOnClickListener(add_listener);
    }

    OnClickListener add_listener = new OnClickListener() {
        @Override
        public void onClick(android.view.View v) {
            Intent myIntent = new
Intent(getApplicationContext(),AddClass.class);
            myIntent.putExtra("id_item", 0);
            startActivity(myIntent);
        }
    }
}
```

*Vale à pena observar o tratamento de eventos em uma interface Android. No exemplo acima, foi criado um Objeto "OnClickListener" e atribuído ao botão "addButton". Desta forma, ao acioná-lo na aplicação, é criado um novo "Intent" referente à "Activity" "AddClass", passado um parâmetro de referência "id\_item" como "0" (em uma edição, pode-se passar o ID do item, por exemplo) e iniciada a atividade.*

*Outro código importante é referente à criação do botão "addButton". É possível notar que o parâmetro passado ao método "findViewById" é "R.id.bAdd". Isso porque a*

classe “R”, responsável por armazenar referências em hexadecimal dos elementos de layout possui subclasses, dentre elas a “id”, responsável por referenciar diretamente elementos gráficos. Imaginado que existe um arquivo de layout correspondente à atividade (no caso, “R.layout.main” representa o arquivo “main.xml”), entende-se que existe um elemento com “ID” “bAdd”, neste arquivo.

Após a utilização do “putExtra()” para passar parâmetros ao “Intent”, é possível, na inicialização da atividade chamada recuperar estes parâmetros através do seguinte código.

```
Bundle extras = getIntent().getExtras();  
int id_item = extras.getInt("id_item");
```

É possível buscar o conteúdo do armazenamento de parâmetros a partir de métodos como “getInt()”, também utilizado com String (“getString()”) ou qualquer tipo primitivo, dependendo do parâmetro passado no “putExtra()”.

Dentre os principais métodos da classe “Activity”, pode-se destacar:

**onCreate()** - é o primeiro método a ser executada quando uma “Activity” é instanciada. Geralmente é o método responsável por carregar os layouts (XML). É executada somente uma vez durante o ciclo de vida;

**onStart()** - é chamado imediatamente após a criação, mas também quando uma “Activity” em background volta a ter foco;

**onResume()** - assim como “onStart()”, é chamado na inicialização e na reinicialização da “Activity”. A diferença é que “onStart()” só é chamado quando a “Activity” não mais visível na tela volta a ter foco, enquanto “onResume()” sempre é chamado;

**onPause()** - é o primeiro método a ser invocado quando a “Activity” perde o foco (ou seja, uma outra “Activity” entrou em execução);

**onStop()** - análoga à “onPause()”, só é chamada quando a “Activity” fica completamente encoberta por outra “Activity” (não é mais visível).

**onDestroy()** – último método do ciclo de vida a ser executado. Depois, a “Activity” é considerada “morta”, ou seja, não podendo ser reiniciada;

**onRestart()** - chamado imediatamente antes de “onStart()”, quando uma “Activity” volta a ter o foco após estar em background.

Como já visto anteriormente e utilizado no exemplo, um “**Intent**” (no sentido literário, intenção), é a informação de chamada de execução específica do sistema a realizar. Na análise do código, nota-se o uso simples de um “Intent”, utilizando o contexto atual da aplicação e passando a referência de uma Classe a ser utilizada como atividade. Isso faz com que a atividade chamada entre em plano principal.

Com a utilização de “**IntentFilter**” é possível informar quais recursos estarão disponíveis para a atividade (não confundir com permissões) para que esta possa ser mais enxuta possível. Para cada atividade, é necessário definir no manifesto a utilização deste filtro, assim como criar as categorias disponíveis (ao menos uma é necessária).

Dentro da tag “<application/>”, devem ser definidas todas as atividades do programa (sempre) e os filtros de intenções das respectivas atividades.

```
<activity
    android:name=".Main"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER"
/>
    </intent-filter>
</activity>
```

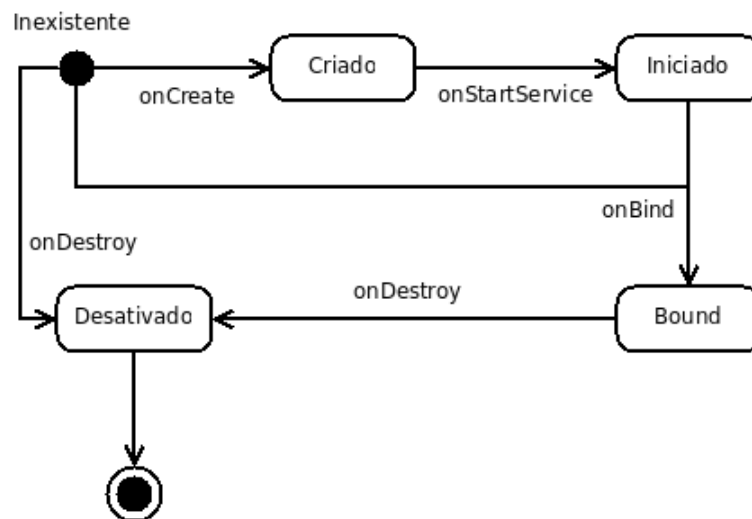
No caso, a atividade referente à classe “Main” possui os interesses em ser a principal atividade (“MAIN”) e poder ser inicializada pelo app launcher (“LAUNCHER”).

A lista completa de filtros pode ser encontrada no endereço:

<http://developer.android.com/reference/android/content/Intent.html>

## 3.2. Service

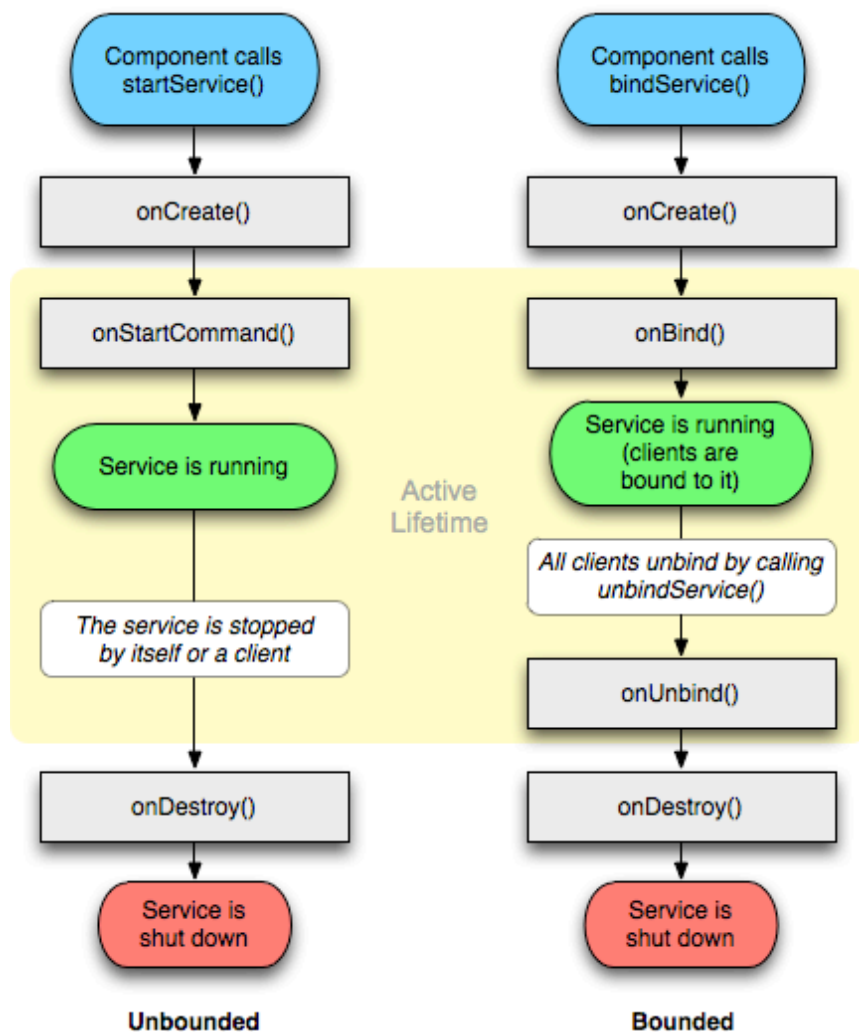
Um serviço (“**Service**”), semelhantemente a uma atividade, também possui um ciclo de vida, porém este é válido até o sistema necessitar de memória, caso não haja um pedido de interrupção previsto.



Serviços são utilizados normalmente para executar algum processo em segundo plano, onde não seja necessária a intervenção por parte do usuário. Um exemplo comum é o download de apps do Google Play. Ao solicitar o download, é criado um ícone na barra de notificação que indica que o app está sendo baixado. Este processo de download é executado em segundo plano e não necessita intervenção por parte do usuário para ser concluído.

Existem dois tipos de serviços, os **locais** e os **remotos**. Serviços locais são aqueles executados dentro do processo da aplicação, sem necessidade de nenhuma interferência remota. Poderiam ser substituídos por “Thread”, porém, em processamentos mais complexos, uma “Thread” pode gerar problemas, enquanto um “Service” tem propriedades específicas de processamentos mais complexos. Já os serviços remotos são executados em um processamento em separado da aplicação principal, totalmente independente, podendo ser executado em segundo plano sem a aplicação principal estar em execução (como uma quebra de memória). Geralmente, são utilizados para construção de webservices, onde a dependência de uma conexão de Internet é alta e torna-se interessante a exclusão do processamento. O que diferencia estes dois tipos é sua declaração no manifesto (“AndroidManifest.xml”).

Além disso, existem dois tipos de implementação de um serviço Android. A primeira consiste no “**Unbounded**” ou “**Started**”, onde o serviço é iniciado e somente será interrompido quando for solicitada sua parada através de código próprio ou de outra aplicação que tenha poder de interferência nele (o que é declarado no manifesto). A segunda consiste no “**Bounded**”, onde o serviço fica permanentemente amarrado à aplicação que o iniciou, sendo o serviço finalizado por código explicitou ou encerramento da aplicação que o iniciou. A diferença está nos métodos utilizados para codificação. Enquanto um serviço “**Started**” é executado a partir do “`startService()`”, um serviço “**Bounded**” é iniciado pelo “`bindService()`”.



```
public class HelloService extends Service {
    int mStartMode;          // indica o comportamento do serviço
    IBinder mBinder;         // interface para os clientes amarrados
    boolean mAllowRebind;    // indica que o service pode ser amarrado

    @Override
    public void onCreate() {
        // serviço criado
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        // service é iniciado pelo startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // cliente é conectado ao serviço pelo bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // todos os clientes são desamarrados pelo unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // cliente é conectado ao serviço novamente através do
        bindService(), depois que o onUnbind() foi chamado
    }
    @Override
    public void onDestroy() {
        // o service é finalizado e destruído
    }
}
```

*Dentre os principais métodos de “Service”, pode-se destacar:*



***onStartCommand()*** - método chamado quando outro componente, como uma atividade, faz a requisição para que o serviço seja iniciado chamando “startService()”. O serviço é iniciado e pode rodar em background indefinidamente. Se for implementado isso, é responsabilidade do código parar o serviço, uma vez que o trabalho é feito chamando “stopSelf()” ou “stopService()”;

***onBind()*** - método chamado quando outro componente quer fazer o bind com o serviço (para realizar um RPC, Remote Procedure Call) chamando “bindService()”. Na implementação deste método, deve ser provida uma interface onde os clientes poderão comunicar-se com o serviço, retornando um “IBinder”;

***onCreate()*** - método chamado quando o serviço é criado pela primeira vez, para realizar procedimentos de configuração iniciais (antes mesmo de executar ou o “onStartCommand()” ou “onBind()”);

***onDestroy()*** - método chamado quando o serviço está sendo destruído. Método que deve ser implementado para limpar os recursos instanciados, como threads, listeners registrados, receivers, etc. É a última chamada que o serviço recebe.

### 3.3. ContentProvider

*“**ContentProvider**” é uma das partes mais importantes no estudo de arquitetura de aplicações Android, pois ele é responsável pela comunicação saudável entre apps. O sistema operacional da Google foi construído com base em algumas normas de boas práticas, dentre as quais consta que o ideal é que a persistência de um programa seja transparente, tanto ao código-fonte de negócio do próprio programa quanto a programas externos. Dessa forma, não é possível acessar um banco de dados de uma aplicação externa diretamente e é aí que entram os provedores de conteúdo.*



*A comunicação entre aplicações e provedores de conteúdo se torna simples, realizada a partir de um endereço **URI**, onde é possível realizar ações como inclusão, exclusão e, principalmente, consulta de dados. Cada “Content Provider” possui uma estrutura de URI própria, baseada na estrutura geral. Por exemplo, para acessar o conteúdo da caixa de mensagens de texto, é possível solicitar o “ContentProvider” compartilhado do aplicativo de mensagens através a URI “**content://sms/inbox**”. É possível ainda realizar a passagem de parâmetros para esta URI.*

***content://pacote.app.nomeDoCP/nomeEntidade/ID***

*Da mesma forma que o utilizado por “Activity” e “Service”, para criar um provedor de conteúdo, basta derivar uma Classe da “ContentProvider”, implementando seus métodos, e realizar a inclusão referente no manifesto.*

```
public class HelloProvider extends ContentProvider {
    public static final Uri CONTENT_URI = Uri
.parse("content://hello.world.helloprovider");

    @Override
    public int delete(Uri uri, String selection, String[]
selectionArgs) {
        return 0;
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }

    @Override
    public boolean onCreate() {
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String
selection,
        String[] selectionArgs, String sortOrder) {
        return null;
    }

    @Override
    public int update(Uri uri, ContentValues values, String
selection,
        String[] selectionArgs) {
        return 0;
    }
}
```

*Dessa, forma é possível realizar a chamada de conteúdo a partir de outro app, através da URI “content://hello.world.helloprovider”.*

*Dentre os principais métodos da classe “ContentProvider”, pode-se destacar:*

***query()*** - usado para recuperar dados;

***insert()*** – usado para inserir dados;

***update()*** – usado para atualizar dados;

***delete()*** - usado para excluir dados;

***getType()*** – usado para obter o MIME type de certo dado;

### 3.4. Processo e Thread

*Cada aplicação possui um processo único de execução e todos os elementos pré-definidos dela são executados dentro deste processo. Em alguns casos, pode ocorrer sobrecarga de memória e processamento. Para resolver isso, existem algumas artimanhas, como a utilização de serviços remotos (visto anteriormente). Já para processamentos em menor escala, mas que necessitam de execução concorrente, pode ser utilizada uma implementação da classe “**Thread**” (ou “**Runnable**”), como uma aplicação Java comum.*

```
public class ProgressDialogExample extends Activity implements
Runnable {
    private String pi_string;
    private TextView tv;
    private ProgressDialog pd;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        tv = (TextView) this.findViewById(R.id.main);
        tv.setText("Press any key to start calculation");
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {

        pd = ProgressDialog.show(this, "Working..",
"Calculating Pi", true,false);

        Thread thread = new Thread(this);
        thread.start();

        return super.onKeyDown(keyCode, event);
    }

    public void run() {
        pi_string = Pi.computePi(800).toString();
        handler.sendMessage(0);
    }

    private Handler handler = new Handler() {
```

```
        @Override
        public void handleMessage(Message msg) {
            pd.dismiss();
            tv.setText(pi_string);
        }
    };
}
```

*O exemplo acima calcula o valor do PI com 800 comutações e o escreve na tela. O processo é disparado por um simples apertado de tecla (caso teclado físico, senão toque) e, caso o evento de tecla ocorra durante o processamento, é disparada uma mensagem informando o não término dele.*

*O objeto “**Handler**” é muito importante, pois é ele que faz a comunicação entre os processos concorrentes.*

*<http://developer.android.com/reference/android/os/Handler.html>*

### 3.5. Manifesto

*Conhecidos os principais elementos de uma estrutura de código Android, pode-se estudar o arquivo de manifesto da aplicação. Este arquivo é responsável por definir as informações vitais para o app, como identificadores, versões, permissões de acesso externo e definir quais elementos são presentes nesta aplicação, como atividades, serviços, provedores de conteúdo e filtros de intenções.*

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

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
```

```

    </receiver>

    <provider>
        <grant-uri-permission />
        <meta-data />
    </provider>

    <uses-library />

</application>
</manifest>

```

*Cada tag de utilização prevista no “AndroidManifest.xml” possui diversos atributos que podem ser utilizados e sofrem constantes mudanças devido às novas versões do Android. A relação completa de atributos mais atual pode ser encontrada seguinte endereço:*

*<http://developer.android.com/guide/topics/manifest/manifest-intro.html>*

*Como exemplo de um manifesto de aplicação “HelloWorld”, tem-se:*

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="hello.world"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".HelloWorldActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>

```

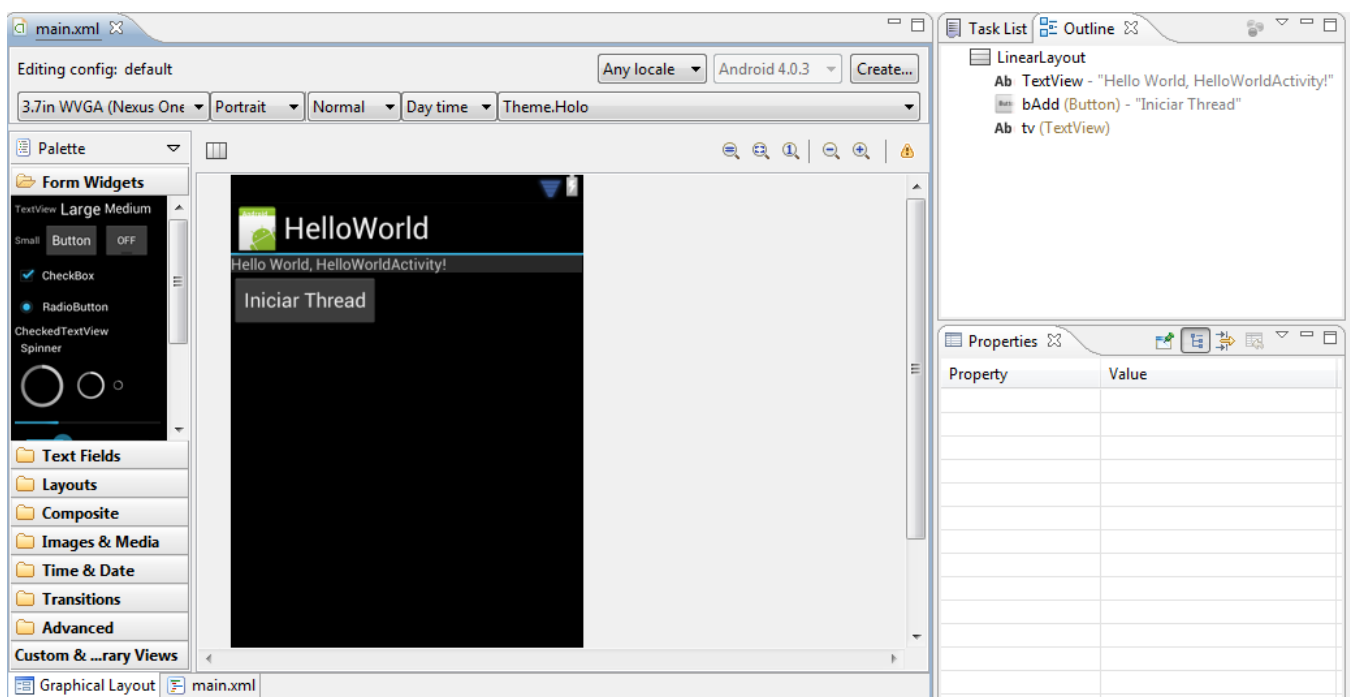


## 4. Interface Gráfica

*Com os aplicativos de smartphones e tablets cada vez mais integrados à web, a catálogos e ao conceito de armazenamento de informações em nuvem, um dos principais elementos de um app é a apresentação gráfica. Tanto em jogos, quanto em catálogos e em aplicativos corporativos ou de produtividade, a direção de arte converge para elementos mais limpos e de navegação mais intuitiva, atingindo diversos níveis de usuários, desde os mais novatos aos mais experientes.*

*O Android SDK combinado com o ADT possui recursos e bibliotecas de enorme qualidade para construção de uma arquitetura de layout moderna e representativa. A construção a partir destes recursos é como uma mistura entre as interfaces “Swing” e HTML (web), suportando a construção de layouts em tabela, em camadas e atributos de posicionamento, controle de bordas, imagens, cores, e diversos componentes visuais.*

*Como já visto, a arquitetura de um aplicativo Android Java possui elementos de layout em XML, carregados, geralmente, a partir de uma atividade. Ao editar um arquivo desse tipo com o Eclipse, com o ADT, é possível visualizar tanto o código texto (XML), quanto a edição visual, onde é possível adicionar componentes e controlar os layouts a partir de “drag and drop”.*

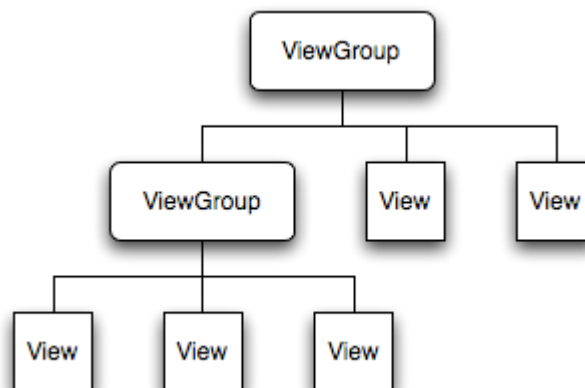


É possível notar basicamente três blocos principais para arquitetura de tela no ADT. A parte principal, o editor, é composta de uma aba para edição visual (“Graphic Layout”) e outra de código (“nomeDoArquivo.xml”). Na guia de edição visual, é possível observar uma barra de componentes visuais e uma área de montagem de tela. É possível, então, arrastar um componente para a área de montagem. Cada modificação feita no editor visual é automaticamente carregada no editor de texto e vice e versa.

Além disso, é possível observar uma barra lateral contendo a exibição de hierarquia de componentes em árvore e um bloco, talvez o principal, de edição de propriedades do componente. Ao selecionar um componente específico, são exibidas neste bloco as propriedades referentes a ele, como alinhamento, posicionamento, cores, textos e o ID do componente.

Por padrão, a propriedade ID do componente é composta de um valor em sequência (ex.: “@+id/button1”). É importante manter o valor até a primeira barra (“/”), alterando apenas o conteúdo final, o ID em si (“button1”, no caso). Isto faz com que, no componente de referências gerado automaticamente (“R.class”), seja adicionada uma referência a este objeto, podendo ser utilizada a chamada direta (como “R.id.button1”).

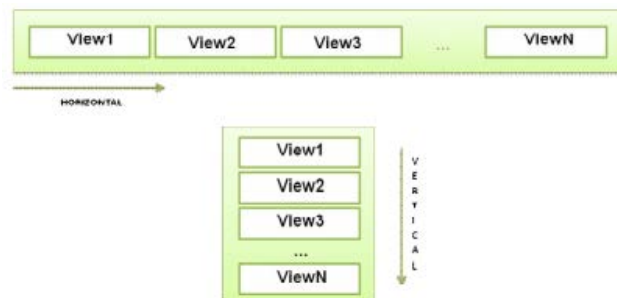
Cada componente de layout é derivado de “View”(widgets) ou “ViewGroup”, sendo que um “ViewGroup” pode ser composto de “View” ou de outros “ViewGroup”, na quantidade que for necessária para construir um layout.



## 4.1. Layouts

*Existem diversos tipos de arquitetura de layouts em Java e em HTML, e algumas das principais estão presentes no desenvolvimento Android. Os containers de layout são componentes derivados de “ViewGroup”. Dentre os principais tipos de layout de uma arquitetura de tela Android, pode-se destacar os seguintes:*

**LinearLayout** – é o layout utilizado por padrão nos componentes XML criados. Seu principal objetivo é montar os elementos em forma corrente, sendo da esquerda para a direita em layout horizontal e de cima para baixo em layout vertical. A orientação do layout é definida pela propriedade “orientation”.



*Este estilo de layout pode ser útil em casos de design de tela simples, com elementos ocupando áreas completas de largura ou altura (dependendo da orientação especificada), pois ele respeita automaticamente as margens e os alinhamentos perante outras “View”. É possível definir o alinhamento dos componentes na edição dos layouts Android, através da propriedade “gravity”. Através da propriedade “layout\_weight” é possível organizar os elementos por “pesos” (prioridades), a fim de algumas “Views” ocuparem maior ou menor espaçamento em tela (o padrão é “0”).*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

    <Button
        android:id="@+id/bAdd"
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="Iniciar Thread" android:layout_weight="0"/>

<TextView
    android:id="@+id/tv"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="" />

</LinearLayout>

```

**FrameLayout** - organiza o elemento em pilhas conforme sua adição, onde os elementos adicionados por último são mostrados na camada mais acima, podendo estes elementos ser reordenados através da árvore de hierarquias. É normalmente utilizado como container para troca de componentes de mesmo tipo. Por exemplo se existe uma imagem a ser trocada temporariamente (ou em algum evento específico, como um toque em um botão), é possível colocar todas as imagens em um “FrameLayout”, ocultando as inutilizadas a partir da propriedade “visibility”.



```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frameLayout1"
    android:layout_width="match_parent"
    android:layout_height="fill_parent" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" android:visibility="invisible"/>

    <Button

```

```

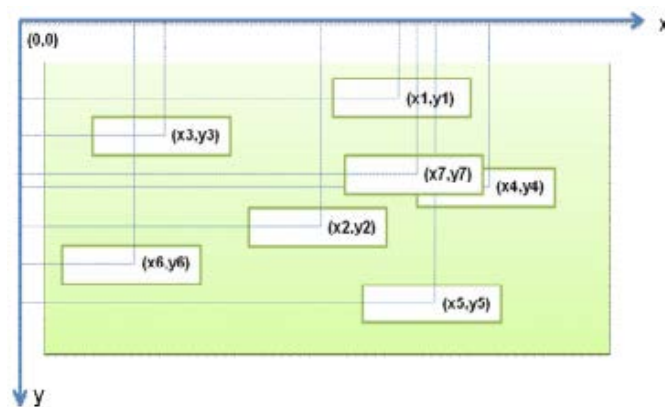
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" android:visibility="invisible"/>

<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 3" />

</FrameLayout>

```

***AbsoluteLayout*** - este container é derivado da programação HTML, onde é possível definir camadas de posicionamento “x,y” absoluto sobre outros elementos, como um plano cartesiano. A coordenada “x” é medida da esquerda para a direita e a camada “y” de cima para baixo. Por padrão, os elementos adicionados em um “AbsoluteLayout” possuem coordenadas (0,0), ficando no topo esquerdo da tela. Este tipo de layout é comum na exibição de mensagens personalizadas sobre a tela (e sobre outros componentes). É importante cuidar ao construir um layout absoluto, pois além de elementos poderem ser sobrepostos por descuido no posicionamento, é possível que ocorra muita diferença entre resoluções diferentes (as medidas são absolutas).



```

<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/absoluteLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="248dp"

```

```

        android:layout_height="wrap_content"
        android:layout_x="39dp"
        android:layout_y="58dp" >

        <requestFocus />
    </EditText>

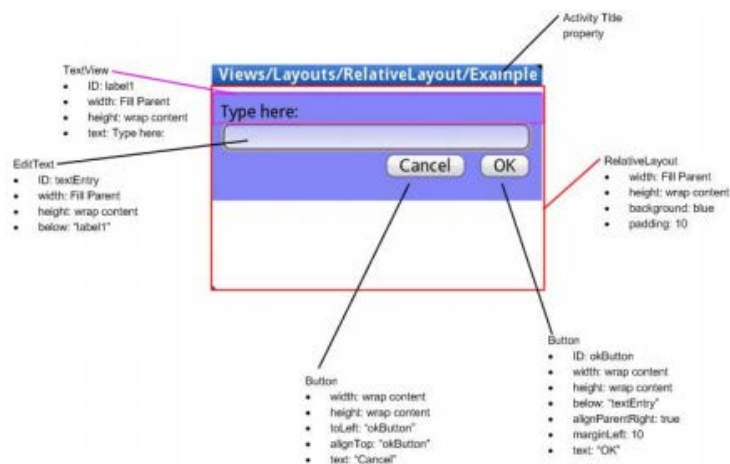
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="208dp"
        android:layout_y="119dp"
        android:text="Button" />

</AbsoluteLayout>

```

*Apesar de possuir vantagens, as falhas devido à má construção fizeram com que o layout absoluto fosse depreciado no Android, tendo sua implementação descontinuada.*

**RelativeLayout** - os layouts relativos servem para posicionar componentes com relacionamentos entre eles, podendo um elemento ser posicionado abaixo de outro, por exemplo.



*É o tipo de container mais utilizado e recomendado, pois com ele é possível realizar “amarrações” de elementos às bordas da tela, ficando o layout íntegro e robusto, independente da resolução. Além disso, é possível definir posicionamentos relativos à outros elementos do layout, podendo ser especificada uma margem a partir da “amarração”, novamente mantendo o layout íntegro e robusto.*

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/relativeLayout1"
        android:layout_width="fill_parent"

```

```

        android:layout_height="fill_parent" >

        <TimePicker
            android:id="@+id/timePicker1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentRight="true"
            android:layout_alignParentTop="true" />

        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_below="@+id/timePicker1"
            android:text="TextView" />

        <EditText
            android:id="@+id/editText1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentRight="true"
            android:layout_below="@+id/textView1" >

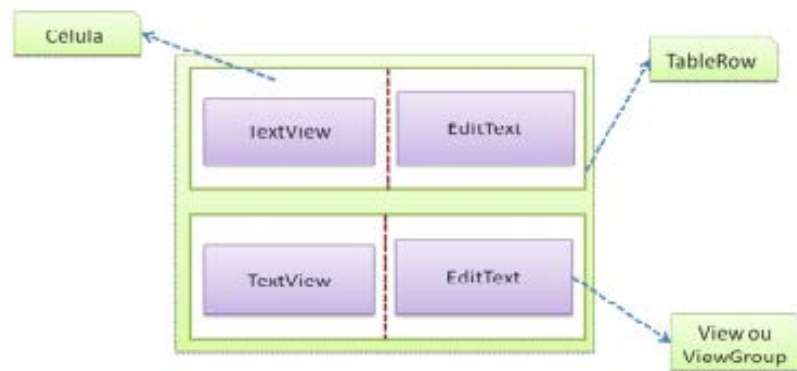
            <requestFocus />
        </EditText>

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_below="@+id/editText1"
            android:text="Button" />

    </RelativeLayout>

```

**TableLayout** - este container é utilizado da mesma forma que tabelas em HTML ou que o container de tabelas em “Swing”. A partir dele, podem ser criados “TableRow”, que funcionam, basicamente, como o “LinearLayout”. Cada “View” adicionado é considerado uma célula e o número de colunas da tabela é definido pela linha que contém mais células.



```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent">

    <TableRow>
        <TextView android:text="Nome:" />
        <EditText android:text="" />
    </TableRow>

    <TableRow>
        <TextView android:text="Data Nasc.:" />
        <EditText android:text="" />
    </TableRow>

    <TableRow>
        <Button android:text="Cadastrar" />
    </TableRow>

</TableLayout>
```

*A partir dos diversos tipos de layout existentes, é possível realizar diversos aninhamentos, a fim de realizar a melhor arquitetura de tela para a aplicação. É importante que um designer de interface tenha conhecimentos, além das suas ferramentas, do funcionamento dos containers de layout, para poder agregar os elementos da melhor forma possível e não complicar o trabalho dos engenheiros de software.*

*Além da utilização padrão dos arquivos de layout, referenciados no contexto da atividade a ser executada, é possível a utilização de um componente chamado “LayoutInflater”, utilizado para a criação de layouts personalizados em objetos com visualização padrão de componente. Um exemplo muito comum é a utilização do*



*“LayoutInflater” para a criação de notificações, como visto na seção de notificações e “Toaster” desta apostila.*

*A documentação completa do “LayoutInflater” pode ser encontrada em:*

*<http://developer.android.com/reference/android/view/LayoutInflater.html>*

## 4.2. Componentes de tela

*Os componentes visuais disponibilizados pelo SDK Android são derivados de “View” e, por isso, possuem propriedades em comum, além de suas específicas. Dentre as principais propriedades em comum, pode-se destacar as referentes ao layout, como posicionamento, tamanho em largura e altura (x,y) e margens.*

*O objetivo desta apostila é apresentar as possibilidades de componentes existentes no Android SDK e como podem ser utilizados entre si. A documentação completa da biblioteca de componentes visuais pode ser encontrada no seguinte endereço:*

<http://developer.android.com/guide/topics/ui/index.html>

*Existem alguns componentes que são aninhamentos de containers de layout, pronto para executarem alguma função específica.*

<b>Componente</b>	<b>Descrição</b>
<i>Gallery</i>	<i>Contém uma lista de imagens (suportando rolagem para direita e esquerda)</i>
<i>GridView</i>	<i>Contém uma grade com rolagem e esta contém linhas e colunas</i>
<i>ListView</i>	<i>Possui uma lista de “View” (coluna única)</i>
<i>ScrollView</i>	<i>Coluna única de elementos (“View” ou “ViewGroups”) com rolagem vertical</i>
<i>Spinner</i>	<i>Semelhante à uma caixa de seleção, realiza rolagem horizontal ou vertical, exibindo um único item por vez, como uma linha de texto</i>
<i>SurfaceView</i>	<i>Container semelhante a uma superfície de desenho, utilizado para desenhar pixels ao invés de widgets, mesmo assim suportando componentes dispostos acima</i>
<i>TabHost</i>	<i>Componente armazenador de abas, pronto para recebê-las com suporte a eventos das mesmas</i>
<i>ViewFlipper / ViewSwitcher</i>	<i>Lista de exibição única (um item por vez) que pode ser configurado para trocar componentes, como um slideshow</i>

*É muito interessante este tipo de componente, pois, mesmo sendo agrupadores de outros componentes, tem dependência total e podem ser facilmente implementados. Por exemplo, para criar uma lista de planetas selecionável, pode ser utilizado um “Spinner”.*

*No arquivo de layout:*

```
<Spinner
```

```

        android:id="@+id/spinner1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"/>

```

*No arquivo de constantes (“strings.xml”):*

```

<string name="planet_prompt">Selecione</string>
<string-array name="planets_array">
    <item>Mercúrio</item>
    <item>Vênus</item>
    <item>Terra</item>
    <item>Marte</item>
    <item>Júpiter</item>
    <item>Saturno</item>
    <item>Urano</item>
    <item>Netuno</item>
</string-array>

```

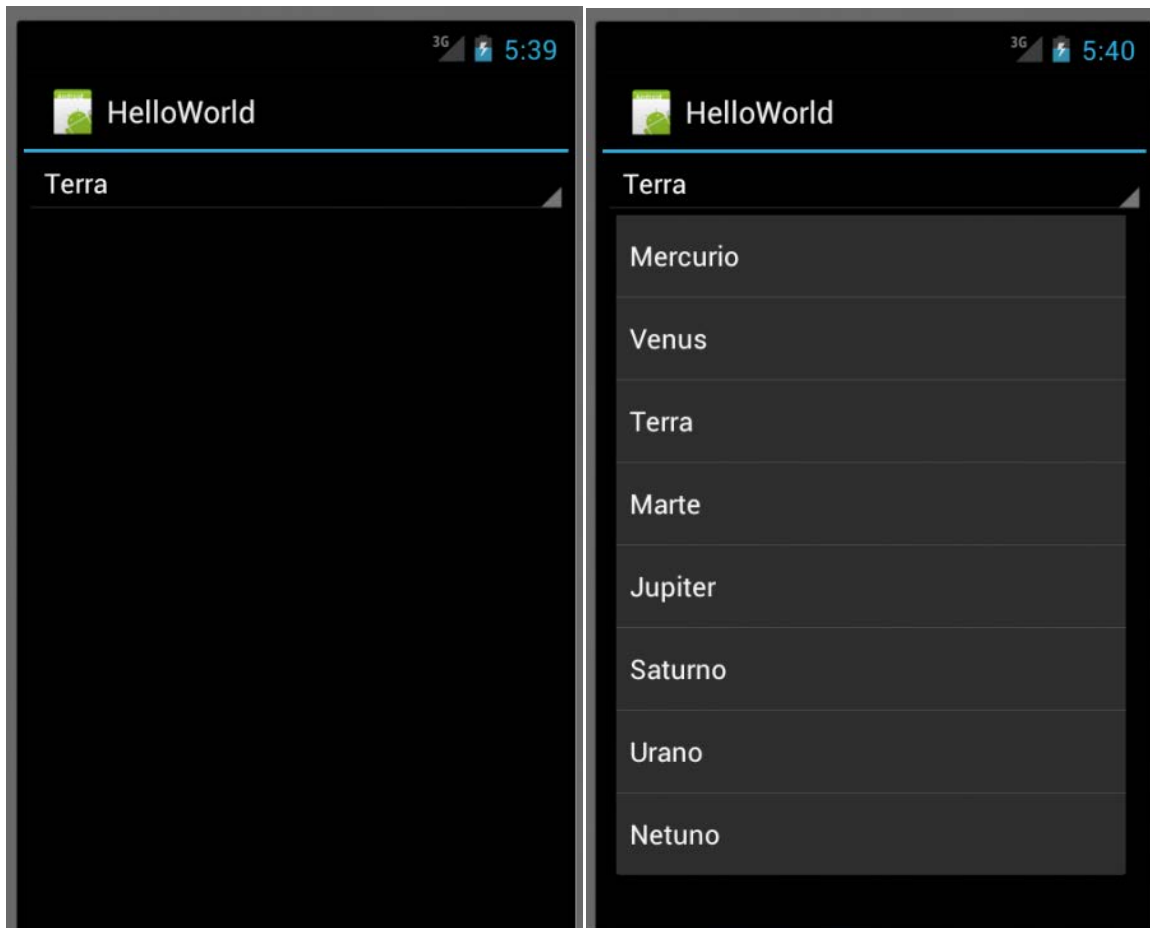
*No código da atividade que utilize o layout contendo o “Spinner”:*

```

Spinner spinner = (Spinner) findViewById(R.id.spinner1);
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
    this, R.array.planets_array,
    android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropd
own_item);
spinner.setAdapter(adapter);

```

*O resultado apresentado será algo como:*



*É interessante observar no código da atividade a utilização de uma referência ao tipo de layout utilizado para o “Spinner”. A lista completa e atualizada das constantes de layout que podem ser utilizadas pode ser encontrada em:*

*<http://developer.android.com/reference/android/R.layout.html>*

*Mas estes componentes descritos na grade acima são os chamados “componentes avançados”. Dentre os básicos do desenvolvimento visual para Android pode-se destacar:*

***TextView** – responsável pela exibição de texto em tela, possui características como cor de fundo, de texto, tamanho de texto;*

***Button** – responsável pela exibição de um botão padrão de texto simples, possui características como cor de fundo, de texto, tamanho de texto;*

***ImageButton** – semelhante ao “Button”, com a possibilidade de utilizar uma imagem (“res”) como fundo;*

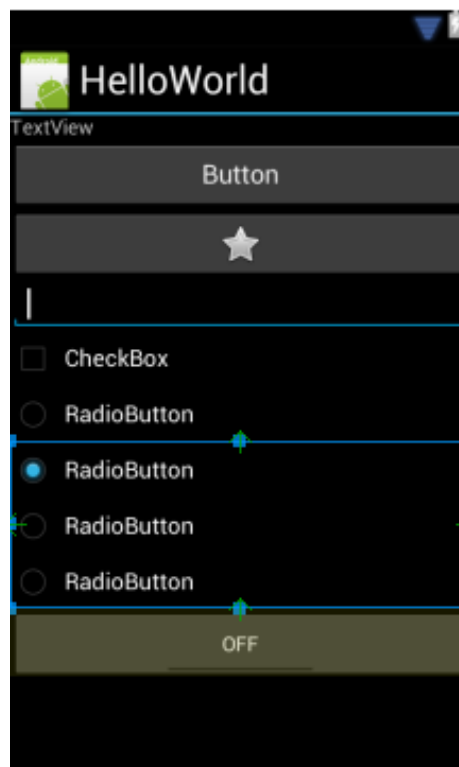
***EditText*** – campo editor de texto, a partir de qual é possível o usuário colocar informações, pode ser configurado como uma área de texto, adicionando suporte a quebras de linha;

***CheckBox*** – caixa de marcação múltipla, por padrão é acompanhada de um texto de etiqueta ao lado;

***RadioButton*** – caixa de marcação única, por padrão é acompanhada de um texto de etiqueta ao lado;

***RadioGroup*** – agrupamento de “RadioButton”;

***ToggleButton*** – botão com identificador de ligado/desligado, pode ser utilizado semelhante a um “CheckBox”, porém sem valor fixo de seleção.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/relativeLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="TextView" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/textView1"
    android:text="Button" />

<ImageButton
    android:id="@+id/imageButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/button1"
    android:src="@android:drawable/btn_star_big_off" />

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/imageButton1" >

    <requestFocus />
</EditText>

<CheckBox
    android:id="@+id/checkbox1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/editText1"
    android:text="CheckBox" />

<RadioButton
    android:id="@+id/radioButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```

        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/checkbox1"
        android:text="RadioButton" />

<RadioGroup
    android:id="@+id/radioGroup1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/radioButton1" >

    <RadioButton
        android:id="@+id/radio0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="RadioButton" />

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="RadioButton" />

    <RadioButton
        android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="RadioButton" />
</RadioGroup>

<ToggleButton
    android:id="@+id/toggleButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/radioGroup1"
    android:text="ToggleButton" />
</RelativeLayout>

```

*As principais propriedades que podem ser utilizadas nos componentes de tela são:*  
**id** - identificador do componente, usado para chamá-lo a partir de referências;

**layout\_width** – definição de como será a largura do componente. Para preencher a tela inteira é utilizado “fill\_parent”, caso a necessidade é de adaptação ao tamanho do conteúdo, “wrap\_content”;

**layout\_heigh** – similar ao “layout\_width”, porém, definição de altura do componente;

**text** - propriedade que corresponde ao texto exibido no componente;

**src** – endereço de localização da imagem a ser carregada no componente;

**style** - local onde é definido o estilo do botão;

**orientation** – definição em um “ViewGroup” (como o “RadioGroup”) de como será distribuídas as “View”, verticalmente ou horizontalmente;

**textOn** - atributo do “ToggleButton” que define o texto a ser exibido quando a flag estiver marcada;

**textOff** - atributo do “ToggleButton” que define o texto a ser exibido quando a flag estiver desmarcada;

**gravity** – define como é a distribuição de alinhamento do componente perante seu pai (centralizado, topo, rodapé).

A documentação completa pode ser encontrada em:

<http://developer.android.com/guide/topics/ui/index.html>



## 4.3. Menus

*Existem três tipos de menus para aplicações Android. O **menu de opções** é aberto ao utilizar o botão correspondente à abertura do menu (geralmente botão físico) e é referente ao **menu principal** da atividade em execução. Já o menu de contexto é aberto ao tocar e manter pressionado sobre um componente. O menu suspenso é semelhante ao menu de contexto, mas é aberto automaticamente ao simples toque em um componente que o executa (como visto no exemplo, utilizado pelo “Spinner”).*

*Para criar um menu de opções, podem ser feitas duas implementações. A primeira, a partir de um menu gerado em XML (armazenado em “res/menu/”) e amarrado à atividade a partir do comando “inflate()”.*

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game" />
    <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

*Observa-se a utilização de três elementos principais no item de menu declarado no XML. O “id” é utilizado para referenciar na Classe “R” (no código na atividade é possível observar a utilização), “icon” é referente à imagem em “res/drawable/” nas diferentes qualidades de resolução e “title” é referente ao texto exibido.*

*Para mapear a seleção de um item, pode ser utilizado o seguinte código, também na atividade em execução:*

```
@Override
```

```

public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

*A outra forma de criar um menu de opções (principal) é diretamente a partir da atividade, sem a manutenção do arquivo XML, porém isso torna mais complicada a referência, comprometendo a referência externa, caso necessário.*

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);

    // Create and add new menu items.
    MenuItem itemAdd = menu.add(0, ADD_NEW, Menu.NONE,
                                R.string.add_new);
    MenuItem itemRem = menu.add(0, REMOVE, Menu.NONE,
                                R.string.remove);

    // Assign icons
    itemAdd.setIcon(R.drawable.add);
    itemRem.setIcon(R.drawable.remove);

    // Allocate shortcuts to each of them.
    itemAdd.setShortcut('0', 'a');
    itemRem.setShortcut('1', 'r');
    return true;
}

```

*A criação de um menu de contexto é realizada de forma semelhante, mudando apenas as chamadas de método;*

```

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}

```

```
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

*Os menus de contexto não suportam ícones ou teclas de atalho.*

*É possível, ainda, a criação de sub-menus. Para isso, pode-se criar tags “<menu/>” recursivas no XML referente.*

*A documentação completa pode ser encontrada em:*

<http://developer.android.com/guide/topics/ui/menus.html>

## 4.4. Janelas de Diálogo

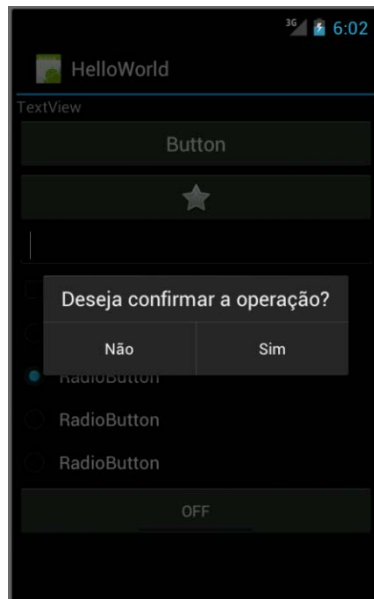
*Janelas de diálogo são aquelas que permitem ao usuário tomar uma decisão específica ou apenas ser informado de alguma mensagem. No Android, assim com o em Java para desktop, é muito simples de criar estas interfaces. Através de uma Classe construtora de diálogos, com métodos prontos para receber a personalização, torna-se bastante rápido e intuitivo incluir um diálogo na aplicação.*

*Da mesma maneira como é criado um “listener” de evento de clique, pode ser criado para a janela de diálogo.*

```
DialogInterface.OnClickListener dialogClickListener = new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        switch (which){
            case DialogInterface.BUTTON_POSITIVE:
                //botão SIM clicado
                break;
            case DialogInterface.BUTTON_NEGATIVE:
                //botão NÃO clicado
                break;
        }
    }
};
```

*Este é o código referente à manipulação da decisão do usuário ao tocar em algum dos botões. Depois, é preciso inserir o código de chamada de criação da janela de diálogo, que pode ser inserido no evento de clique de algum elemento, por exemplo.*

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Confirma operação?").setPositiveButton("Sim",
dialogClickListener).setNegativeButton("Não",
dialogClickListener).show();
```



*Este modelo de janela de diálogo consiste em utiliza a interface padrão da versão do sistema Android (como outros componentes de layout funcionam). Pode-se analisar os principais métodos das Classes de diálogo para criação de uma caixa de diálogo padrão:*

***setIcon()*** – método onde é definido um ícone para a caixa de diálogo;

***setMessage()*** – método onde é definida a mensagem exibida na caixa de diálogo;

***setTitle()*** – método onde é definido o título da caixa de diálogo;

***setNeutralButton()*** – definição de um botão neutro (normalmente rotulado com o título “OK”) e seu evento de ação;

***setPositiveButton()*** - definição de um botão positivo (normalmente rotulado com o título “Sim” ou “Yes”) e seu evento de ação;

***setNegativeButton()*** - definição de um botão negativo (normalmente rotulado com o título “Não” ou “No”) e seu evento de ação.

*Caso seja necessário utilizar um layout específico para uma janela de diálogo, o mesmo pode ser criado na pasta “res/layout/” e associado a um diálogo na atividade específica.*

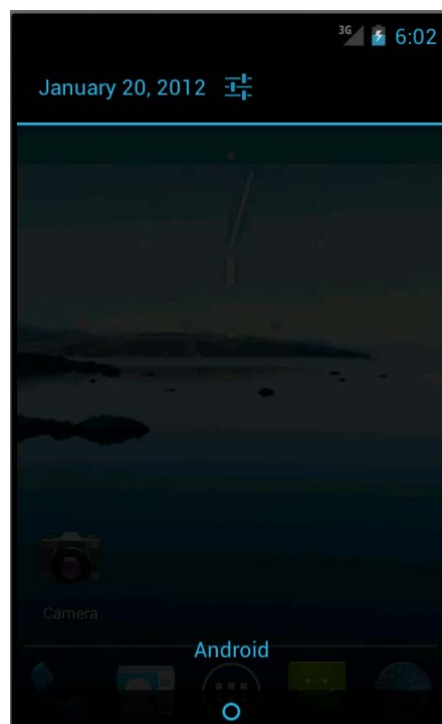
```
Dialog dialog = new Dialog(main.this);
dialog setContentView(R.layout.meuDialogo);
dialog.setTitle("Título da janela");
dialog.setCancelable(true); // fornece a opção de cancelar a partir
de uma ação (como um botão)

dialog.show();
```

## 4.5. Notificações

*Uma das vantagens do sistema operacional Android é a qualidade da sua multitarefa. Para melhor controle das tarefas, a interface contém uma barra de tarefas (também chamada de barra de status), por padrão situada na parte superior da tela. Com um gesto de deslize com o dedo para baixo, é possível expandir esta barra e visualizar os detalhes das aplicações abertas. Os elementos na barra de status do Android são chamados de notificações e servem para, dentre outras coisas, “alertar” o usuário sobre alguma pendência em alguma aplicação, ou simplesmente avisar sobre alguma novidade.*

*Utiliza-se notificação ao invés de janelas de diálogo quando uma aplicação, rodando em segundo plano, deseja comunicar-se com o usuário sem interromper a atividade dele no momento, dando a ele a liberdade de responder à aplicação dona da notificação no momento em que desejar.*



*O método mais simples de criar uma notificação totalmente customizável é a partir da criação de uma classe “helper” específica, contendo variáveis e constantes pré-definidas (algumas mutáveis) para auxiliar a criação da notificação sem a necessidade de preocupação com os diversos parâmetros utilizados, realizando assim o encapsulamento.*

```
public class Notificacao {
```

```

public static final int TIPO_ACTIVITY = 1;
public static final int TIPO_BROADCAST = 2;
public static final int TIPO_SERVICE = 3;
public static final String INTENT_STRING_MSG_STATUS = "MSGs";
public static final String INTENT_STRING_MSG = "MSG";
public static final String INTENT_STRING_TITULO = "titu";
public static final String INTENT_LONG_QUANDO = "WHEN";
public static final String INTENT_INT_ICONE = "icone";
public static final String INTENT_INT_FLAGS = "FLS";

public static Notification criarNotificacao(Activity activity,
Intent param, Intent acao, int tipo) {
    //cria a notificacao
    Notification n = new
Notification(param.getIntExtra(INTENT_INT_ICONE, 0),
                param.getStringExtra(INTENT_STRING_MSG_STATUS),
                param.getLongExtra(INTENT_LONG_QUANDO,
System.currentTimeMillis()));

    //Responsavel pela acao caso o usuario selecione a
notificacao;
    PendingIntent p;
    if (tipo == TIPO_ACTIVITY) {
        p = PendingIntent.getActivity(activity, 0, acao,
param.getIntExtra(INTENT_INT_FLAGS, 0));
    } else if (tipo == TIPO_BROADCAST) {
        p = PendingIntent.getBroadcast(activity, 0, acao,
param.getIntExtra(INTENT_INT_FLAGS, 0));
    } else if (tipo == TIPO_SERVICE) {
        p = PendingIntent.getService(activity, 0, acao,
param.getIntExtra(INTENT_INT_FLAGS, 0));
    } else {
        throw new IllegalArgumentException("tipo indefinido");
    }

    //Vincula o PendingIntent com a notificacao
    n.setLatestEventInfo(activity,
param.getStringExtra(INTENT_STRING_TITULO),
param.getStringExtra(INTENT_STRING_MSG), p);

    //Define valores default para a notificacao ex. som,vibra.
    n.defaults = Notification.DEFAULT_ALL;

    return n;
}

```

```

    public static NotificationManager notificar(Activity activity,
Notification notification, int id) {
        //pega serviço de notificacao
        NotificationManager nm = (NotificationManager)
activity.getSystemService(Activity.NOTIFICATION_SERVICE);
        //exibe a notificacao
        nm.notify(id, notification);
        return nm;
    }
}

```

*É interessante analisar a utilização do Objeto “PendingIntent”, que indica a dependência de uma chamada via “Intent”. Este objeto tem o poder de aguardar a utilização de um “Intent” a partir de uma atividade ou um serviço, realizando recebimentos e chamadas de parâmetros.*

*Após isso, basta na atividade ou serviço desejado instanciar o “helper” e realizar a chamada de notificação.*

```

Intent it = new Intent();
it.putExtra(Notificacao.INTENT_STRING_MSG, "HelloWorld em
execução!");
it.putExtra(Notificacao.INTENT_STRING_MSG_STATUS, "HelloWorld!");
it.putExtra(Notificacao.INTENT_STRING_TITULO, "HelloWorld!");
it.putExtra(Notificacao.INTENT_LONG_QUANDO,
System.currentTimeMillis());
it.putExtra(Notificacao.INTENT_INT_ICONE,
android.R.drawable.ic_dialog_alert);

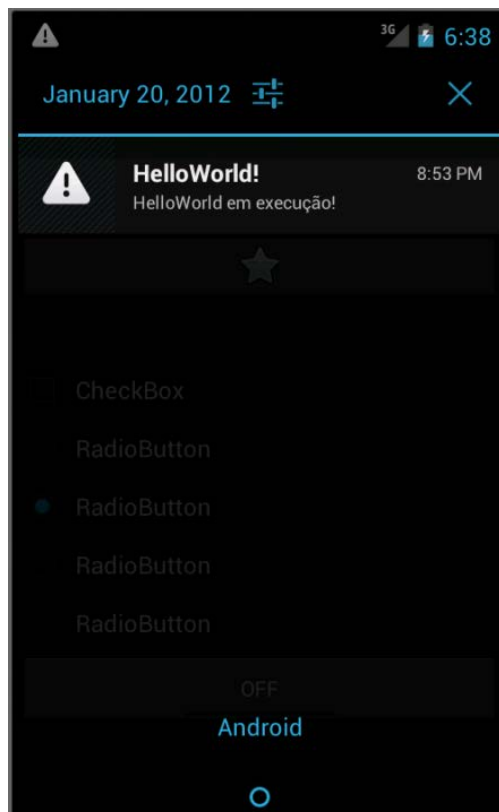
String url = "http://www.uzzye.com";
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse(url));

Notification notf = Notificacao.criarNotificacao(this, it, i,
Notificacao.TIPO_ACTIVITY);
Notificacao.notificar(this, notf, 550);

```

*Analisando o código, pode-se observar a passagem de parâmetros para o “Intent”, como a mensagem para a barra de status, o título da notificação, quando a notificação deve ser executada, o ícone (utilizado o padrão de alerta), além da configuração de ação utilizada ao tocar na notificação.*





*Para cancelar a exibição, pode ser utilizado o método “cancel()” da notificação.*

*Outro método de notificação de usuário muito comum em Android é através do componente “Toast”. Toast é um facilitador de mensagens rápidas, contendo somente texto, com exibição e ocultamento automáticos, sem interferência do usuário. O código é bastante simples e de fácil implementação.*

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();

toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0); // para posicionar
```

*É possível, ainda, montar um “Toast” personalizado, com um layout pré-definido a partir de código XML e da utilização do componente de layout “LayoutInflater”.*

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.toast_layout,
                             (ViewGroup)
findViewById(R.id.toast_layout_root));

ImageView image = (ImageView) layout.findViewById(R.id.image);
```

```
image.setImageResource(R.drawable.android);  
TextView text = (TextView) layout.findViewById(R.id.text);  
text.setText("Este é um Toast personalizado!");  
  
Toast toast = new Toast(getApplicationContext());  
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);  
toast.setDuration	Toast.LENGTH_LONG);  
toast.setView(layout);  
toast.show();  
  
toast.show();
```

*A documentação completa do “Toast” pode ser encontrada em:*

*<http://developer.android.com/guide/topics/ui/notifiers/toasts.html>*

## 4.6. Drag and Drop

*O sistema operacional Android possui uma interface muito responsiva a toques, capaz de realizar operações de arrastar e soltar e até de múltiplos toques (mais de um dedo), a fim de melhorar a experiência de usuário em aplicações mais robustas. Como exemplo, a galeria de fotos padrão do Android, em versão acima de 2.1, suporta o comando de pinça com dois dedos para aproximar ou afastar a imagem (zoom) e o navegador de Internet possui a navegação pela janela a partir de arrastar e soltar. Obviamente, o hardware deve ter suporte em tela a múltiplos toques para executar a pinça e ser sensível a toque para executar o mesmo.*

*A versão de biblioteca de arrastar e soltar para desenvolvedores Android foi disponibilizada apenas em 2011, o que gerou suporte apenas para versões superiores à 2.1 do Android, não podendo ser utilizado em dispositivos com o sistema operacional em versão mais antiga.*

*A utilização deste framework é um tanto quanto complicada, pois requer muito estudo de posicionamento e das diversas situações do toque, desde o iniciado até o finalizado. É possível realizar um simples movimento de arrastar e soltar componentes em tela e até criar containers como áreas de recepção de elementos arrastados, podendo assim criar, por exemplo, listas personalizadas a partir destes elementos.*

*A referência completa pode ser encontrada em:*

<http://developer.android.com/guide/topics/ui/drag-drop.html>

*Como exemplo prático, pode ser criado um simples “Drag and Drop” pela tela. Para isso, cria-se um container no layout e uma “View” para ser arrastada.*

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/linearLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="90dp"
        android:src="@android:drawable/dialog_holo_light_frame"
    />
```

```
</LinearLayout>
```

*Após isso, é necessário criar um controlador de arrastar e soltar, derivando do “ViewGroup” específico utilizado como container e contendo o método “onInterceptTouchEvent()”.*

```
public class TouchInterceptor extends LinearLayout {

    public TouchInterceptor(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {
        return super.onInterceptTouchEvent(ev);
    }
}
```

*Na atividade a ser utilizado o arrastar e soltar é necessário referenciar ao controlador criado, o que pode ser feito diretamente no “onCreate()”.*

```
ViewGroup container = (ViewGroup)findViewById(R.id.linearLayout1);
container.setOnTouchListener(new View.OnTouchListener() {

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch(event.getActionMasked())
        {
            case MotionEvent.ACTION_MOVE:
                int x = (int)event.getX() - offset_x;
                int y = (int)event.getY() - offset_y;

                int w =
getWindowManager().getDefaultDisplay().getWidth() - 100;
                int h =
getWindowManager().getDefaultDisplay().getHeight() - 100;
                if(x > w)
                    x = w;
                if(y > h)
                    y = h;

                LinearLayout.LayoutParams lp = new
LinearLayout.LayoutParams(
                new ViewGroup.MarginLayoutParams(
                LinearLayout.LayoutParams.WRAP_CONTENT,
                LinearLayout.LayoutParams.WRAP_CONTENT)
                );
```

```

        lp.setMargins(x, y, 0, 0);
        selected_item.setLayoutParams(lp);
        break;

        default:
            break;
    }
    return true;
}
});

ImageView img = (ImageView)findViewById(R.id.imageView1);
img.setOnClickListener(new View.OnClickListener() {

    @Override

    public boolean onTouch(View v, MotionEvent event) {
        switch(event.getActionMasked())
        {
            case MotionEvent.ACTION_DOWN:
                offset_x = (int)event.getX();
                offset_y = (int)event.getY();
                selected_item = v;
                break;

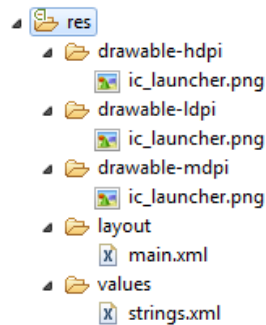
            default:
                break;
        }
        // retorna falso para evitar indesejados MotionEvent
        // e dar ao pai a chance de utilizar MotionEvent
        return false;
    }
});

```

*A partir deste código, podem ser utilizados outros métodos de toque, medição de coordenadas, além de definir outros containers, definir sobreposição e gerar componentes melhores de UX (User Experience), como listas em duas colunas com atribuição por arrastar e soltar.*

## 4.7. Recursos

*Em todos os elementos de layout utilizados até agora, no back-end é possível realizar acesso direto através da classe de referências “R.class”. Esta classe é chamada de “**Resource**”, pois referencia todos os elementos XML utilizados na aplicação. Estes elementos são armazenados dentro do diretório “res/”.*



*Após o salvamento de um arquivo XML ou compilação do projeto, a Classe “R” é gerada novamente, realizando as mudanças necessárias nas referências. Caso um elemento adicionado em um XML não conste nesta Classe, pode ser necessário realizar o “Clean” e o rebuild do projeto.*

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int imageView1=0x7f050001;
        public static final int linearLayout1=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

*A Classe de recursos é composta de subclasses, algumas das quais são referentes às pastas internas do diretório “res/”.*

***drawable*** – responsável pelos elementos da pasta “drawable”. É detectada automaticamente a qualidade de resolução do dispositivo (ou forçado através de código). Dessa forma, a referência também é realizada automaticamente, através da chamada “drawable”, sem especificação de qual subpasta utilizar;

***layout*** – define as referências diretas aos arquivos XML dentro da pasta “layout”;

***string*** – define as referências aos elementos declarados nos arquivos XML contidos na pasta “values”;

***id*** – define as referências diretas a todos os componentes visuais declarados nos arquivos XML de layout. Referência esta definida através do valor da propriedade “id” do componente. Ex.: “@+id/button1”, criará a referência “button1”.

Para utilizar as referências, de qualquer Classe pode ser realizada a chamada “R.subclasse.referência”, como por exemplo “R.layout.main”.

## 5. Conceitos Avançados

*Após o aprendizado sobre o desenvolvimento básico em Android, pode-se aprofundar nos conhecimentos mais avançados.*

*Dentre as principais execuções que podem ser realizadas em Android, dependendo do dispositivo anfitrião da versão do sistema, deve ser considerado o uso do acelerômetro e do giroscópio (sistemas de controle de posicionamento e angulação do dispositivo), o suporte a GPS (Global Positioning System), o suporte nativo a banco de dados SQL (Structured Query Language), no caso o SQLite, etc.*

*Nos itens a seguir, o detalhamento destes principais recursos.*

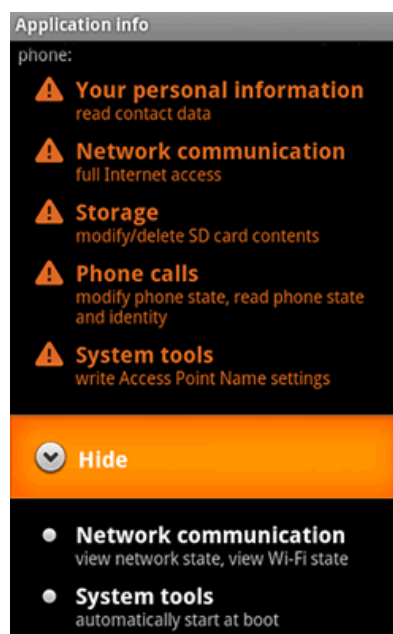


## 5.1. Permissões e Segurança

*Ao realizar um download de aplicativo no Google Play, é solicitada a confirmação de segurança, que consiste em descrever para serem aceitos todos os recursos exigidos pelo app.*

*No decorrer desta apostila, foram analisados vários recursos, incluindo de integração com outros apps, coletas de dados, utilização de recursos do próprio dispositivo onde o sistema está embarcado e armazenamento. Para cada recurso “estranho” ao app utilizado é necessário realizar um registro de permissão, caso contrário o sistema poderá parar inesperadamente e não conseguir finalizar sua execução corretamente.*

*Este registro de solicitação de permissão é feito no manifesto e pode ser alterado a qualquer momento, porém novas alterações implicam em novas aceitações de solicitação de permissão ao usuário baixar o aplicativo no Play.*



*No exemplo acima, o app requer que sejam acessadas informações de contato, que tenha acesso à Internet completo, que possa gerenciar conteúdo no cartão de memória SD, que possa modificar o estado do telefone (como com rede, sem rede e ocupado), e escrever um novo nome de “Access Point”.*

*É importante entender que, ao usuário aceitar as permissões de acesso de um app ao realizar o download, ele está automaticamente isentando a Google de qualquer culpa em problemas gerados por ele. Por exemplo, se um aplicativo realiza ligações telefônicas em segundo plano e o usuário aceitou essas permissões (contanto que estivessem*

*explícitas como na imagem acima), a Google não tem responsabilidade sobre os custos gerados por essas ligações. Exatamente por isso existe este sistema de controle de permissão e segurança. Para monitorar melhor o que os apps fazem em um dispositivo, pode ser interessante utilizar antivírus, disponíveis inclusive no Play.*

*Para definir uma solicitação de permissão de um aplicativo, basta incluir o seguinte código no Manifesto (diferentemente das outras declarações, como de atividades ou intenções, as de permissões e funcionalidades devem ser feitas fora da tag “<application/>”, mas ainda dentro da “<manifest/>”).*

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-feature android:name="android.hardware.camera.autofocus"
android:required="false" />
```

*A diferença entre declaração de permissão (permission) e de funcionalidade (feature) é que estas tendem a ser mais específicas e não comprometem a segurança do usuário, sendo geralmente algum recurso interno de algum item do dispositivo (como o “autofoco” da câmera).*

*Ao definir uma permissão simples, como no exemplo acima, a descrição e o nível de segurança utilizados são os padrões. Para customizar, é possível utilizar atributos específicos das tags.*

**android:label** – define o título da solicitação.

**android:description** – define o texto de conteúdo da solicitação.

**android:permissionGroup** – define o grupo que a solicitação pretence. Exemplo: "android.permission-group.COST\_MONEY";

**android:protectionLevel** – define o nível de proteção da permissão. Exemplo: "dangerous" .

*É interessante lembrar que, para utilização de atributos customizados, podem ser utilizadas constantes definidas no diretório de recursos “values”, utilizando a chamada “@string/nomeDaConstante”.*

*A lista completa de permissões pré-definidas, grupos e níveis de proteção para serem utilizados no manifesto, pode ser encontrada em:*

<http://developer.android.com/guide/topics/security/security.html>

## 5.2. Preferências de Usuário

*Em um website que requer autenticação, como um portal ou uma área restrita, geralmente existe uma opção para mantermos o último usuário e até mesmo a senha de login. Muitas vezes, o próprio navegador web realiza isso. Geralmente, esse armazenamento de preferência é realizado com um recurso chamado “Cookie”, presente em todos os navegadores modernos e de grande utilidade para rápida navegação.*

*Em Android, também existem objetos capazes de realizar o armazenamento e a recuperação das preferências de usuários, como o “SharedPreferences”.*

*De forma bem simples para codificação, semelhante ao uso de “Cookies” na web, um gerenciamento de preferências pode ser programado da seguinte forma. Como atributo da atividade, é necessário criar um objeto de preferências compartilhadas.*

```
private SharedPreferences sharedPreferences;
```

*Em seguida, pode ser implementado um método para armazenamento das preferências com o seguinte código.*

```
sharedPreferences = getSharedPreferences("PREFS_PRIVATE",
Context.MODE_PRIVATE);

Editor prefsPrivateEditor = sharedPreferences.edit();

prefsPrivateEditor.putString("nome", "João");
prefsPrivateEditor.putString("email", "joao@meuemail.com.br");
prefsPrivateEditor.putString("apelido", "Jão");

prefsPrivateEditor.commit();
```

*Por fim, para realizar a recuperação das preferências, pode-se implementar um método com o seguinte código.*

```
sharedPreferences = getSharedPreferences("PREFS_PRIVATE",
Context.MODE_PRIVATE);

String nome = sharedPreferences.getString("nome", "");
String email = sharedPreferences.getString("email", "");
String apelido = sharedPreferences.getString("apelido", "");

sharedPreferences = null; // utilizado somente em caso de necessidade
de limpar as preferências salvas
```

*É interessante observar a utilização do parâmetro “PREFS\_PRIVATE”, que define uma string de referência ao arquivo utilizado para armazenamento das preferências, e o “Context.MODE\_PRIVATE”, que define que somente a aplicação atual pode ter acesso às preferências. Para permitir o acesso a partir de outras aplicações, pode-se usar “MODE\_WORLD\_READABLE”, que permite apenas leitura, ou “MODE\_WORLD\_WRITEABLE”, que permite apenas escrita, além de estas poderem ser concatenadas com o operador de “String”, “+”.*

*Outro ponto a observar é a utilização de “getString” e “putString” para controle dos parâmetros. Pode ser utilizado qualquer tipo primitivo, além do objeto String, como “putInt”, ou “getBoolean”.*

### 5.3. Persistência em Banco de Dados (SQLite)

*O gerenciamento de preferências de usuário é muito útil em caso de controle rápido e simples, porém, em aplicações maiores faz-se necessário o uso de uma arquitetura de dados mais completa. Para isso, Android conta com um excelente banco de dados exclusivo para plataformas móveis e embarcadas.*

*Baseado na linguagem estruturada SQL, comum em bancos de dados como MySQL e Oracle, o SQLite é uma plataforma com recursos resumidos, mas segura e capaz de executar praticamente qualquer necessidade de persistência que uma aplicação complexa pode ter. Uma aplicação que requer uso de banco de dados não necessita de solicitação de permissões explícita, uma vez que SQLite é um recurso integrado e seguro por si só. Ao criar um banco de dados a partir de um app, o seguinte caminho físico é criado no sistema de arquivos do Android.*

*DATA/data/APP\_NAME/databases/FILENAME*

*A maneira mais fácil e robusta de construir uma aplicação com SQLite integrado é utilizando um “helper” para conexão de dados, semelhante ao que é feito em aplicações Java com JDBC.*

```
public class DataBaseHelper extends SQLiteOpenHelper {

    private static final String DBNAME = "meuDB";
    private DataBaseHelper myDBHelper;
    private SQLiteDatabase myDB;

    private final Context myContext;

    public DataBaseHelper(Context context) {
        super(context, DBNAME, null, 2);
        this.myContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    }

    public DataBaseHelper open() throws SQLException {
```

```

    myDBHelper = new DataBaseHelper(myContext);
    myDB = myDBHelper.getWritableDatabase();
    return this;
}

public void close() throws SQLException {
    this.myDB.close();
}

public SQLiteDatabase getDB()
{
    return this.myDB;
}
}

```

*A partir desta Classe auxiliar, pode-se realizar uma chamada simples de instância execução de comandos para o SQLite.*

```

DataBaseHelper dbHelper = new DataBaseHelper(this);
dbHelper.open();
SQLiteDatabase dbCon = dbHelper.getDB();

int id = 999;
Cursor cursor = dbCon.rawQuery("SELECT * FROM nomeTabela WHERE (id = ?)", new String[] {String.valueOf(id)});
startManagingCursor(cursor);
if(cursor.getCount()>0)
{
    cursor.moveToFirst();
    String valorColuna =
cursor.getString(cursor.getColumnIndex("nomeColuna"));
}

```

*Neste exemplo, é realizado um comando “SELECT” para recuperação de dados de uma entidade (tabela), a partir da comparação de uma coluna específica (no caso a “id”). A passagem de parâmetros SQL em Java pode ser feita através de uma lista de “String” ordenada conforme a aparição das solicitações (“?”). Para trabalhar com recuperação de dados SQL em Android/Java é utilizado o conceito de cursores, onde o cursor é posicionado em linhas dentro da tabela de retorno de resultado e pode ser posicionado livremente, avançando ou recuando.*

*Já para inserção de dados, por exemplo, pode ser utilizado o seguinte código:*

```
DataBaseHelper dbHelper = new DataBaseHelper(v.getContext());
dbHelper.open();
SQLiteDatabase dbCon = dbHelper.getDB();
String sqlCmd = "INSERT INTO nomeTabela (coluna1, coluna2) VALUES
(?,datetime('now'))";
dbCon.execSQL(sqlCmd,new String[] { "João" });
```

*É importante sempre depois de utilizar uma conexão com o banco de dados fechá-la, o que pode ser realizado com o comando “dbHelper.close();”.*

*Como em outras plataformas de desenvolvimento SQL, também é possível realizar a manutenção do banco de dados visualmente, através da visualização de uma tabela, com linhas e colunas representando a estrutura e os registros de uma entidade no banco de dados ([http://www.softsland.com/sqlite\\_expert\\_personal.html](http://www.softsland.com/sqlite_expert_personal.html)), através de linha de código em “Shell” com o ADB (Android Debug Bridge, <http://developer.android.com/guide/developing/tools/adb.html>), ou até mesmo através de um aplicativo instalado diretamente no dispositivo ([https://play.google.com/store/apps/details?id=com.speedsoftware.sqleditor&hl=pt\\_BR](https://play.google.com/store/apps/details?id=com.speedsoftware.sqleditor&hl=pt_BR)).*

*A documentação completa do SQLite em Android pode ser encontrada em:*

*<http://developer.android.com/reference/android/database/sqlite/package-summary.html>*

*A documentação completa dos comandos SQL pode ser encontrada em:*

*<http://www.sqlite.org/lang.html>*



## 5.4. Câmera e Arquivos

*Algumas vezes, dependendo do estilo de aplicativo que se está desenvolvendo, pode ser necessária (ou apenas um recurso interessante) a incorporação de uma câmera ao código. A melhor maneira de fazer isso é através da chamada de intenção.*

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, TIRAR_FOTO);
```

*Onde a constante “TIRAR\_FOTO” é apenas um número inteiro identificador de requisição. Ela é usada para identificar o resultado da ação após a execução da câmera.*

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    if (requestCode == TIRAR_FOTO) {
        if (resultCode == RESULT_OK) {
            Bitmap bitmap = (Bitmap) data.getExtras().get("data");
        } else if (resultCode == RESULT_CANCELED) {
        } else {
        }
    }
}
```

*Com o código acima, podemos identificar a definição de ação após executar um resultado específico, como o “TIRAR\_FOTO”, constante definida anteriormente. A partir disso, é possível verificar se a atividade foi concluída corretamente, cancelada ou simplesmente finalizada.*

*Depois de termos um Objeto do tipo “Bitmap” criado a partir da câmera, pode-se realizar o salvamento do arquivo em disco.*

```
String externalStoragePath =
Environment.getExternalStorageDirectory().getAbsolutePath()+"/download";
File sdImageMainDirectory = new File(externalStoragePath);
FileOutputStream fileOutputStream = null;
String nameFile;
try {
    fileOutputStream = new FileOutputStream(
sdImageMainDirectory.toString() +"/image.jpg");
    BufferedOutputStream bos = new BufferedOutputStream(
                                fileOutputStream);
    bitmap.compress(CompressFormat.JPEG, quality, bos);
}
```

```

        bos.flush();
        bos.close();
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

*Basicamente, são utilizadas constantes pré-definidas para manutenção de arquivos a partir de “Stream”. Após isso, é importante referenciar no manifesto a utilização tanto da câmera quanto do gerenciamento de arquivos.*

```

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

```

*Um exemplo de código interessante é o de upload de uma imagem tirada da câmera para a web.*

```

String serverUrl = "http://www.uzzye.com/android/";
String urlString = serverUrl + "android_file_upload.php";

try {
    InputStream
fis=this.getContentResolver().openInputStream(externalStoragePath +
"/image.jpg");
    HttpFileUploader htfu = new HttpFileUploader(urlString,"",
SELECTED_IMAGE.toString(),imageName);
    htfu.doStart(fis);
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

```

*Para realizar postagem de conteúdo em formato de texto para uma página web (através de métodos POST/GET), utiliza-se os componentes “HttpClient” e “HttpPost”.*

```

HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost(serverUrl + "android_post.php");

try {
    List<BasicNameValuePair> nameValuePairs = new
ArrayList<BasicNameValuePair>(2);
    nameValuePairs.add(new BasicNameValuePair("id","0"));
}

```

```
    httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

    // Execute HTTP Post Request
    HttpResponse response = httpclient.execute(httppost);
} catch (ClientProtocolException e) {
    // TODO Auto-generated catch block
} catch (IOException e) {
    // TODO Auto-generated catch block
}
```

## 5.5. GPS e Mapas

*Dentre as API disponibilizadas pela Google juntamente ao SDK, existe uma muito comum, a do Google Maps. É possível combinar a utilização dos mapas com o sistema de posicionamento global (GPS), a fim de aprimorar a experiência de usuário e fornecer localidades próximas, ou o posicionamento exato de algum objeto. Um exemplo de utilização comum é, além do próprio aplicativo Google Maps, aplicativos de marcação de presença (checkin), como o Foursquare. Mesmo assim, os componentes para utilização de GPS são independentes e podem ser utilizados sem a combinação com mapas.*

*A utilização do GPS requer a seguinte definição de uma solicitação de permissão.*

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

*Para monitorar a mudança de localização a partir de comandos de GPS, é necessário utilizar o componente “LocationManager”.*

```
public class GPS extends Activity
{
    private LocationManager lm;
    private LocationListener locationListener;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---use the LocationManager class to obtain GPS locations---
        lm = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);

        locationListener = new MyLocationListener();

        lm.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            0,
            0,
            locationListener);
    }
}
```

*Através da solicitação do método “requestLocationUpdates()” é criado um “listener” responsável por pegar todas as informações de mudança de posição geográfica. Pode-se notar o uso do provedor padrão de posicionamento, interno do próprio dispositivo, podendo ser configurado um GPS externo, caso necessário. Após, é necessário realizar a implementação da Classe “listener” utilizada.*

```
private class MyLocationListener implements LocationListener
{
    @Override
    public void onLocationChanged(Location loc) {
        if (loc != null) {
            Toast.makeText(getBaseContext(),
                "Location changed : Lat: " + loc.getLatitude() +
                " Lng: " + loc.getLongitude(),
                Toast.LENGTH_SHORT).show();

            // aqui pode ser inserido um código de “Maps”
        }
    }

    @Override
    public void onProviderDisabled(String provider) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProviderEnabled(String provider) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onStatusChanged(String provider, int status,
        Bundle extras) {
        // TODO Auto-generated method stub
    }
}
```

*Já para a utilização do “Maps” como recurso no Android é requerida a seguinte definição de uma solicitação de permissão no manifesto.*

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

*Ainda no manifesto, dentro da tag “<application/>”, é necessário especificar o uso da biblioteca de mapas da API.*

```
<uses-library android:name="com.google.android.maps" />
```

*No arquivo de layout da atividade, é necessário utilizar um componente de visualização de mapas (“MapView”).*

```
<com.google.android.maps.MapView
    android:id="@+id/mapview1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="apisamples" />
```

*É possível, no perfil de usuário desenvolvedor da Google, registrar uma chave de API para utilização do “Maps” e análise de dados utilizada pela aplicação. Essa chave de API pode ser utilizada na propriedade “apiKey” do componente de layout.*

*No código da Classe de atividade, é possível utilizar o controlador de mapa do componente visual e trabalhar com as propriedades dele.*

```
MapView mapView = (MapView) findViewById(R.id.mapview1);
MapController mc = mapView.getController();
```

*Integrando o “Maps” com o código anterior referente à utilização de GPS, é possível realizar a movimentação do mapa a partir do posicionamento geográfico do usuário.*

```
GeoPoint p = new GeoPoint(
    (int) (loc.getLatitude() * 1E6),
    (int) (loc.getLongitude() * 1E6));
mc.animateTo(p);
mc.setZoom(16);
mapView.invalidate();
```

*Para simular a troca de posicionament é possível, na guia “DDMS” do Eclipse, aba “Emulator Controls -> Location Controls”, modificar atributos de localização em tempo de execução e enviá-los ao emulador. O mesmo não é válido para depuração em dispositivos reais.*

## 5.6. Bluetooth

*Em algumas aplicações, pode ser necessária a transmissão de dados entre dispositivos, seja uma transferência de arquivo ou apenas texto. Isto pode ser realizado ativando o recurso de “Bluetooth” do Android, que também conta com uma biblioteca de Classes específicas para este tipo de transferências.*

*A utilização do bluetooth como recurso no Android requer a seguinte definição de uma solicitação de permissão no manifesto.*

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

*O primeiro passo para a utilização do bluetooth é verificar que o dispositivo possui o suporte necessário e se o recurso está ativado.*

```
BluetoothAdapter mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth
} else {
    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    }
}
```

*Com o bluetooth do dispositivo ativo, é necessário verificar o pareamento, que consiste em um dispositivo servidor da conexão receber e permitir a entrada de outro dispositivo nela.*

```
Set<BluetoothDevice> devices = adapter.getBondedDevices();
// Servidor Bluetooth
BluetoothDevice server = null;
String servername = "NomeDoServidor";
// Pesquisa o Servidor Pareado
for (BluetoothDevice device : devices) { // Laço de Busca
    if (servername.equals(device.getName())) { // Nomes Idênticos?
        server = device; // Dispositivo Encontrado e Selecionado
    }
}

if (server == null) {
```

```
// Mensagem de Erro ao Usuário
alert("Servidor não Pareado");
finish();
}
```

*No exemplo acima, a procura foi pelo nome do servidor bluetooth, que pode não ser único. Por isso, o ideal é realizar a busca pelo endereço MAC do dispositivo do servidor. Com o servidor pareado, é necessário realizar o código de comunicação bluetooth dentro de uma “Thread”, pois este tipo de conexão é bloqueável e depende de respostas do servidor. Para acessar o dispositivo através da “Thread”, é necessário definir uma variável como “final”, recebendo o dispositivo pareado.*

```
final BluetoothDevice computer = server;
```

```
Thread connector = new Thread(){
    public void run() {
        // Identificador Único do Servidor
        UUID ident = UUID.fromString("879c3537-ae66-4013-a677-
9b7e5339d13c");
        try {
            // Socket de Conexão
            BluetoothSocket s =
            // Conexão a Partir do Identificador Único
            computer.createRfcommSocketToServiceRecord(ident);
        } catch (IOException e) { // Erro de Entrada e Saída de Dados
            // Mensagem de Erro ao Usuário
            alert("Erro de Conexão");
            finish();
        }

        // Conectar ao Servidor
        s.connect(); // Execução Bloqueante
        // Fluxos de Entrada e Saída de Dados
        InputStream in = s.getInputStream();
        OutputStream out = s.getOutputStream();
    }
};
```

*A criação de um socket é necessária para a comunicação e esta é realizada através de um identificador único utilizado no servidor. Tanto o socket quanto os objetos de “streaming” devem ser armazenados em variáveis globais da atividade, para acesso em outros métodos. Após estes códigos, é possível manipular a transferência de dados, como, por exemplo, utilizando um método chamado no clique de um botão.*

```
final String conteudo = "Mensagem enviada para o servidor.";
Thread sender = new Thread(){
```



```
public void run() {  
    byte content[] = conteudo.getBytes();  
    try { // Possibilidade de Erro  
        output.write(content.length);  
        output.write(content);  
        alert("Texto Enviado");  
    } catch (IOException e) { // Erro Encontrado  
        alert("Erro na Transferência");  
    }  
}  
};  
sender.start(); // Inicialização
```

*Com os códigos acima é possível realizar uma conexão bluetooth em qualquer servidor de identificadores conhecido e a criação destes pode ser realizada em diversas linguagens, com diversos dispositivos.*

## 5.7. Sensores

*O sistema operacional Android, assim como muitos dos dispositivos que o possuem instalado, suporta alguns recursos de sensores, dentre os quais pode-se destacar o acelerômetro (responsável pela medição de aceleração aplicada a movimentos) e o giroscópio (responsável pela medição de movimentos de giro/inclinação). Estes sensores são utilizados para diversas aplicações, como jogos, ajustes de tela, melhorias de usabilidade, etc.*

	Acceler.	Magnetic	Gyroscope	Light	Pressure	Proximity	Temperature	Camera
Nexus One	x	x		x		x		x
HTC Incredible	x	x		x		x		x
HTC Desire	x	x		x		x		x
HTC Evo	x	x		x		x		x
Motorola Droid	x	x		x		x		x
Samsung Galaxy S	x	x	?	x		x		x
Garminfone	x	x						x
HTC Hero	x	x					?	x
HTC Droid Eris	x	x		x		x		x
Motorola CHARM	x			x		x		x
Motorola DROID™ 2	x	x		x		x		x
Samsung Epic	x	x				x		x
Samsung Captivate	x	x				x		x
Sony Ericsson Xperia X10	x	x				x		
Motorola Backflip	x							x

*No Java para Android, a captação do giroscópio é realizada através de “listener” de eventos.*

```
private void hookupSensorListener() {
    SensorManager sm =
    (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    sm.registerListener(sel,
        sm.getDefaultSensor(Sensor.TYPE_GYROSCOPE),
        SensorManager.SENSOR_DELAY_UI);
}

private final SensorEventListener sel = new SensorEventListener() {
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
            updateOrientation(event.values[0], event.values[1],
event.values[2]);
        }
    }
}
```

```

    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {}
};

private void updateOrientation(float heading, float pitch, float
roll) {
    // Update the UI
}

```

*Como nem todo dispositivo possui o suporte a giroscópio, é possível identificar esse suporte através do seguinte código.*

```

PackageManager paM = getPackageManager();
boolean gyroExists =
paM.hasSystemFeature(PackageManager.FEATURE_SENSOR_GYROSCOPE);

```

*De qualquer forma, é necessário especificar uma solicitação de permissão para utilização do giroscópio, o que atualmente filtra downloads no Google Play para os dispositivos suportados.*

```
<uses-feature android:name="android.hardware.sensor.gyroscope"/>
```

*Para utilização de acelerômetro, é possível utilizar a Classe “SensorManager”, responsável por métodos e atributos específicos de controle do recurso.*

```

public class SensorActivity extends Activity, implements
SensorEventListener {
    private final SensorManager mSensorManager;
    private final Sensor mAccelerometer;

    public SensorActivity() {
        mSensorManager =
(SensorManager) getSystemService(SENSOR_SERVICE);
        mAccelerometer =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
    }

    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }
}

```

```
}  
  
public void onAccuracyChanged(Sensor sensor, int accuracy) {  
}  
  
public void onSensorChanged(SensorEvent event) {  
}  
}
```

*A manutenção de dados obtidos pelos métodos recuperadores da sensibilidade requer algum conhecimento de física, pois os movimentos são realizados em um plano tridimensional, com coordenadas obtidas a partir de matrizes e definição de vetores. A documentação completa pode ser encontrada no seguinte endereço.*

*<http://developer.android.com/reference/android/hardware/SensorManager.html>*

## 5.8. OpenGL, games, redes sociais e tendências

*Com todos os principais recursos e facilidades disponibilizadas pelo Android estudados, é possível desenvolver uma aplicação completa e disponibilizá-la no Google Play. Mesmo assim, é possível incrementar ainda mais uma aplicação com recursos cada vez mais utilizados e em constante evolução. Um exemplo disso é o mercado de jogos. O número de games para Android cresce cada vez mais e com ele os recursos para produção desses jogos. Pode-se encontrar em distribuições livres muitos recursos, além de engines (motores de execução) de física, montadas sobre a codificação crua do OpenGL, recurso de licença aberta para produção de elementos tridimensionais em interface gráfica.*

*É possível obter mais informações sobre OpenGL no site oficial:*

[\*http://www.opengl.org/\*](http://www.opengl.org/)

*Além de estudar OpenGL, é interessante analisar o código-fonte das principais bibliotecas de engines de física gratuitas, para entender melhor a integração e o funcionamento do OpenGL com outros recursos.*

***AndEngine** – principal engine de física gratuita para Android, contém uma biblioteca de exemplo com aplicativos de demonstração e blog de atualização e suporte.*

[\*http://www.andengine.org/\*](http://www.andengine.org/)

***Cocos2D** – engine em andamento, migrada da versão oficial iOS para Android, fez muito sucesso nos dispositivos da Apple e tenta ganhar seu espaço no sistema da Google.*

[\*http://code.google.com/p/cocos2d-android/\*](http://code.google.com/p/cocos2d-android/)

*A documentação completa do OpenGL para Android pode ser encontrada no seguinte endereço.*

[\*http://developer.android.com/guide/topics/graphics/opengl.html\*](http://developer.android.com/guide/topics/graphics/opengl.html)

*Com os games, surgiram as necessidades de novos recursos para entretenimento, e a competitividade tornou-se o principal. Como nos jogos de outrora, onde os melhores jogadores eram identificados pelos recordes em pontuações, nos jogos para plataformas móveis muitas vezes é possível compartilhar os resultados nas redes sociais. Esta integração é muito comum, principalmente quando fala-se em Facebook. Dentre as principais integrações com a plataforma, estão o compartilhamento de conquistas e*

resultados e colaboração de amigos. Nos jogos da gigante Zynga, é comum a necessidade da participação de amigos na evolução do jogo de um usuário. No CityVille, por exemplo, o usuário precisa solicitar recursos aos amigos que, colaborando, acabam ganhando bônus em retorno. Tanto na versão incorporada no Facebook, quanto na versão móvel do jogo Zynga Poker, o jogador pode jogar contra amigos do Facebook ou contra outros usuários do mundo inteiro que possuem conta na rede social, além de poder fornecer prêmios para eles.

Não só de integrações com jogos vivem as redes sociais nos aplicativos móveis. Aplicativos como o Cardio Trainer, por exemplo, utilizam compartilhamento de treinos programados e executados no Twitter e Facebook, com direito a informações detalhadas do treino e mapa de execução. Outros aplicativos, como o TweetDeck, utilizam das redes sociais para o principal objetivo do aplicativo, integrar as redes em um único formato de informações.

A Facebook oferece um excelente portal de desenvolvimento para desenvolvedores de várias plataformas, com documentação completa das bibliotecas (com exemplos de código) e fórum de discussão.

<http://developers.facebook.com/?ref=pf>

<http://developers.facebook.com/docs/guides/mobile/>

A Twitter segue o mesmo exemplo e fornece uma plataforma bem completa.

<https://dev.twitter.com/>

Ambas as redes sociais fornecem recursos para usuários domésticos e corporativos, portanto, a integração entre aplicativos móveis e redes sociais pode fazer uma grande diferença nas estratégias de marketing digital.

Outras plataformas sociais oferecem recursos para desenvolvedores móveis e podem facilmente serem implantadas, fornecendo maior abertura na construção de novas ideias. LinkedIn, Foursquare, Instagr.am, são exemplos de plataformas sociais com recursos que podem agregar muito valor aos aplicativos a serem disponibilizados no Google Play.

Estas, aliás, são as principais tendências de mercado para as próximas gerações de dispositivos e versões do Android. Serão vistos cada vez mais jogos utilizando recursos de hardware e conexões sem fio, utilização de multi-toque, aplicativos de produção e estilo de vida conectados com plataformas sociais, informações armazenadas em nuvem e, principalmente motivados pela Google, produtores independentes de boas aplicações móveis. Graças à abertura a desenvolvedores e ao suporte com boa e clara

*documentação, a Google conquista cada vez mais um espaço maior entre os dispositivos móveis, sejam smartphones ou tablets, além de mais usuários utilizarem de forma mais profunda seu sistema operacional e desenvolvedores se interessarem pela plataforma perante os considerados “rivais” de mercado.*

*Android já é uma ótima opção de desenvolvimento, tornando-se uma ferramenta mais necessária a cada dia, tanto para usuários domésticos que procuram muitas opções de produtividade e entretenimento para seu dia-a-dia, quanto para grandes corporações que buscam novas estratégias de marketing. Conquistando mais fabricantes de dispositivos, adaptando-se às tendências de mercado e conquistando novos desenvolvedores, o Android é uma ferramenta poderosa que vale muito a pena ser estudada.*

## 4. Exercícios

*Este capítulo da apostila contém os exercícios para estudo e fixação referentes aos demais respectivos capítulos.*

### 1) Introdução Java

- 1.1) *Configurar ambiente de desenvolvimento Java.*
  - a) *Seguir as instruções de instalação e configuração do Java SDK;*
  - b) *Seguir as instruções de instalação e configuração da IDE Eclipse;*
- 1.2) *Criar um projeto Hello World Java.*
  - a) *Criar um novo projeto Java;*
  - b) *Inserir um código de escrita “Hello World”;*
  - c) *Criar um perfil de execução de projeto e executar;*
  - d) *Criar um perfil de depuração de projeto e executar;*

### 2) Introdução Android

- 2.1) *Configurar ambiente de desenvolvimento Android.*
  - a) *Seguir as instruções de instalação e configuração do Android SDK;*
  - b) *Seguir as instruções de instalação e configuração do Android Development Tools;*
- 2.2) *Criar um projeto Hello World Android.*
  - a) *Criar um novo projeto Java;*
  - b) *Inserir um código de escrita “Hello World”;*
  - c) *Criar um emulador compatível com a versão escolhida na criação do projeto;*
  - d) *Criar um perfil de execução de projeto e executar;*
  - a) *Criar um perfil de depuração de projeto e executar;*

### 3) Conceitos Básicos Android

- 3.1) *Iniciar a implementação de um aplicativo gerenciador de listas de tarefas (checklist), contemplando cadastro de listas, cadastro de tarefas, marcação de tarefas concluídas, compartilhamento de tarefas (e-mail e redes sociais), câmera para armazenamento de foto e marcação de localização.*



3.2) *Implementar duas primeiras atividades do aplicativo.*

a) *Atividade Principal, responsável pela inicialização do app, com definições das variáveis globais e constantes e layout associado contendo a listagem das listas cadastradas;*

b) *Atividade Cadastro de Lista, contendo formulário de cadastro com os campos “Título”, “Resumo”, “Data Meta”;*

3.3) *Implementar diversos eventos de controle do ciclo de vida das atividades, tratando “Pause”, “Resume”, “Start”, “Stop”.*

3.4) *Implementar a estrutura de banco de dados e métodos de persistência para armazenamento das listas, utilizando SQLite e SQLiteOpenHelper;*

3.5) *Preparar serviço para execução em segundo plano, que verifique data e hora atual para alertas de tarefa.*

#### **4) Interface Gráfica**

4.1) *Criar demais layouts para as telas do aplicativo:*

a) *Cadastro de Tarefas;*

4.2) *Implementar diversas telas, utilizando os principais tipos de layout estudados, combinando-os e analisando benefícios a serem implementados no aplicativo:*

a) *LinearLayout;*

b) *FrameLayout;*

c) *RelativeLayout;*

4.3) *Implementar diversos componentes, utilizando seus recursos e analisando benefícios a serem implementados no aplicativo:*

a) *TextView;*

b) *EditText;*

c) *CheckBox;*

d) *RadioButton;*

e) *ToggleButton;*

f) *Spinner;*

4.4) *Implementar avisos diversos e sobre as operações de inclusão, alteração e exclusão no aplicativo, utilizando os diversos recursos de alertas:*

a) *Toast;*

b) *AlertDialog;*

c) *Notification (implementar no serviço);*

4.5) *Criar um menu para o aplicativo:*

- a) Opções;*
- b) Sair;*

*4.6) Implementar sistema de tabulação de layouts no aplicativo, deixando todas as opções de tela visíveis, utilizando TabView;*

## **5) Conceitos Avançados**

*5.1) Implementar um layout contendo visualização de mapa:*

- a) Pesquisa de endereço no mapa;*
- b) Overlays contendo apontadores para localização de tarefas;*
- c) No evento de pressionar sobre o mapa, abrir tela de cadastro de tarefa para realizar associação com a localização informada;*

*5.2) Converter a estrutura de banco de dados do aplicativo em ContentProvider;*

*5.3) Implementar sistema de preferências de usuário para armazenamento das informações no menu opções;*

*5.4) Implementar sistema de armazenamento de imagem à tarefa, realizando a utilização da câmera através de um dos dois métodos apresentados:*

- a) Por Intent;*
- b) Por SurfaceView;*

*5.5) Implementar compartilhamento de tarefa:*

- a) Por E-mail;*
- b) Por Facebook;*

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

