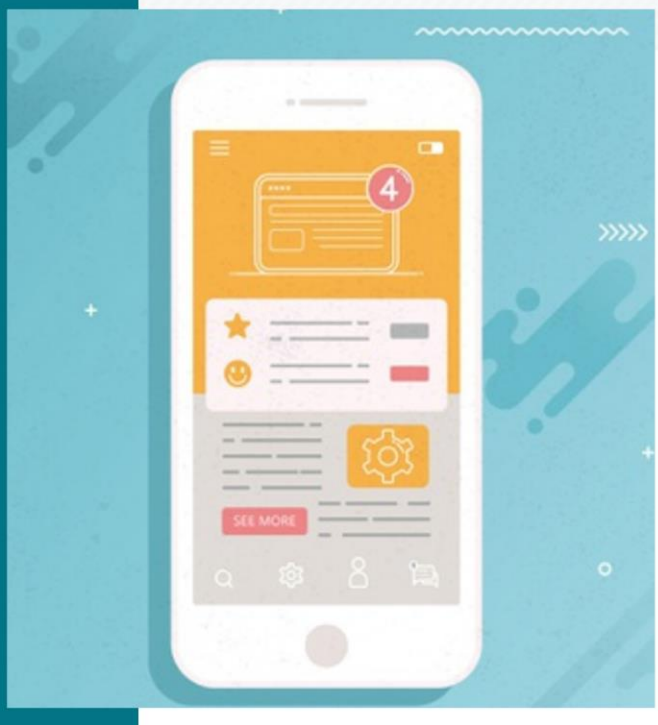
AGENDA 2

LAYOUT E ESTRUTURA DO PROJETO NO ANDROID STUDIO



Para que você consiga digitar um texto e enviar para seu amigo por meio de um App de troca de mensagens, vários **padrões** e **programações** foram estabelecidas para que o resultado seja alcançado por você, usuário, e seu amigo receba o texto digitado!

O aplicativo é constituído, basicamente, de uma tela e de uma classe responsável pelas ações realizadas por ela.

Nesta agenda, vamos aprender com nosso amigo de estudos, Marcelo, como estabelecer e criar o layout do aplicativo, desenvolvendo as telas e seus componentes. Vamos trabalhar com cores e definir os padrões de desenvolvimento através do "**ConstraintLayout**".

MERGULHANDO NO TEMA...

ConstraintLayout

O layout de uma aplicação é algo muito importante, é ele o responsável pela não satisfação do usuário com um determinado aplicativo. Não adianta o aplicativo conter as melhores codificações e desempenho, se o usuário não aprovar o seu layout e usabilidade.

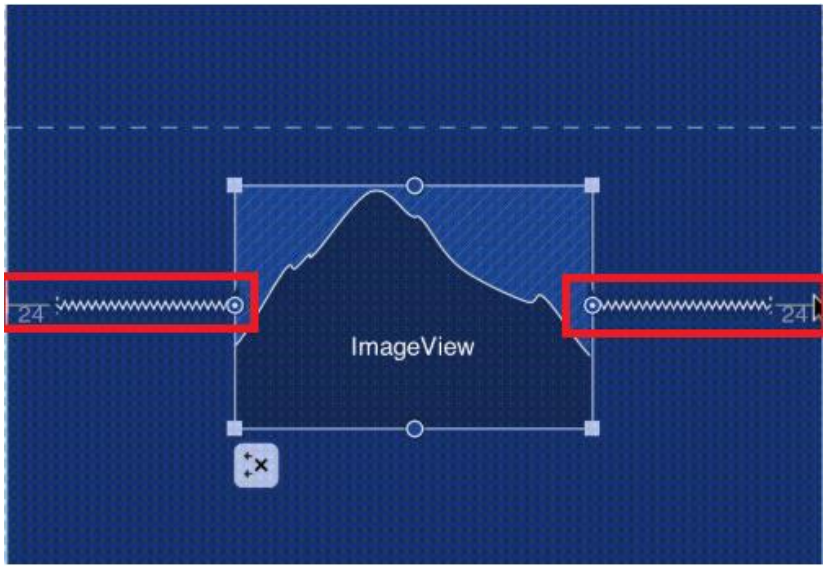
O **Constraintlayout** foi desenvolvido para melhorar a experiência do desenvolvedor durante o processo de criação de uma Activity (tela da aplicação). Ele necessita que sua criação seja fielmente apresentada nas telas dos mais diferentes dispositivos, assim como foi desenvolvida no Android Studio.E isso não estava ocorrendo com padrões de desenvolvimento de interface anteriores, por exemplo, com o **LinearLayout**.

O Android Studio sofreu algumas remodelações em seu **"Editor de Layout"** para que o desenvolvedor utilize o padrão *ConstraintLayout*, e não pense que isso tornou o processo de construção mais difícil e demorado!

Aconteceu exatamente o inverso, o processo de desenvolvimento de interface ficou mais simples e menos complexo.

Para trabalhar com o *ConstraintLayout* é necessário compreender como ele funciona. Constraint significa “limitação”, desta forma, vamos imaginar que em um determinado componente na tela, este padrão oferece uma limitação para que esse componente seja “amarrado” em uma determinada posição da *Activity* (tela).

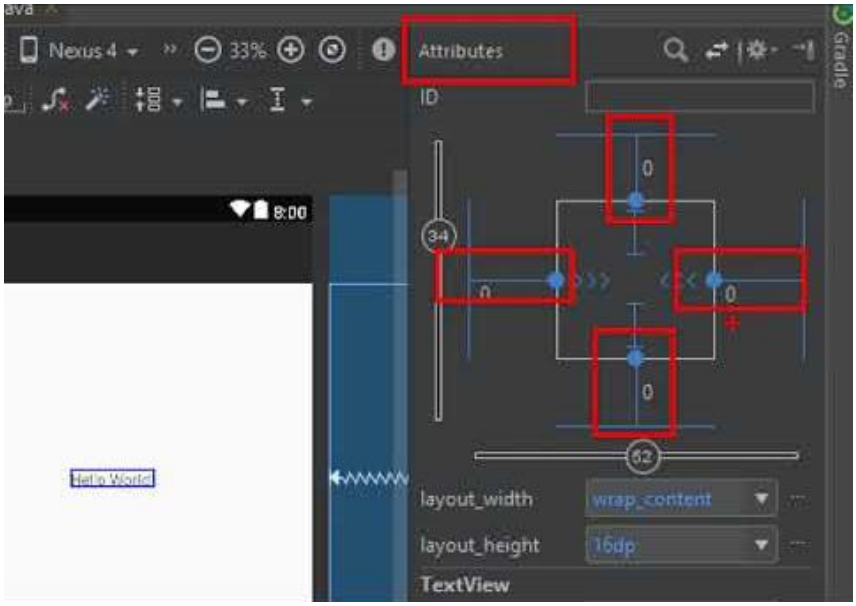
A figura a seguir, mostra um componente **ImageView** em uma **Activity**. O componente recebeu a **Constraint** esquerda e direita, gerando a sua amarração na tela, ou seja, o componente é fielmente exibido neste ponto, independentemente do tamanho e posição da tela do celular durante a execução do aplicativo pelo usuário.



Quando um componente é arrastado para a tela, ele fica na posição escolhida pelo desenvolvedor, só que, mesmo assim, é necessário criar suas “amarrações”. Caso o desenvolvedor não efetue esse processo, todos os componentes são mostrados na posição superior esquerda da tela, ficando impossível sua utilização.

Para saber como criar uma Constraint manual e uma automática realizado pelo Android Studio, assista ao vídeo Tutorial a seguir.

Esse tutorial mostra a construção de um projeto simples chamado de “Aula02”, este projeto contém uma Activity inicial com um botão. Verifique o processo de criação de uma Constraint manual e de uma automática realizado pelo Android Studio.



Pacotes do Projeto

Após a criação do projeto "**Aula02**" o Android Studio lista em sua guia "**Project**" todos os arquivos necessários para execução do aplicativo.

É importante observar e compreender a função de cada pacote e arquivo encontrado na pasta "**app**", vamos utilizar esses pacotes para criar no futuro nossas Activitys adicionais do projeto.

Na **pasta "Manifests"** encontramos o arquivo "AndroidManifests.xml" responsável pelas configurações essenciais do aplicativo. É nele que encontramos as permissões necessárias para que o aplicativo funcione em um determinado dispositivo e as Activitys presentes no projeto.

A segunda **pasta "Java"** é a responsável por armazenar as classes Java do aplicativo juntamente com suas respectivas codificações. É importante reforçar que toda Activity é composta de um arquivo XML e de um arquivo Java. O arquivo Java de uma Activity fica disponível na primeira pasta no pacote "Java".

A terceira **pasta "res"** abreviação de "resources" é responsável pelos recursos do projeto, é nesta pasta ou pacote, que encontramos imagens, layouts, ícones, entre outros componentes do aplicativo.

Manifests

O arquivo Manifest é encontrado em todos os aplicativos para dispositivos Android. Ele é fundamental para a instalação do aplicativo e posteriormente na sua execução.

Para a sua perfeita execução o arquivo fica localizado na pasta raiz do sistema e leva o nome de "**AndroidManifests.xml**".

A figura a seguir mostra as principais áreas de atuação do arquivo, fundamentando sua importância. A imagem foi retirada do site oficial do Android Studio e está disponível para consulta de outras informações referentes ao arquivo "AndroidManifests.xml" através do link:

https://developer.android.com/guide/topics/manifest/manifest-intro?hl=pt-br

1. Nomear o pacote Java para o aplicativo. O nome do pacote serve como identificador exclusivo para o aplicativo.
2. Descrever os componentes do aplicativo que abrangem atividades, serviços, receptores de transmissão e provedores de conteúdo que compõem o aplicativo. Ele também nomeia a classe que implementa cada um dos componentes e publica suas capacidades, como as mensagens intent que podem lidar. Essas declarações informam ao sistema Android os componentes e as condições em que eles podem ser inicializados.
3. Determinar os processos que hospedam os componentes do aplicativo.
4. Declarar as permissões que o aplicativo deve ter para acessar partes protegidas de API e interagir com outros aplicativos. Ele também declara as permissões que outros devem ter para interagir com os componentes do aplicativo.
5. Listar a classe de instrumentos que fornecem geração de perfil e outras informações durante a execução do aplicativo. Essas declarações estão presentes no manifesto somente enquanto o aplicativo está em desenvolvimento e são removidas antes da publicação do aplicativo.
6. Declarar o nível mínimo de Android API que o aplicativo exige.
7. Listar as bibliotecas às quais o aplicativo deve se vincular.

O arquivo conta com uma estrutura padrão no formato XML, onde é feita a inserção e alteração das marcações (comandos). No código a seguir, encontramos o arquivo do nosso projeto, e vamos destacar algumas informações importantes, e no decorrer do curso vamos apresentando outras funções.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.guilherme.aula02">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".Principal">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

1. **Package** - identifica o local dos arquivos Java de aplicação
2. **Application -** configuração dos conteúdos de aplicação como ícones, títulos, estilos entre outros recursos.
3. **Activity -** declaração de todas as Activitys utilizadas no aplicativo.

Java

O Android Studio trabalha com alguns padrões de programação, entre eles o JUnit que basicamente é uma ferramenta para auxiliar as equipes de desenvolvimento nos testes com o programa ou aplicativo.

Na pasta Java, encontramos dois pacotes utilizados para os processos de programação conforme figura 4, os pacotes são “androidTest” e “test”.

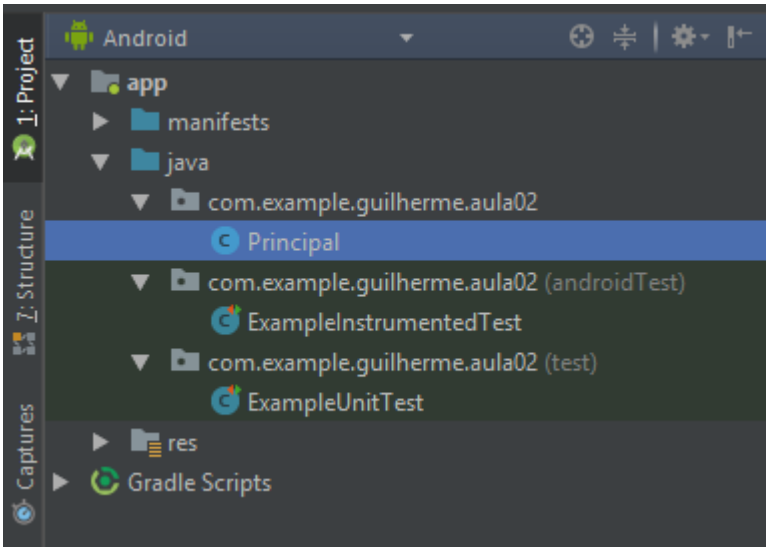


Figura 4 – Pacotes Java.

No decorrer do desenvolvimento do projeto, o pacote que vamos utilizar é o primeiro. É nele que encontramos todas as classes Javas, que levam a codificação das nossas Activitys.

Res

A pasta res é um conjunto de subpastas, onde encontramos os recursos do projeto. Quando incluímos imagens, ícones, entre outros efeitos visuais, são nas dependências da pasta res que esses arquivos são inseridos.

O quadro, retirado do site oficial do Android Studio, apresenta e explica todas os componentes da pasta **“res”**.

Diretório	Tipo de Recurso
animator/	Arquivos XML que definem animações da propriedade.
anim/	Arquivos XML que definem as animações intermediárias. (As animações de propriedade também podem ser salvas neste diretório, mas o diretório **animator/** é o preferencial para animações de propriedade para distinguir os dois tipos.
color/	Arquivos XML que definem uma lista de estado de cores. Consulte Recurso de lista de estado de cores.
drawable/	Os arquivos Bitmap (**png, .9 .png, .jpg, .gif**) ou arquivos XML são compilados nos seguintes subtipos de recurso drawable: • Arquivos Bitmap • Nine-Patch (bitmaps redimensionáveis) • Listas de estado • Formatos • Desenháveis de animação • Outros desenháveis Veja Recursos desenháveis.
mipmap/	Arquivos drawable para diferentes densidades do ícone do inicializador. Para obter mais informações sobre o gerenciamento de ícones do inicializador com pastas **mipmap/**, consulte Visão geral do gerenciamento de projetos.
layout/	Arquivos XML que definem um layout de interface do usuário. Consulte Recurso de layout.
menu/	Arquivos XML que definem os menus do aplicativo, como menu de opções, de menu de contexto ou submenu. Consulte Recurso de menu.
raw/	Arquivos arbitrários para salvar na forma bruta. Para abrir esses recursos com **InputStream** bruto, chame **Resources.openRawResource()** com o código do recurso, que é **R.raw.filename**.

Diretório	Tipo de Recurso
	No entanto, se precisar de acesso aos nomes e à hierarquia dos arquivos originais, considere salvar alguns recursos no diretório **assets/** em vez de **res/raw/**. Os arquivos em **assets/** não recebem um código de recurso, portanto é possível lê-los usando apenas o **AssetManager**.
values/	Arquivos XML que contêm valores simples, como strings, números inteiros e cores. Enquanto os arquivos de recurso XML estiverem em outros subdiretórios **res/**, defina um único recurso com base no arquivo XML, os arquivos no diretório **values/** descrevem vários recursos. Para cada arquivo neste diretório, cada filho do elemento **<resources>** define um único recurso. Por exemplo: um elemento **<string>** cria um recurso **R.string** e um elemento **<color>** cria um recurso **R.color**. • arrays.xml matriz de recurso (matriz digitadas). • colors.xml para valores de cor. • dimens.xml para valores de dimensão. • strings.xml para valores de string. • styles.xml para estilos. Consulte Recursos de string, Recurso de estilo e Mais tipos de recursos.
xml/	Arquivos arbitrários XML que podem ser lidos em tempo de execução chamando **Resources.getXML()**. Vários arquivos de configuração XML devem ser salvos aqui, como uma **configuação buscável**.

Para maiores informações e explicações sobre o conteúdo do pacote "**res**", o site oficial do Android Studio disponibiliza um ótimo artigo, disponível no site:

https://developer.android.com/guide/topics/resources/providing-resources?hl=pt-br

Inserção de uma nova Activity no projeto

Um aplicativo mobile contém uma ou mais Activity's. Geralmente os grandes projetos necessitam de várias telas de interação com o usuário. Esse processo de criação e desenvolvimento das telas de um aplicativo, necessita de alguns conceitos entre eles o processo de transição utilizado pelo Android.

Durante a exibição ou retirada de uma Activity da tela do nosso dispositivo mobile, alguns métodos são executados, o que chamamos de ciclo de vida de uma Activity.

Os métodos de controle do ciclo de vida de uma Activity são:

- **onCreate()** - Chamado quando a atividade é criada pela primeira vez. É onde se deve fazer toda a configuração estática normal, criar exibições, vincular dados a listas etc. Esse método recebe um objeto Bundle (pacote) contendo o estado anterior da atividade, caso esse estado tenha sido capturado (consulte Gravação do estado da atividade mais adiante). Sempre seguido de onStart().

- **onRestart()** - Chamado depois que atividade tiver sido interrompida, logo antes de ser reiniciada. Sempre seguido de onStart().

- **onStart()** - Chamado logo antes de a atividade se tornar visível ao usuário. Seguido de onResume() se a atividade for para segundo plano ou onStop() se ficar oculta.

- **onResume()** - Chamado logo antes de a atividade iniciar a interação com o usuário. Nesse ponto, a atividade estará no topo da pilha de atividades com a entrada do usuário direcionada a ela. Sempre seguido de onPause().

- **onPause()** - Chamado quando o sistema está prestes a retomar outra atividade. Esse método normalmente é usado para confirmar alterações não salvas a dados persistentes, animações interrompidas e outras coisas que talvez estejam consumindo CPU e assim por diante. Ele sempre deve fazer tudo bem rapidamente porque a próxima atividade não será retomada até ela retornar. Seguido de onResume() se a atividade retornar para a frente ou de onStop() se ficar invisível ao usuário.

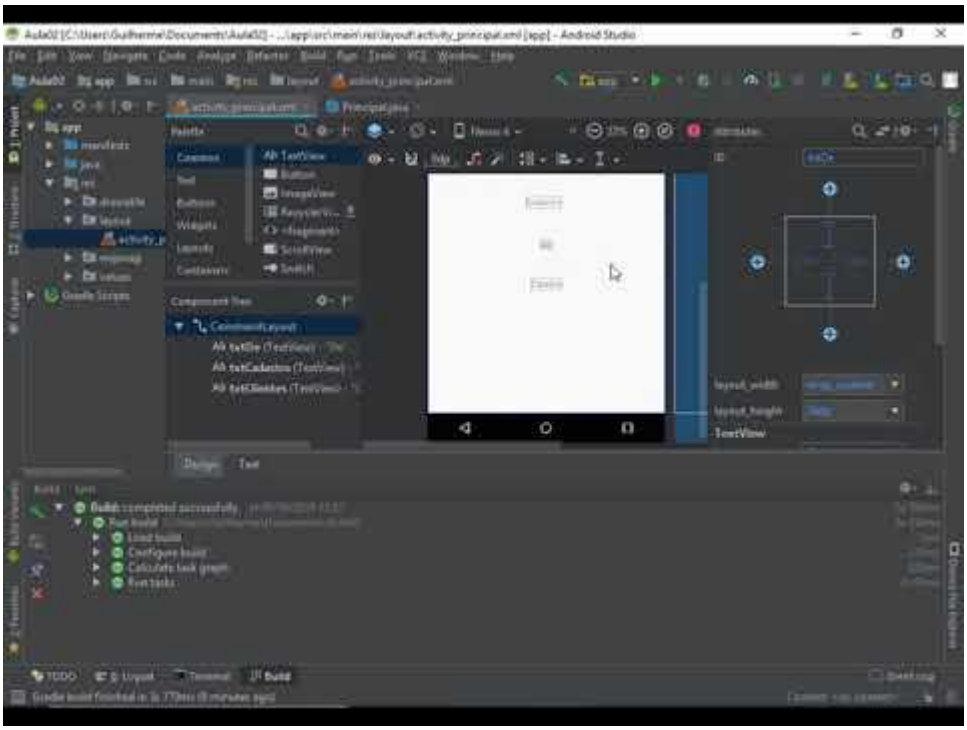
- **onStop()** - Chamado quando a atividade não está mais visível ao usuário. Isso pode acontecer porque ela está sendo destruída ou porque outra atividade (uma existente ou uma nova) foi retomada e está cobrindo-a. Seguido de onRestart() se a atividade estiver voltando a interagir com o usuário ou onDestroy() se estiver saindo.

- **onDestroy()** - Chamado antes de a atividade ser destruída. É a última chamada que a atividade receberá. Pode ser chamado porque a atividade está finalizando (alguém chamou finish() nela) ou porque o sistema está destruindo temporariamente essa instância da atividade para poupar espaço. É possível distinguir entre essas duas situações com o método isFinishing().

As definições anteriores foram retiradas da documentação oficial do Android Studio, disponível no site:

https://developer.android.com/guide/components/activities?hl=pt-br

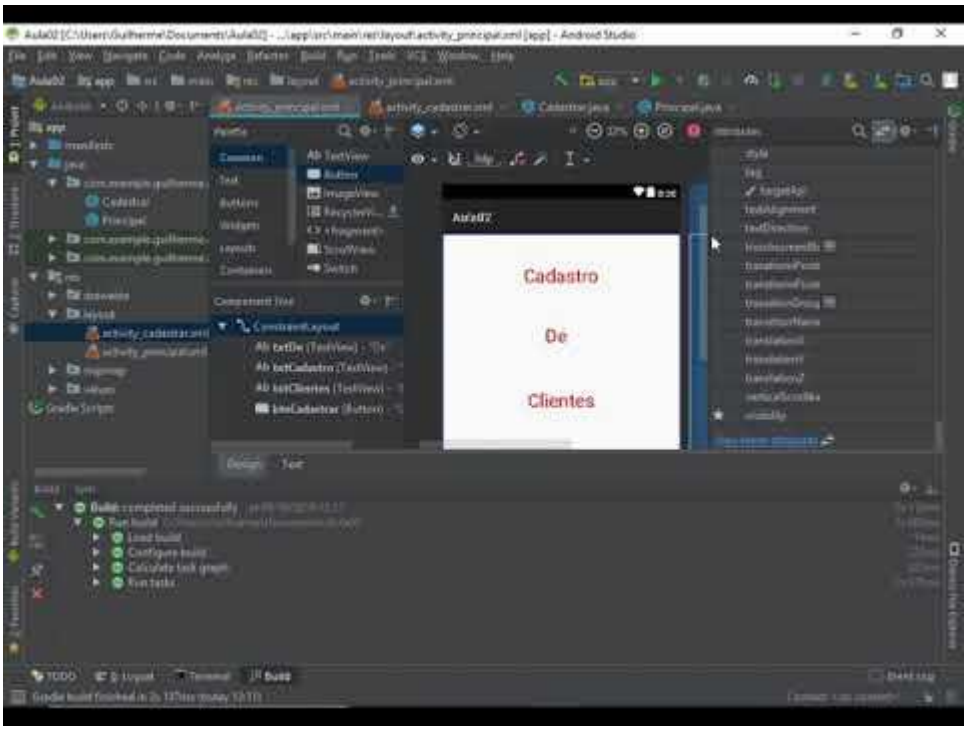
Assista a seguir o tutorial de criação de Activitys no Android Studio.



Componentes de texto e formatação de fontes e cores

O Android Studio é uma ferramenta poderosa, oferecendo aos desenvolvedores fantásticos recursos de interface. Podemos trabalhar nos componentes: cores, formatos de texto, tamanho e vários outros recursos.

No vídeo a seguir, abordamos algumas características e tipos de componentes de texto.



Transição entre Activity's

Para efetuar uma transição de tela no Android, é necessário informar a intenção da sua ação para o sistema operacional. O comando "Intent" informa essa intenção para o Android, que administra o ciclo de vida das Activitys.

O código a seguir mostra a utilização do "**Intent**" para chamar a Activity **Cadastrar**. Este comando é disparado pela ação de clique no botão "**btnCadastrar**" presenta na Activity **Principal**.

```java
package com.example.guilherme.aula02;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class Principal extends Activity {

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

        Button btnCadastrarProg = (Button) findViewById(R.id.btnCadastrar);

        btnCadastrarProg.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent it = new Intent(Principal.this,Cadastrar.class);
                startActivity(it);
            }
        });
    }
}
```

Verifique o vídeo para complementar os estudos.