**Curso de Dispositivos Móveis**

**Dispositivos Móveis**

**Aula 01 - Introdução ao Android**

**- Android**

É definido como um conjunto de software para dispositivos móveis com um sistema operacional, um middleware. Tem seu núcleo baseado no sistema operacional Linux.

Na programação para um ambiente Android a principal linguagem utilizada é a Java.

O Android trouxe para o mundo da programação a facilidade de acesso a funcionalidades dos dispositivos, como câmera e GPS, assim como acesso a qualquer parte do software nativo.

**- Programar para dispositivos móveis**

A programação para dispositivos móveis requer atenção a alguns detalhes:

\* Observar que a aplicação pode sofre interrupções a qualquer momento. Tratar eventos assíncromos.

\* A capacidade limitada do hardware.

\* Consumo de energia e descarga da bateria durante a execução do aplicativo.

**- Intents**

São mensagens que possibilitam o acesso a outros aplicativos que estão instalados no aparelho. Assim podemos criar sobre APIs para fazer uso de recursos como GPS, câmera ou utilizar recursos de outros softwares para envio de mensagens e e-mails, por exemplos.

**- Partes de uma arquitetura Android**

1º- Camada de Aplicações: É a camada mais externa onde ficam as aplicações nativas, que já vem instaladas no aparelho, e as aplicações instaladas pelos usuários. Essa aplicações tem Intents específicos que possibilitam o uso delas por outras aplicações.

2º – Camada Java API Framework: Essa camada permite ao desenvolvedor fazer uso direto do hardware do aparelho através de métodos pré-definidos.

3º - Camada Bibliotecas Nativas: São bibliotecas disponibilizadas pelo Android utilizada para exibição multimídia, SQLite, OpenGl ES, OpenCore entre outras. Nesta camada também se encontra o Android Runtime responsável pelas bibliotecas de suporte a linguagem Java e acesso ao Kernel do Linux.

4º - Camada de Abstração de Hardware (HAL): Permite ao desenvolvedor ter acesso as funcionalidades do hardware.

5º - Camada do Kernel do Linux: Camada não acessível ao desenvolvedor. Roda todos os processos, faz gerenciamento do hardware, threads e recursos críticos do sistema.

**- Activity**

É o componente do Android responsável por gerenciar as telas de uma aplicação, controla a interação do usuário com o sistema através da tela e botões do aparelho.

Uma aplicação pode ter uma ou mais activity e as activity dessa aplicação também pode se comunicar com activity de outras aplicações. Entretanto só uma activity pode estar ativa por vez.

Neste contexto deve-se entender o ciclo de vida de uma activity para a correta programação de seu funcionamento dentro da proposta da aplicação.

Para cada ciclo de vida de uma activity o Andróid faz a chamada de um método específico.

\* Método onCreate( ): é chamado pelo sistema quando uma activity é criada. Onde normalmente são iniciadas variáveis globais e os elementos da interface gráfica.

\* Método onStart( ): a activity é preparada para mostra sua tela de interação com o usuário.

\* Método onResume( ): é quando fica com o foco no aparelho. Depois disso a tela de interação é

mostrada ao usuário. Neste estado se a activity for interrompida parcialmente é chamado o método onPause( ), se a activity sair completamente da tela é chamado o método onStop( ).

\* Método onPause( ): é ativado quando a activity entra em segundo plano. Se foi interrompida por uma aplicação de mais alta prioridade, precisando de memória, quando o usuário retomar a aplicação é chamado o método onCreate( ), novamente. Se não foi retirado da memória, quando o usuário retomar a aplicação é chamado o método onResume( ).

\* Método onStop( ): a aplicação é parada. Se foi interrompida por uma aplicação de mais alta prioridade, precisando de memória, quando o usuário retomar a aplicação é chamado o método onCreate( ), novamente. Se não foi retirado da memória, quando o usuário retomar a aplicação é chamado o método onResume( ).

\* Método onDestroy( ): A activity está terminada ou sendo destruída pelo sistema e são liberados todos os recursos que estavam sendo alocados por ela.

**- Services**

Services são componentes que trabalham no sistema Android em segundo plano e não possui uma interface gráfica.

Um service também possui um ciclo de vida.

Existem dois tipos de Services: **Bounded Service** e **Unbounded Service**.

**Unbouded Service:**

Este serviço é iniciado por algum componente Android tendo após isso um ciclo de vida independente do componente que o iniciou podendo ficar ativo mesmo depois de este componente tive seu ciclo de vida terminado.

Seu ciclo de vida é:

\* Método onCreate( ): este método ocorre logo após o serviço ser chamado, nele os dados iniciais necessários para a execução do serviço são inicializados, sendo assim criado o serviço. Após isso chama o método onStartCommand( ).

\* Método onStartCommand( ): Neste método o serviço passa a ser executado até que haja uma interrupção do cliente ou seu tempo de vida chegue ao fim. Em qualquer dos casos é chamado o método onDestroy( ).

\* Método onDestroy( ): Este método é executado e o serviço é desligado.

B**ouded Service:**

Neste serviço existe uma ligação entre eles e outros componentes do Android havendo assim uma comunicação entre eles.

Há assim uma dependência deste serviço, sendo, portanto, destruído quando todas as conexões com eles são encerradas.

Seu ciclo de vida é:

\* Método onCreate( ): este método ocorre logo após o serviço ser chamado inicializando-o. Fica aguardando algum cliente se conectar a ele. Havendo uma conexão o método onBind( ) é chamado.

\* Método onBind( ): Este método prepara o serviço para receber o cliente, ficando o cliente apto a fazer uso do serviço. Se não houve mais nenhum cliente conectado o método onUnbind( ) é chamado.

\* Método onUnbind( ): Neste método o serviço verifica se foi configurado para ficar aguardando novas conexões, se sim, o método onCreate( ) é chamado. Se foi configurado para que não tendo conexão seja encerrado o método onDestroy( ) é chamado.

\* Método onDestroy( ): Esse método é executado e o serviço é encerrado.

**- Content Provinders**

É o componente responsável por armazenar, gerenciar e ou tornar disponível dados que são coletivos entre aplicações.

Há o content provinders que são nativos do Android e os que podem ser criados por aplicações não nativas. Eles tem a mesma forma de funcionamento tem a capacidade de compartilhar dados gerados por sua aplicação com outras aplicações.

Seu funcionamento é semelhante a um banco de dados relacional tendo as suas consultas executadas pelo Content Resolver através de seus métodos disponíveis, que são:

\* query( ): método responsável por realizar as consultas dentro do content provinders e retorná- las a quem fez a chamada.

\* insert( ): método responsável para fazer a inserção de novos dados no content provinders.

\* update( ): faz a atualização de dados.

\* delete( ): deleta um dado existente.

\* getType( ): informa que tipo de retorno deve ser esperado de um caminho de acesso ao provinders.

\* onCreate( ): responsável por todas as iniciações pertinentes ao content provinders quando criado.

**- Broadcast Receiver**

Este é o componente que recebe e gerencia todos os sistemas que são gerados pelo sistema e distribui para todas as aplicações interessadas.

São exemplos de broadcats gerados pelo sistema:

- Aviso de conexão a uma rede móvel ou rede wifi.

- Aviso de bateria fraca.

- Aviso de chamada de voz ou mensagem.

As aplicações não nativas também podem gerar seus próprios broadcasts.

Seu ciclo de vida é:

\* Método onReceive( ): este método é chamado quando um Broadcast a ser capturado ocorre. Após este método ser executado o Broadcast é encerrado.

**Aula 02 - Ambiente de desenvolvimento**

**- Instalação do Android**

\* Ter a instalação do JDK (java develop Kit) da versão 7 ou superior.

\* Fazer o download do Android Studio no site developer.android.com.

\* Fazer a instalação do Android Studio no computador.

\* Fazer configuração da versão do Android que irá trabalhar e configurar o dispositivo vritual (em AVD manager).

**- Estrutura de um projeto Android**

Quando se desenvolve uma aplicação Android deve-se respeitar uma estrutura de pastas, de códigos, recursos e configurações do aplicativo.

Dentro da pasta da aplicação teremos as seguintes pastas.

- Pasta manifests: é a pasta onde colocamos o arquivo que terá as informações na aplicação, o arquivo androidManifest.xml. Neste arquivo há informações como:

\* Versão da aplicação.

\* Desenvolvedor.

\* Versão mínima necessário do android para execução.

\* Permissões necessárias para a instalação.

\* Telas e serviços disponíveis.

- Pasta java: é onde são colocados os arquivos de classe Java da nossa aplicação.

- Pasta res: é a pasta onde são colocados os arquivos de configurações. Todos os arquivos que não são classes Java ficam nesta pasta. Dentro da pasta res encontra-se mais três subpastas:

\* Pasta drawable: nesta pasta ficam as imagens utilizadas na aplicação.

\* Pasta layout: é o local dos arquivos XMLs que formam os layouts das telas da aplicação.

\* Pasta values: contém os arquivos Strings e outros valores que poderão ser usados na aplicação. As strings internalizáveis e sons devem ser colocadas nesta pasta.

**Aula 03 - Interfaces Gráficas I**

Existem duas maneiras de se criar interfaces gráfica no desenvolvimento android, sendo que uma não substitui a outra, mas se completam.

\* Declarativa: Nesta maneira são declarados em um arquivo XML as propriedades que irão ditar o formato da interface gráfica. Uma vez que o arquivo XML tenha sido carregado na aplicação as características ditadas para formação da tela se mantém até o final da programação.

\*Programática: Temos aqui uma forma mais dinâmica para trabalhar as interfaces gráficas, visto que nesta maneira os elementos são descritos em tempo de execução, permitindo uma mudança na tela baseada em eventos manipulados pelo usuário ou por partes do código.

**- Componentes Gráficos**

Os componentes gráficos na programação android podem ser divididos em dois grupos, os views e os viewsGroups.

\* View: São os componentes da interface gráfica em si, botões, textos, campos de texto. As propriedades desses componentes são relevantes somente para eles mesmo não afetando outros.

\* ViewGroup: São um conjunto de views, também podem conter outros viewgroup. As propriedades deste são relevantes para todos os componentes do grupo.

Toda interface começa a partir de um viewgroup que irá acomodando views ou outras viewgroup, criando-se assim uma hierarquia onde as modificações que são feitas no nível mais alto afetam os níveis mais baixos.

**- Atributos das Views**

Toda view ou gruopview possuem um conjunto de atributos que irão estabelecer sua aparência, seu comportamento e sua identificação como objeto.

Alguns atributos são obrigatórios outros são opcionais. Alguns são atributos comuns a todos e outros são específicos.

Como exemplo de alguns atributos temos:

\* id: permite a view ter uma identificação, um nome, que é usado para manipula-lo dentro da programação das classes Java. Não é obrigatório, mas se uma parte do código pretende acessá-la ela deve ter um id. O formato desta declaração é:

Android:id = “@+id/nomeDoObjeto

\* layout\_width: Atributo obrigatório que define o tamanho horizontal do componente.

\* layout\_height: Também obrigatório a todos os componentes, definindo o tamanho vertical.

Esses atributos podem ter um tamanho exato, determinando um valor numérico em pixel ou de forma dinâmica através de duas constantes:

\* mach\_parent: indica que a view deve ocupar todo o espaço disponível no elemento pai.

\* wrap\_content: indica que a view deve fazer uso somente do espaço suficiente para acomodar seu conteúdo.

Caso queira fazer uso de um tamanho fixo deve-se fazer uso da unidade dip (density-independent pixel), que mantém o tamanho da view mesmo com variação de densidade de pixel das variadas telas.

**- Layouts**

Os layouts são um tipo de viewgruop pois suas propriedades vão influenciar no posicionamento das views que estiverem dentro deles.

Todos os arquivos de layouts são posicionados dentro da pasta res, e todos os objetos desses arquivos são referenciados dentro de uma classe, que é criada pelo android studio, que é chamada de **R**.

O android consegue carregar um objeto que não esteja em uma classe Java que esteja na classe **R**.

A chamada de um layout para a aplicação é feita através do comando setContentView(R.layout.NomeDoLayout).

Os três principais tipos de layout disponível no android são:

\* LinearLayout: organiza os elementos contidos nele de forma linear seja verticalmente ou horizontalmente. Alguns de seus atributos são:

\* android:orientation – define se a orientação será vertical ou horizontal.

\* android:id – identifica o layout.

\* android:layout\_height – define o atributo de altura.

\* android:layout\_width – define o atributo de largura.

\* android:gravity – define a posição dos filhos do layout, podendo ser top, center, bottom, entre outros.

\* android:padding – define um valor fixo em pixel ou dip que será adicionado aos views do layout, também define em qual direção este valor será agregado.

\* android:weight – atributo definido para os elementos do layout que irá definir o peso que terá para ser exibido. Quanto maior o peso mais espaço a view terá. O peso default é 0.

\* RelativeLayout: neste tipo de layout podemos organizar os views em relação ao pai ou a outros views. Para relacionar com o pai usaríamos comandos como:

android:layout\_alignParentBotton = “true”

android:layout\_alignParentLeft = “true”

Para relacionar com outros views usaríamos comandos como:

android:layout\_toRightOf = “id\_da\_view\_de\_referencia”

\* TableLayout: neste tipo de layout deve-se definir as linhas do layout e quais elementos devem se posicionar naquela linha, formando assim as colunas, como em uma tabela.

\* GridLayout: torna a tela uma espécie de grade onde poderemos organizar os views de forma reativa e definir seus espaços.

\* FrameLayout: é um layout onde devemos colocar um só componente e este ocupará toda a sua área, se outros elementos forem colocados eles irão se sobrepor. É muito utilizado para se colocar outros tipos de layouts dentro dele.

**Aula 04 - Activities I**

O código para a implementação de uma activity simple é:

package br.com.teste; //nomeação do pacote da aplicação

import android.app.Activity;

import android.os.Bundle;

public class NomeActivity extends Activity{ // toda activity deve extender da android.app.Activity

@Override

public void onCreate(Bundle savedInstanceState){ // onCreate único callback implementado

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

}

}

Após codificar a Activity esta deve ser declarada no AndroidManifest.xml.

<manifest....>

<aplication...>

<activity android:name = “.NomeActivity” />

</aplication>

</manifest>

Sendo declarada no manifest a activity está pronta para ser usada na aplicação ou por outras aplicações, se assim for configurada.

**- Estados de um Activity**

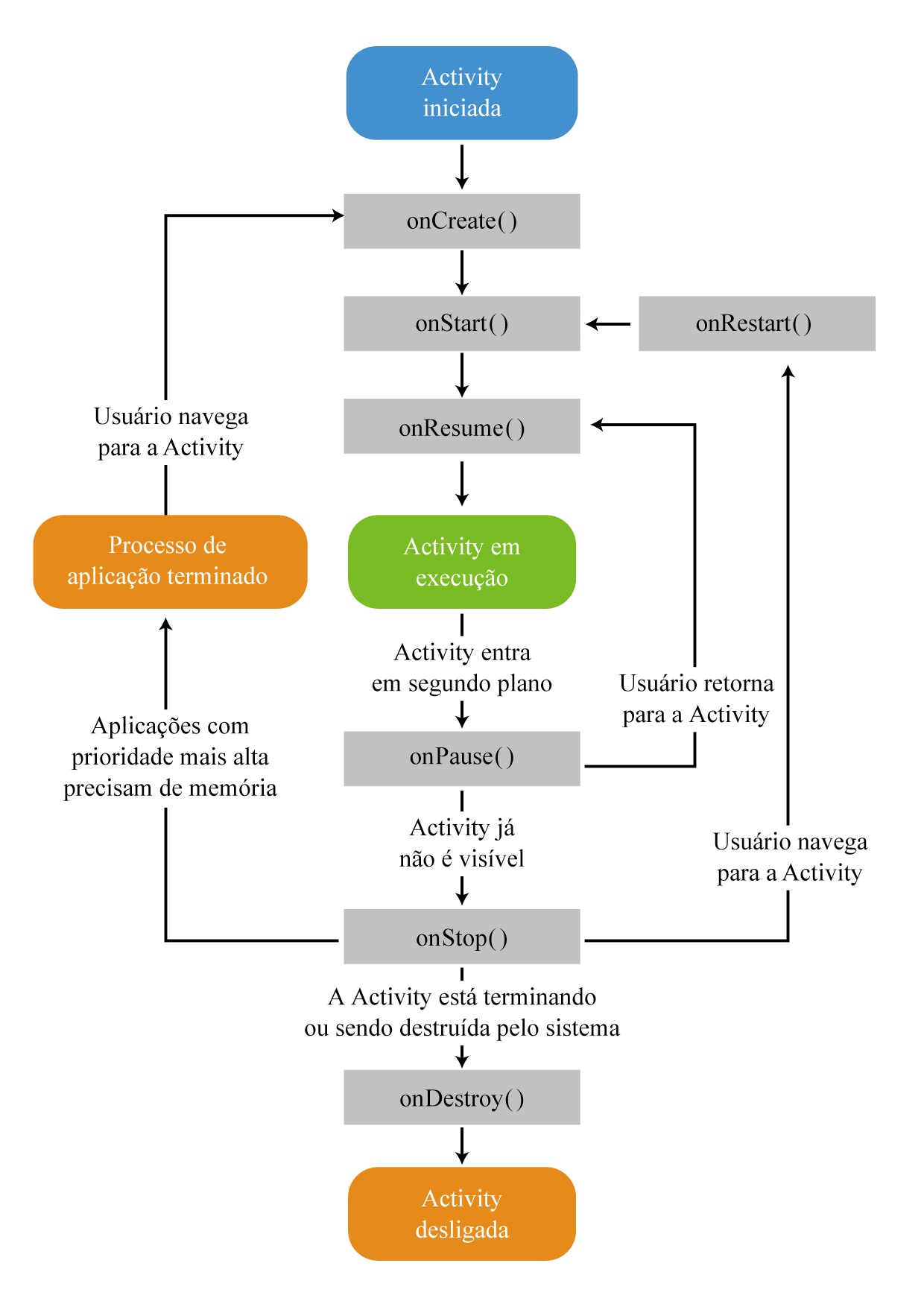
Os estados em que uma activity pode se encontrar são:

\* **Ativa**: Está inicializada e em primeiro plano.

\* **Pausada**: A activity ainda está visível mas tem uma janela em sua frente. Apesar de estar visível ela não estará sendo executada.

\* **Parada**: Ocorre quando outra activity sobrepõe totalmente ela. Estará válida podendo ainda ser executada se voltar ao primeiro plano.

\* **Morta**: A aplicação não estará mais na memória do dispositivo. Ela terá sido encerrada por falta de memória ou pelo usuário.

 **- Ciclo de Vida de uma Activity**

- Método onCreate

É o primeiro método a ser chamado por uma activity.

É o único método callback que deve obrigatoriamente ser implementado em todas as activitys.

Nele implementaremos tarefas que devem ser executadas apenas uma vez na vida da aplicação.

Ela será executada quando: for criada, cada seja finalizada prematuramente ou por interrupção causada por mudança de configuração do aparelho.

Este método faz uso do parâmetro savedInstanceState que é responsável por salvar qualquer dado que seja importante para a activity antes de salvá-los. Então se houver uma interrupção ao se recriar a activity este parâmetro será consultado para ver se há dados que existiam antes e devem ser recarregados.

- Método onDestroy

É chamado quando uma aplicação está sendo encerrada.

Deve ser usado para liberação de recursos como, encerramento de conexões, fechamento de arquivos.

- Método onStart e onRestart

Quando a aplicação é iniciada a primeira vez ela passa do método onCreate para o método onStart.

Este método ocorre quando a activity está se tornando visível, e é a chance de se inicializar algum componente não inicializado no onCreate.

Quando a aplicação passa do estado Parado voltando a estar visível novamente ela passa pelo método onRestart e novamente pelo método OnStart.

- Método noStop

É chamada quando a activity vai deixar de ser visível.

Não se deve usar este método para salvamento de coisas importantes, pois o sistema poderá encerrar a aplicação sem passar por ela.

Observe em colocar apenas processos leves nesse método, pois a activity pode voltar a se ativada rapidamente.

- Método onResume

Este é o último método a ser chamado antes a activity estar visível.

Ele ocorre tanto quando a aplicação está sendo executada pela primeira vez quanto está sendo restaurada.

- Método onPause

É o oposto ao método onResume.

Nele deve ser desfeito tudo que foi feito no método onResume.

Aqui deve-se ter o cuidado de liberar qualquer recurso de uso exclusivo, como a câmera.

**Aula 05 - Interfaces Gráficas II**

Criando uma activity para login:

- **Layout**: Será usado o relativelayout.

\* **Mudança de cor**: android:background = “#007” -> aqui foi usado o formato RGB onde ainda temos os ARGB, #RRGGBB ou #AARRGGBB, onde A é alpha e representa o nível de transparência da cor, sendo que quanto maior o valor de A mais sólida é a cor. Os valores são expressos em hexadecimal( 0 – A).

- **TextView**: terá a seguinte configuração

<TextView

android:id="@+id/titulo"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:layout\_marginTop="15dip"

android:gravity="center"

android:text="TELA DE LOGIN"

android:textColor="#131313"

android:textSize="30dip"

android:textStyle="bold" />

- **EditText**: terá a seguinte configuração

<EditText

android:id="@+id/login\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentLeft="true"

android:layout\_alignParentRight="true"

android:layout\_below="@+id/welcome\_msg"

android:layout\_marginTop="50dp"

android:hint="Login"

android:maxLines="1" >

</EditText>

**- Para configurar o espaçamento interno da tela:**

Na configuração do layout utilizado usar a tag padding:

android:padding=”16dp”

Isso determina o espaçamento interno das bordas da tela.

**- Passando um aviso Toast na tela.**

Usa-se o comando: Tost.makeText que possui os seguintes parâmetros:

1 – Context: determina o contexto onde será exibido. Se for na activity usa-se this.

2 – Mensagem: Pode ser um objeto ou um texto entre aspas.

3 – Tempo de duração da exibição: Length\_Long(1) ou Length\_Short(0)

Para ser exibido deve-se colocar após o parênteses do parâmetro a comando .show.

Toast.makeText(this,”Mensagem”,Toast.LENGTH\_LONG).show;

**- Configurando uma view para ficar centralizada**

Na configuração da view usar a tag gravity como center.

android:gravity=”center”

**- Chamando uma activity**

1 – Cria-se um objeto Intent com os seguintes parâmetros no construtor:

1 – Contexto: Se é o da activity atual usa-se this.

2 – Destino: Coloca-se o nome da activity que será chamada.

2 – Chama-se o método start.Activity onde será chamada a Intent (intenção) criada.

Intent intent = new Intent(this, nomeActivity.class);

start.Activity(intent);

- T

**Aula 06 - Activities II**

**Intent**

É um componente responsável pela comunicação entre componentes, carregando dentro de si as informações relacionadas as ações que devem ser tomadas.

Os Intents ao serem lançados levam dentro de si alguns objetos que indicam a intenção deles e dados que devem ser enviados. Esses campos são:

\* Action: é responsável por dizer qual o componente que deverá ser inicializado. Isso pode ser feito fazendo uso de uma string que contenha o caminho até o componente que será inicializado ou através de uma ação padrão do Android. Exemplos de ações padrões do Android.

- ACTION\_VIEW: inicia uma nova activity.

- ACTION\_CALL: fazer uma chamada telefônica.

- ACTION\_SEND: enviar informações.

- ACTION\_WEB-SEARCH: realizar uma pesquisa web.

\* Data: passa os dados necessário para se realizar a ação desejada. Exemplo:

intent intentLigar = new intent(intent, ACTION\_DIAL);

intentLigar.setData(Uri.parse(“tel:3838833838”));

startAcitivity(intentLigar);

\* Category: fornecem informações extras ao componente que deve responder ao intent.

\* Component: passa o nome do componente que a intent é destinada.

- **Tipos de Intents**.

As intents estão em duas categorias:

\* Implícitas: são aquelas que não possuem o campo Component configurado, fazendo com que seja respondida por qualquer componente que se encaixe nos campos configurados.

Caso o android encontre mais de uma aplicação habilitada para responder ao intent é solicitado ao usuário que escolha qual aplicação deverá recebe-la.

Não se deve iniciar um serviço com uma intent implícita, uma vez que estas ocorrem em background e não há intervenção do usuário.

\* Explícitas: é definido para qual componente a intent se destina através do campo Component. Usado, normalmente, para um componente que faça parte da aplicação.

- Intent Filters

É a forma de buscar quais componentes são hábeis para responder a um intent, sendo colocados filtros para fazer essa procura, sendo assim, as intents que forem explícitas, ou seja, tem seu campo Component definido tem esses filtros ignorados.

Os filtros são declarados no junto a declaração do componente no AndroidManifest. Uma intent pode declarar um ou mais filtros, sendo que um componente para ser habilitado para recebe-la dever passar em todos os filtros.

Os Filters possuem os seguintes campos: Action, data e category.

Se o android não consegue encontrar uma aplicação que consiga atender a chamada da intent é dada uma mensagem de erro encerrando a aplicação. Para evitar isso pode-se fazer uma verificação prévia da existência de aplicações que possam atende-la. Isso é feito com o seguinte código.

List listaDeProgramas = getPackagerManange().queryIntentActiveties(NomeDaIntent,PackagerManange.MATH\_DEFAULT\_ONLY);

If(!listaDeProgramas.IsEmpty(){

startyActivity(NomeDaIntent);

}else{

//ação de aviso ao usuário e retorno uma activity

}

**Aula 07 - Interfaces Gráficas III**

**ListView**

É um componente responsável por prover listas de dados.

Carrega dados através de um adapter. Pode ser implementado desta forma:

ListView lv = (ListView) findViewById(R.id.listView1);

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.list\_item, PAISES);

lv.setAdapter(adapter);

lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> pai, View view, int posicao, long id) {

Toast.makeText(getApplicationContext(), ((TextView) view).getText(),

Toast.LENGTH\_SHORT).show();

}

});

**Options Menu**

É um componente responsável por dar opções para o usuário na tela.

**Context Menus**

É um menu de contexto que quando acionado passa a flutuar sobre a activity.

**Aula 08 - Localização e Mapas**

O Android tem duas maneiras de fazer a geolocalização do usuário: GPS ou Network Location Provider.

No GPS é feita a localização por satélite e no Network Location Provider fazendo uso das torres de telefonia móvel e informações sobre WIFI.

A implementação do código para uso deste recurso deve ser feito no método onCreator:

LocationManager locationManager = (LocationManager)

this.getSystemService(Context.LOCATION\_SERVICE);

LocationListener locationListener = new LocationListener() {

@Override

public void onStatusChanged(String provider, int status, Bundle extras) {

//Código para responder a alterações em um provedor de localização

}

@Override

public void onProviderEnabled(String provider) {

//Código para responder à habilitação de um provedor de localização

}

@Override

public void onProviderDisabled(String provider) {

//Código para responder à desabilitação de um provedor de localização

}

@Override

public void onLocationChanged(Location location) {

//Código para utilizar a nova localização.

}

};

locationManager.requestLocationUpdates(LocationManager.GPS\_PROVIDER, 0, 0,

locationListener);

locationManager.requestLocationUpdates(LocationManager.NETWORK\_PROVIDER, 0, 0,

locationListener);

Ao se trabalhar com localização devemos observar as seguintes estratégias de implementação;

1º - Assim que a aplicação for iniciada dever se inicializar os serviços de localização.

2º - Manter uma melhor estimativa do local possível, definindo quando ficar com uma atualização menos precisa ou manter uma estimativa mais antiga, porém mais precisa.

3º - Após obter a localização mais precisa e em tempo adequado o serviço de localização deve ser interrompido, para evitar um consumo excessivo de bateria. Essa última localização deve ser mantida para uma referência na necessidade de se reativar o serviço de localização.

Para se obter a última posição de localização usa-se o método getLastKnownLocation.

Para parar o serviço de localização é preciso tirar o registro do Listerner, utilizando o método do LocationManeger removeUpdates, passado ao listerner.

**Aula 09 - Estilos, Temas e Compatibilidade**

**- Estilos**

Estilos são formatações que são definidas em arquivos XML, que podem ser utilizados em vários layouts.

Os estilos são definidos para uma view com a tag style=”@style/nomeDoEstilo”.

Para se definir estilos deve-se criar um arquivo XML dentro da pasta res/values. Dentro deste arquivo a tag principal deve ser a tag <resources>

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

<resources>

<style name="CodeFont">

<item name="android:layout\_width">fill\_parent</item>

<item name="android:layout\_height">wrap\_content</item>

<item name="android:textColor">#00FF00</item>

<item name="android:typeface">monospace</item>

</style>

</resources>

Pode-se criar estilo novos ou estilos herdados e personalizados alguns itens:

<style name="GreenText" parent="@android:style/TextAppearance">

<item name="android:textColor">#00FF00</item>

</style>

Para herdar um estilo da própria aplicação não se usa o comando parent e sim um ponto após o nome do estilo que será o pai seguido do nome da herdeira.

<style name="GreenText.Big">

<item name="android:textSize">30sp</item>

</style>

**- Temas**

Temas são estilos que podem ser aplicados a uma activity ou a aplicação.

O uso de tema deve ser definido no arquivo AndroidManifest da aplicação.

<activity android:theme="@style/Tema">

<application android:theme="@style/Tema">

- Compatibilidade

Para atender as diferentes densidades e tamanhos de tela deve-se observar a medida utilizada nas fontes e tamanhos das views.

Para fontes deve-se utilizar a medida em sp (scale indepedent pixel).

Para as views deve-se utilizar o medida em dp (density independent pixel).

Para fazer a declaração de telas suportadas pela aplicação, que deve ser feita no AndroidManifest, faz uso da tag supportss-screen.

**Aula 10 - Resources, Arquivos e Sharedpreferences**

**- Resouces**

Os arquivos utilizados na nossa aplicação são chamados resources.

Os principais modificadores de recursos relacionados a tela são (onde N = valor mínimo especificado):

\* Largura disponível: w<N>dp

\* Altura disponível: h<N>dp

\* Tamanho: small, normal, large, xlarge.

\* AspectRatio: aspecto da tela. Long ou notlong.

\* Orientação: land (paisagem) o port (retrato).

\* Densidade: ldpi, mdpi, hdpi, xhdpi, nodpi e tvdpi.

**- Arquivos**

Para trabalharmos com arquivos devemos fazer uso de um objeto FileOutputStream para escrever os arquivos em Byte.

String nomeArquivo = "arquivo\_hello";

String conteudo = "hello world!";

FileOutputStream fos;

try {

fos = openFileOutput(nomeArquivo, Context.MODE\_PRIVATE);

fos.write(conteudo.getBytes());

fos.close();

} catch (FileNotFoundException e) {

//...

} catch (IOException e) {

//...

}

**Para escrita em String**.

try {

String nomeArquivo = "arquivo\_hello";

String conteudo = "hello world!";

File file = new File(getFilesDir() + "/" + nomeArquivo+ ".teste");

BufferedWriter bw = new BufferedWriter(new FileWriter(file));

bw.write(conteudo);

bw.flush();

bw.close();

} catch (IOException ex) {

//...

}

**Para leitura de arquivos**

BufferedReader input = null;

try {

input = new BufferedReader(new InputStreamReader(openFileInput(nomeArquivo + "\_string.teste")));

String line;

StringBuffer buffer = new StringBuffer();

while ((line = input.readLine()) != null) {

buffer.append(line);

}

Log.d("Leitura de Arquivo", buffer.toString());

} catch (Exception e) {

e.printStackTrace();

} finally {

if (input != null) {

try {

input.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**- Verificação de disponibilidade e acesso do armazenamento externo:**

boolean mExternalStorageAvailable = false;

boolean mExternalStorageWriteable = false;

String state = Environment.getExternalStorageState();

if (Environment.MEDIA\_MOUNTED.equals(state)) {

mExternalStorageAvailable = mExternalStorageWriteable = true;

} else if (Environment.MEDIA\_MOUNTED\_READ\_ONLY.equals(state)) {

mExternalStorageAvailable = true;

mExternalStorageWriteable = false;

}

**- Adicionando arquivos na estrutura de armazenamento externa**

if(mExternalStorageAvailable && mExternalStorageWriteable) {

try {

String filename = nomeArquivo + "\_externo";

File file = new File(getExternalFilesDir(null) + "/" + filename + ".teste");

BufferedWriter bw = new BufferedWriter(new FileWriter(file));

bw.write(conteudo);

bw.flush();

bw.close();

} catch (IOException ex) {

//..

}

}

**- SharedPreferences**

Em uma aplicação Android, é comum salvar as preferências de um usuário através da SharedPreferences.

**Ler um arquivo da pasta res/raw**

imputStream is = getResources().openRawResource(R.raw.meuarquivo);

b = (byte) is.read();

is.close();

**Ler um arquivo XML**

XmlPullParser xpp = getResources().getXml(R.xml.meuXml);

**Aula 11 - SQLite**

- É um pacote de software de código aberto que fornece uma estrutura de gerenciamento de banco de dados.

- O processamento do SQLite ocorre diretamente em arquivos, isso retira a necessidade de haver um servidor.

- Não é necessária a preparação de conexão pois não faz uso de um servidor.

- Acessando o Banco de Dados SQLite:

**1º - Implementação da classe representativa do modelo de uma tabela de comentários**

package PACOTE.DE.SUA.APLICACAO;

public class Comment {

private long id;

private String comment;

public Comment(long id, String comment) {

this.id = id;

this.comment = comment;

}

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getComment() {

return comment;

}

public void setComment(String comment) {

this.comment = comment;

}

@Override

public String toString() {

return comment;

}

}

**2º - Criar uma classe filha SQLiteOpenHelper, que é responsável por criar e gerenciar versão de dados:**

package PACOTE.DE.SUA.APLICACAO;

public class MySQLiteHelper extends SQLiteOpenHelper {

public static final String TABLE\_COMMENTS = "comments";

public static final String COLUMN\_ID = "\_id";

public static final String COLUMN\_COMMENT = "comment";

private static final String DATABASE\_NAME = "commments.db";

private static final int DATABASE\_VERSION = 1;

private static final String DATABASE\_CREATE = "create table "

+ TABLE\_COMMENTS + "( " + COLUMN\_ID

+ " integer primary key autoincrement, " + COLUMN\_COMMENT

+ " text not null);";

public MySQLiteHelper(Context context) {

super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

}

@Override

public void onCreate(SQLiteDatabase database) {

database.execSQL(DATABASE\_CREATE);

}

@Override

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

Log.w(MySQLiteHelper.class.getName(),

"Upgrading database from version " + oldVersion + " to "

+ newVersion + ", which will destroy all old data");

db.execSQL("DROP TABLE IF EXISTS " + TABLE\_COMMENTS);

onCreate(db);

}

}

**3º - Definindo a estrutura de conexão:**

package PACOTE.DE.SUA.APLICACAO;

public class CommentsDao {

private MySQLiteHelper dbHelper;

private SQLiteDatabase database;

private String[] allColumns = { MySQLiteHelper.COLUMN\_ID, MySQLiteHelper.COLUMN\_COMMENT };

public CommentsDao(Context context) {

dbHelper = new MySQLiteHelper(context);

}

public void open() throws SQLException {

database = dbHelper.getWritableDatabase();

}

public void close() {

database.close();

dbHelper.close();

}

public Comment createComment(String comment) {

ContentValues values = new ContentValues();

values.put(MySQLiteHelper.COLUMN\_COMMENT, comment);

long insertId = database.insert(MySQLiteHelper.TABLE\_COMMENTS, null, values);

Comment newComment = new Comment(insertId,comment);

return newComment;

}

public void deleteComment(Comment comment) {

long id = comment.getId();

database.delete(MySQLiteHelper.TABLE\_COMMENTS, MySQLiteHelper.COLUMN\_ID + " = " + id, null);

}

public List<Comment> getAllComments() {

List<Comment> comments = new ArrayList<Comment>();

Cursor cursor = database.query(MySQLiteHelper.TABLE\_COMMENTS, allColumns, null, null, null, null, null);

cursor.moveToFirst();

while (!cursor.isAfterLast()) {

Comment comment = cursorToComment(cursor);

comments.add(comment);

cursor.moveToNext();

}

cursor.close();

return comments;

}

private Comment cursorToComment(Cursor cursor) {

Comment comment = new Comment(cursor.getLong(0), cursor.getString(1));;

return comment;

}

}

**Aula 12 - Multimídia**

**-** Para a reprodução de áudios e vídeos o android disponibiliza a API MediaPlayer.

- Para a aplicação acessar a internet é necessário que se adicione a permissão no AndroidManifest.

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

- Permissão para que não haja o escurecimento da tela durante a execução de um vídeo.

<uses-permission android:name="android.permission.WAKE\_LOCK"/>

- Os arquivos de mídia devem estar dentro da pasta RAW.

- Código para executar uma mídia da pasta raw:

Context mContext = getApplicationContext();

MediaPlayer mediaPlayer = MediaPlayer.create(mContext, R.raw.sa);

mediaPlayer.start();

- Código para executar um arquivo de áudio de um local específico:

Uri myUri = Uri.parse("Alguma URI");

MediaPlayer mediaPlayer = new MediaPlayer();

mediaPlayer.setAudioStreamType(AudioManager.STREAM\_MUSIC);

try {

      mediaPlayer.setDataSource(getApplicationContext(), myUri);

      mediaPlayer.prepare();

} catch (IllegalArgumentException e) {

} catch (SecurityException e) {

} catch (IllegalStateException e) {

} catch (IOException e) {

}

mediaPlayer.start();

- Código para executar uma mídia da internet:

String url = "http://site.para.stream";

MediaPlayer mediaPlayer = new MediaPlayer();

mediaPlayer.setAudioStreamType(AudioManager.STREAM\_MUSIC);

try {

mediaPlayer.setDataSource(url);

mediaPlayer.prepare();

} catch (IllegalArgumentException e) {

} catch (IllegalStateException e) {

} catch (IOException e) {

}

mediaPlayer.start();

**- Utilização da Câmera**

- Para se fazer uso da câmera na aplicação é necessário autorização no AndroidManifest

<uses-feature android:name = "android.hardware.camera" />

Isso torna sua aplicação no GooglePlay a aparelhos que tenham câmera. Para evitar isso deve-se explicitar que isso não é um requisito para uso da aplicação.

<uses-feature android:name="android.hardware.camera" android:required="false" />

- Para que seja possível gravar áudio, falsar imagens é necessário solicitar permissões.

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

<uses-permission android:name="android.permission.RECORD\_AUDIO" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

- Capturando uma imagem da câmera e salvando em um local específico.

private static final int CAPTURAR\_IMAGEM = 100;

private Uri fileUri;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

Intent intent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

File mediaStorageDir = new File(

Environment.getExternalStoragePublicDirectory(

Environment.DIRECTORY\_PICTURES), "NomeDaPasta");

if (!mediaStorageDir.exists()){

mediaStorageDir.mkdirs();

}

String timeStamp =

new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date());

File file = new File(mediaStorageDir.getPath() + File.separator +

"IMG\_"+ timeStamp + ".jpg");

fileUri = Uri.fromFile(file);

intent.putExtra(MediaStore.EXTRA\_OUTPUT, fileUri);

startActivityForResult(intent, CAPTURAR\_IMAGEM); }

- Capturando um vídeo da câmera e salvando em um local específico

private static final int CAPTURAR\_VIDEO = 200;

private Uri fileUri;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

Intent intent = new Intent(MediaStore.ACTION\_VIDEO\_CAPTURE);

File mediaStorageDir = new File(

Environment.getExternalStoragePublicDirectory(

Environment.DIRECTORY\_PICTURES), "NomeDaPasta");

if (!mediaStorageDir.exists()){

mediaStorageDir.mkdirs();

}

String timeStamp =

new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date());

File file = new File(mediaStorageDir.getPath() + File.separator +

"VID\_"+ timeStamp + ".mp4");

fileUri = Uri.fromFile(file);

intent.putExtra(MediaStore.EXTRA\_OUTPUT, fileUri);

intent.putExtra(MediaStore.EXTRA\_VIDEO\_QUALITY, 0);

intent.putExtra(MediaStore.EXTRA\_DURATION\_LIMIT, 30);

intent.putExtra(MediaStore.EXTRA\_SIZE\_LIMIT, (1024 \* 1024));

startActivityForResult(intent, CAPTURAR\_VIDEO);

}

- O resultado da captura da câmera é retornado para o método onActivityResult.

- Estrutura do onActivityResult para o Intent da câmera:

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

if (requestCode == CAPTURAR\_IMAGEM) {

if (resultCode == RESULT\_OK) {

// Imagem capturada com sucesso e salva no URI especificado na chamada

Toast.makeText(this, "Imagem salva em:\n" +

data.getData(), Toast.LENGTH\_LONG).show();

} else if (resultCode == RESULT\_CANCELED) {

// Captura cancelada pelo usuário

} else {

// Captura falha

}

}

if (requestCode == CAPTURAR\_VIDEO) {

if (resultCode == RESULT\_OK) {

// Vídeo capturado com sucesso e salvo no URI especificado na chamada

Toast.makeText(this, "Vídeo salvo em:\n" +

data.getData(), Toast.LENGTH\_LONG).show();

} else if (resultCode == RESULT\_CANCELED) {

// Captura cancelada pelo usuário

} else {

// Captura falha

}

}

}

- Código para saber o número de câmeras de aparelho e qual é o id delas

int numeroDeCameras = Camera.getNumberOfCameras();

for (int i =0; i < numeroDeCameras; i++){

CameraInfo info = new CameraInfo();

Camera.getCameraInfo(i, info);

if ( info.facing == CameraInfo.CAMERA\_FACING\_FRONT){

// frontal

}else if (info.facing == CameraInfo.CAMER\_FACING\_BACK){

/ traseira

} }

**Aula 13 - Services, Content Providers e Broadcast Receivers**

**- Service**

\* Funcionam em backgroud, sem interagir com o usuário.

\* Podem continuar funcionando mesmo quando a aplicação terminar.

\* Podem ser utilizados por aplicações que não o tenham iniciado.

Um exemplo de aplicação de serviço seria fazer um download em uma thread que não a principal a activity, isso deixaria a aplicação livre enquanto ao download é feito sendo avisado quando terminar.

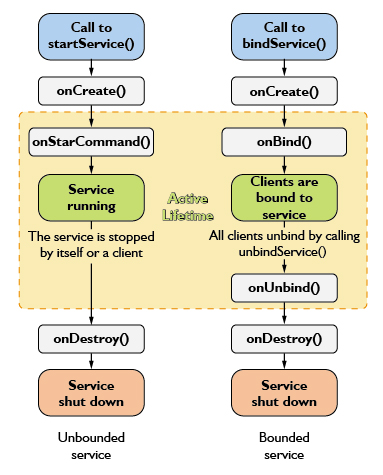
- Passos para iniciar um service.

1 – Estender a classe que irá iniciar o serviço à classe service.

2 – Declarar no AndroidManifest o serviço que será utilizado.

<service android:name=".ExampleService" />

3 – Para que o serviço só seja utilizado para a aplicação deve-se fazer uso do intent filter android:exported com valor false.



-  Declaração de um Service Started que estende de IntentService.

public class ServiceStarted extends IntentService {

public ServiceStarted() {

super("ServiceStarted");

}

@Override

protected void onHandleIntent(Intent intent) {

long endTime = System.currentTimeMillis() + 5\*1000;

while (System.currentTimeMillis() < endTime) {

synchronized (this) {

try {

wait(endTime - System.currentTimeMillis());

} catch (Exception e) {

}

}

}

}

}

**- Content Providers**

É responsável por fazer a leitura de dados que já estejam armazenados no telefone. Por apresentar dados em conjunto de tabelas é bem similar a um banco de dados.

**- Broadcast Receivers**

São responsáveis por informar a incidência de um evento do sistema ou aplicação.

Esses evento são enviados na forma de intent através do método sendBroadcast().

**Aula 14 - Internet**

**- Webview**

É o componente que se utiliza para fazer visualizações de páginas web dentro da aplicação.

Para se utilizar o webview deve-se declarar no AndroidManifest :

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

Essa view deve ser usada para visualização de páginas simples e não para navegação.

Para abrir um browser externo para navegação usa-se o seguinte código:

Uri uri = Uri.parse("http://10.0.2.2:8080");

Intent intent = new Intent(Intent.ACTION\_VIEW, uri);

startActivity(intent);

A Webview também permite a execução de código HTML direto ou a execução de um arquivo que contenha o código HTML.

Para habilitar o suporte a javaScript no webview usa-se o seguinte código:

Webview myWebview = (Webview) findViewById(R.id.webview);

WebSettings webSettings = myWebView.getSettings();

webSettings.setJavaScriptEnabled(true);

- Conexões HTTP

É disponível três formas de fazer a conexão HTTP:

- Api padrão java : java.net.HttpUrlConnection.

- O framework commons HttpClient da Apache: org.apache.http.client.HttpClient;

- Através da classe android : android.net.http.AndroidHttpClient.

**Aula 15 - Distribuição de Aplicações**

Uma aplicação pode ser distribuída por e-mail, site ou pelo Google Play.

**- Preparando a Aplicação para a Distribuição**

1° - Aquisição de materiais:

\* Ícone da aplicação dentro dos requisitos exigido pelo Android.

\* Adquirir a EULA (End User Lincense Agreement), que é distribuída junto com a aplicação para que o usuário final concorde com os termos dela.

\* Para uso da loja do Google Play é necessário ter: um texto promocional explicativo em inglês e língua nativa, vídeos de demonstração e screenshots, no mínimo duas capturas de telas feitas no app.

2º - Nome do pacote:

\* O nome do pacote deve ser único, por isso recomenda-se que use o nome da url ao contrário.

\* O nome do pacote uma vez lançado não pode ser mudado. Caso seja mudado será tratado como uma nova aplicação.

3º - Retirada de Debugs e Logs:

\* Uma vez a aplicação terminada não será mais necessário fazer o debug, por isso no AndroidManifest deve ser configurado o parâmetro android:debuggable para false.

\* Todas as linhas de logs que tiverem sido colocadas no código devem ser retiradas.

4º - Atualização de Diretórios:

\* Retire das pastas da aplicação arquivos que não serão utilizados na aplicação, para evitar que o tamanho da aplicação seja maior que o realmente necessário.

\* Verifique que os arquivos que ficaram são os mais atuais.

5º - Revise o Manifest e o Gradle:

\* Verifique que todas as permissões que a aplicação precisa estão declaradas.

\* Verifique o nome da aplicação no parâmetro label do Manifest.

\* Definir a versão da aplicação.

\* Configure o android:versionCode e o android:versionName. Utilize o padrão como x.y.z, onde x = versão do aplicativo, y = atualizações de aprimoramento da versão e z = atualizações corrigindo erros da versão atual.

6º - Compatibilidade:

\* Teste a compatibilidade com as diferentes versões e tamanho e densidade de tela das diversidades de dispositivos, nas ferramentas do android.

7º - Servidores e Serviços:

\* Faça uma verificação nos endereços de servidores e serviços para ter certeza que todos estão apontando para o local certo e não algum local de teste.

\* Teste o lado do servidor da aplicação se tiver.

8º - Sistema de Licenças:

\* Se a aplicação for paga faça uso da API do google que verifica o status da versão instalada com a versão da loja, se a aplicação está em versão atual, se o usuário tem autorização de uso e outras.

**- Compilando a Aplicação**

Processo de deve-se passar antes de gerar o .apk que será instalado nos aparelhos.

1º - Compilar e assinar a aplicação. Isso é possível pela opção Build -> Generate Signed APK.

Ao criar sua Keystore certifique-se de mantê-la em local seguro.

**- Distribuição da Aplicação**

Publicação no Google Play. As vantagens de se publicar no loja da Google:

\* Se encarrega das versões das atualizações.

\* Oferece proteção contra a pirataria.

\* Faz divulgação da aplicação.

\* Cria uma página personalizada da aplicação.

\* Gerencia o número de instalações.

\* Gerencia dos erros que são gerados pela aplicação.

\* Cobra uma taxa de U$25 e 30% do valor de cada venda.

Acessar o endereço <https://play.google.com/apps/publish/> para enviar a aplicação.

Publicar fora do Google Play.

Para receber uma aplicação que não seja pelo Google Play basta configurar o aparelho receber aplicações de fontes desconhecidas.

Após isso basta receber a .apk por e-mail, bluetooth ou web e instalá-lo. Desta forma a aplicação pode ser livremente distribuída de aparelho para aparelho não tendo assim qualquer controle do desenvolvedor sobre o uso de sua aplicação.