

Universidade Federal do Amazonas - UFAM
Instituto de Ciências Exatas e Tecnologia - ICET
Laboratório de Eng. De Sistemas Computacionais e Robótica - LAB312



X Semana Nacional de Ciência e Tecnologia
Minicurso Android + Material Designer
Por: Cristian Reis e Ruddá Beltrão

Cristian Reis: christianreisoffice@gmail.com
Ruddá Beltrão: beltrao.rudah@gmail.com

Itacoatiara - AM
2016

Introdução.

Bem-vindo ao mundo android!!!

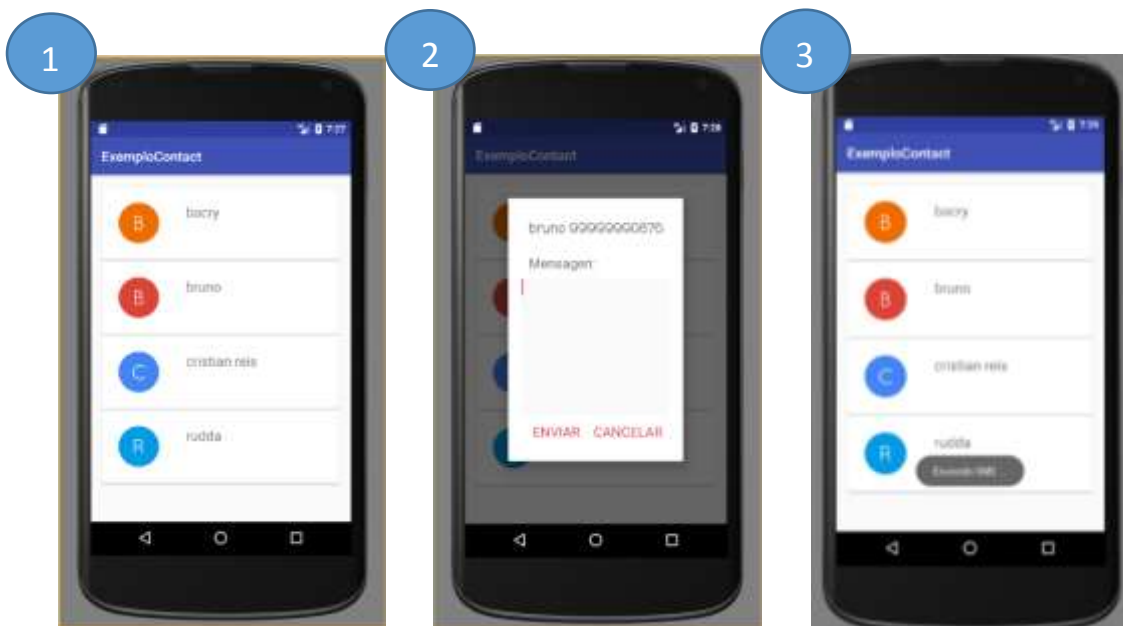
Neste Minicurso você será capaz de construir seu primeiro App android de acordo com Material Designer apenas seguindo passo-a-passo este material.

Os recursos estão disponíveis na **área de trabalho (DESKTOP)** numa pasta com o nome **MINICURSO ANDROID**.

Nesta pasta estão disponíveis alguns arquivos que usaremos no projeto dentre outros recursos.

Sobre o App.

O App que você vai construir é um simples App de envio de SMS para números da sua lista de contatos. Abaixo alguns screenshots.



1

Na tela 1 são listados todos os contatos do dispositivo em uma lista (RecyclerView) onde os itens da lista aparecem em formato de cartão (CardView) este formato de apresentação apresenta dois componentes importantes do MaterialDesigner.

2

A Segunda tela corresponde ao evento de clique de qualquer item da lista de contato. O nome deste componente se chama AlertDialogFragment, ele é inflado e aparece na tela no formato de um pop-up. Ele é designado para ações rápidas em que o usuário quer logo voltar para a tela principal. Neste App vamos usá-lo para enviar o sms para o destinatário da lista.

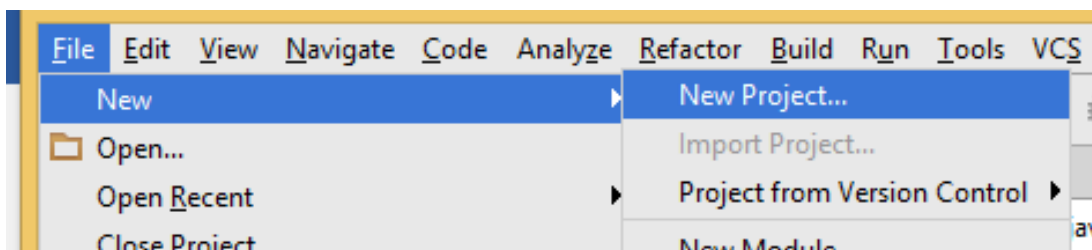
3

A terceira tela apenas apresenta uma mensagem para o usuário informando que seu SMS está sendo enviado. Caso exista algum erro no formato do número de destinatário é informado para o usuário uma mensagem de erro.

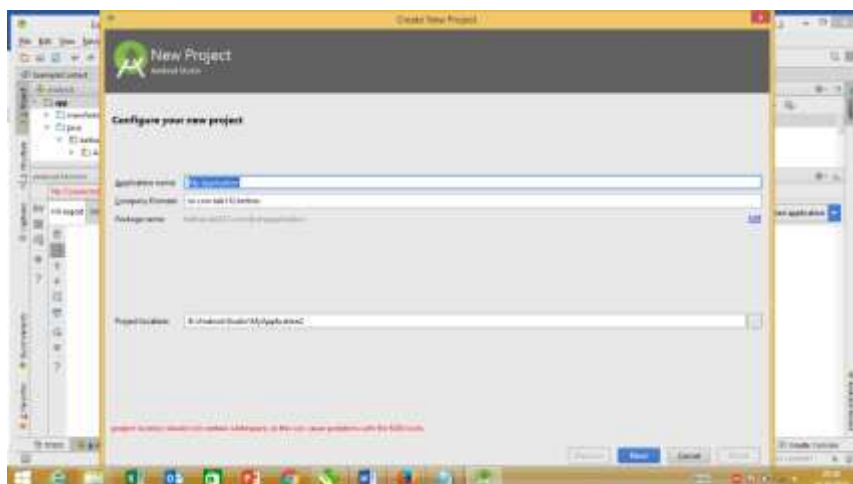
Mãos à Obra!!

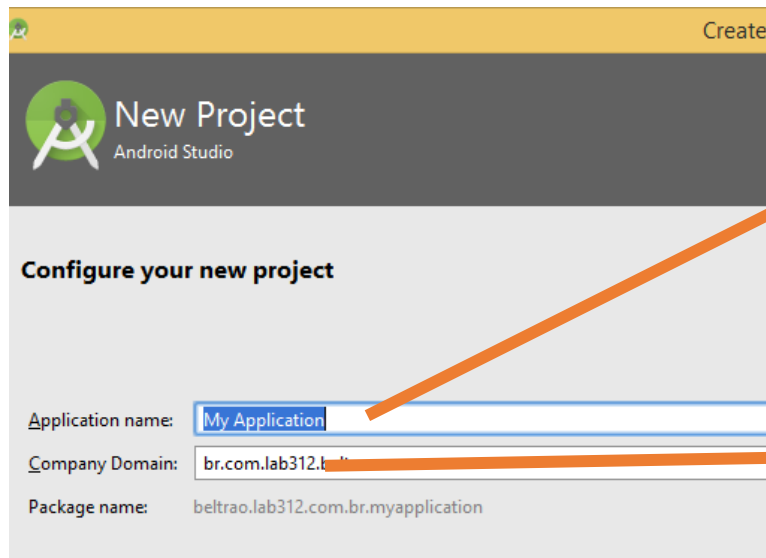
Agora que você já conhece o projeto está na hora de codificá-lo.

1º passo: abra o android Studio e Crie um Novo Projeto clicando em *File>> New>> New Project...* Conforme Imagem Abaixo:



Em seguida irá aparecer a seguinte caixa de diálogo:





1

Indique o Nome do Projeto neste caso My Application

2

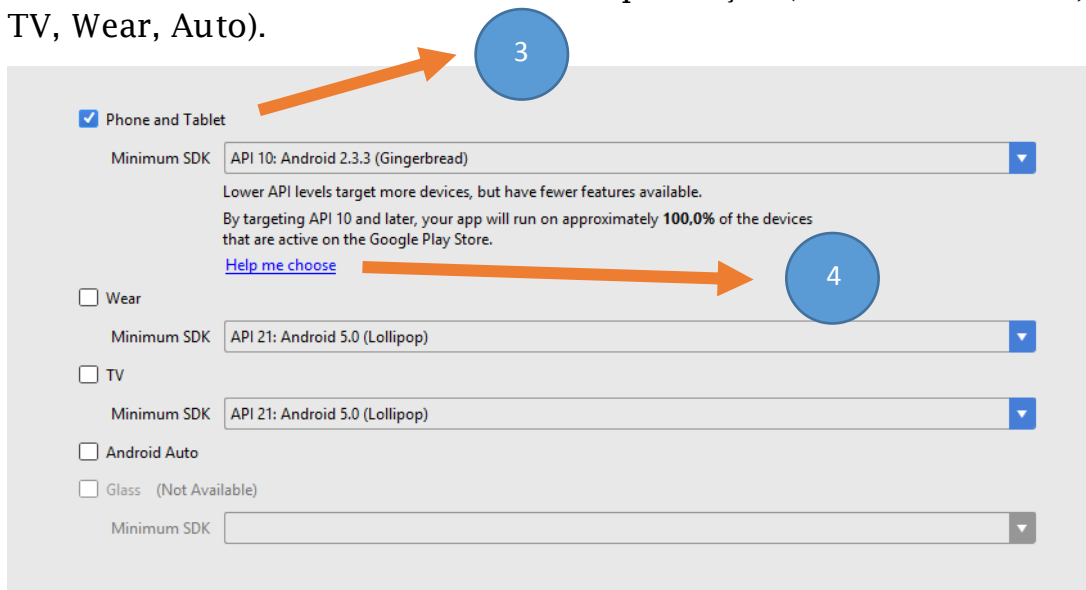
Indique o Nome do pacote principal do projeto neste caso

br.com.lab312.

Siga o padrão

br.com.INSTITUICAO.

Após realizar o passo 1 e 2 clique em **Next**. O passo seguinte consistem em escolhe os módulos de produção (Phone and Tablet, TV, Wear, Auto).



3

☒ Phone and Tablet

Minimum SDK API 10: Android 2.3.3 (Gingerbread)

Lower API levels target more devices, but have fewer features available.

By targeting API 10 and later, your app will run on approximately **100,0%** of the devices that are active on the Google Play Store.

[Help me choose](#)

☐ Wear

Minimum SDK API 21: Android 5.0 (Lollipop)

☐ TV

Minimum SDK API 21: Android 5.0 (Lollipop)

☐ Android Auto

☐ Glass (Not Available)

Minimum SDK

4

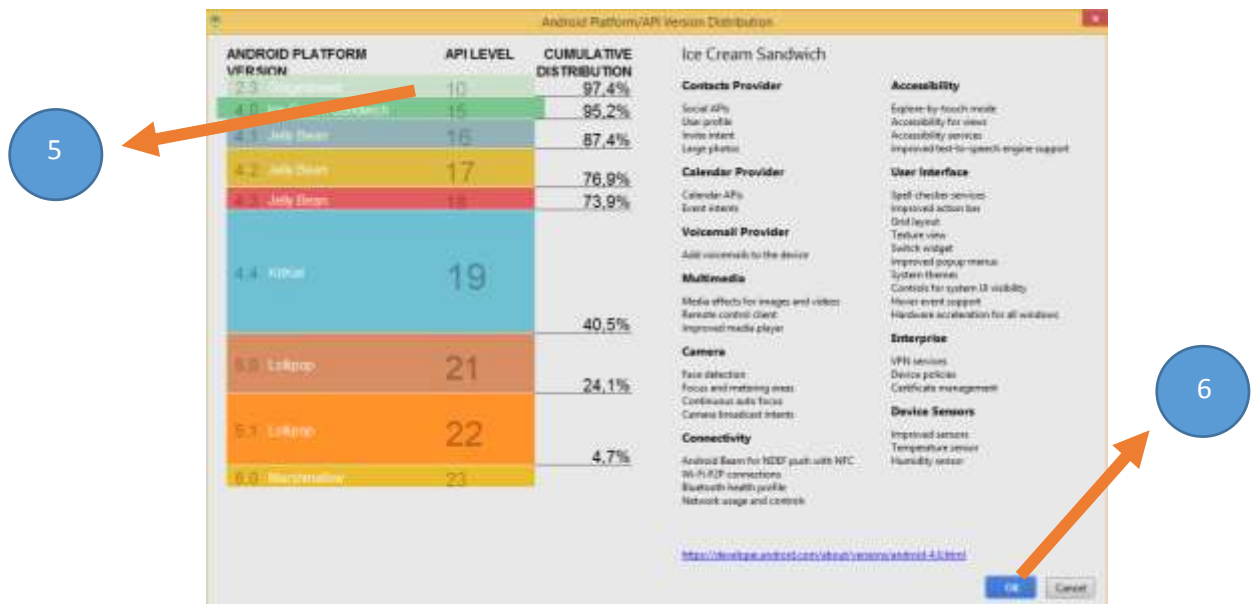
3

Selecione Phone and Tablet

4

Clique em Help me choose.

E então você verá a seguinte tela no computador...

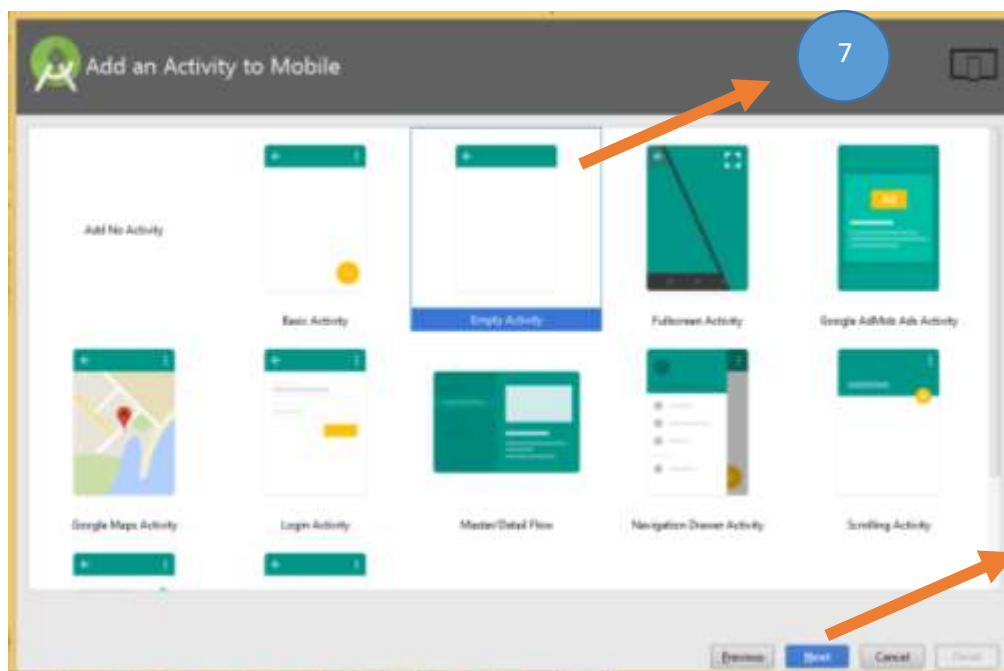


5

Apenas **Clique** em 2.3 API level 10

6

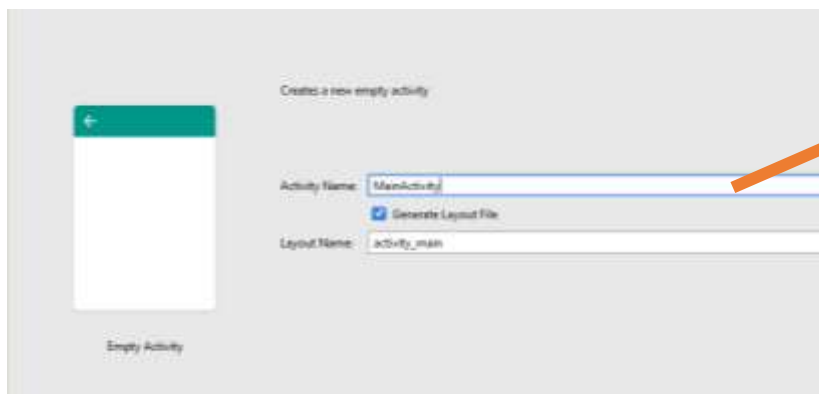
Clique em **Ok**, você vai **retornar** a Tela Anterior Então clique em **Next**



7

Nesta etapa você irá escolher o tipo de Activity que irá trabalhar (Maps, navigation Drawers, Login Activity, etc) selecione **Empty Activity** e clique em **Next**

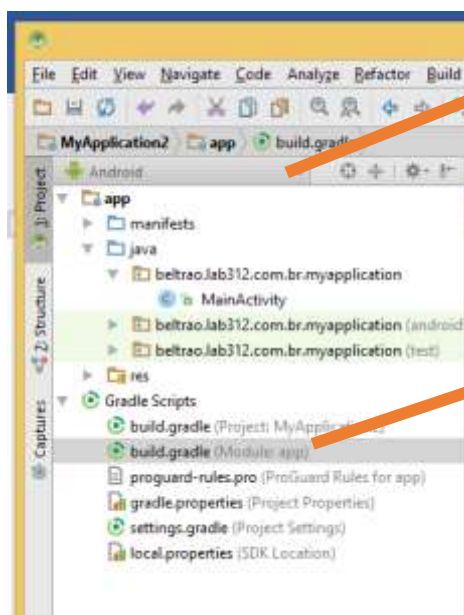
8



Defina um nome para sua activity principal e clique em finish

2º Passo: Para deixar as iniciais dos nomes dos contatos dentro de um círculo colorido, precisamos importa uma biblioteca chamada `materiallettericon`¹ disponível no github. Para fazer isso precisamos usar o gradle, que é gerenciador de dependências do android. Para fazer isso é bastante simples clique em **build.gradle:app** e em **dependencies** adicione o seguinte código.

```
compile 'com.github.ivbaranov:materiallettericon:0.2.2'
```



Escolha a Visão android, caso esta tela não apareça pressione Alt+1

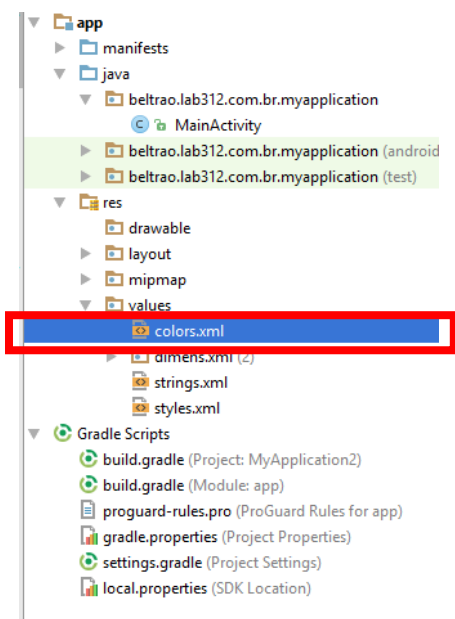
Clique duas vezes sobre o arquivo build.gradle (Modulo app) e então irá aparecer o conteúdo do arquivo.

¹ <https://github.com/IvBaranov/MaterialLetterIcon>



Primeiro adicione o **código** em dependencies e então aparecerá uma mensagem para sincronizar o gradle, clique em **Sync Now**.

3º passo: Após sincronizar o gradle, vamos configurar as cores de fundo que serão usadas no projeto, para isso você deve navegar até o arquivo **app/res/values/colors.xml** conforme indicado na figura abaixo:



Seu arquivo colors.xml deve ficar da seguinte maneira:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="color_1">#DB4437</color>
    <color name="color_2">#E91E63</color>
    <color name="color_3">#FF5722</color>
    <color name="color_4">#EF6C00</color>
    <color name="color_5">#689F38</color>
    <color name="color_6">#0097A7</color>
    <color name="color_7">#4285F4</color>
    <color name="color_8">#039BE5</color>
    <color name="color_9">#3F51B5</color>
    <color name="color_10">#673AB7</color>
    <color name="color_11">#757575</color>
    <color name="white">#ffffff</color>
    <color name="black">#000000</color>

    <array name="colors">
        <item>@color/color_1</item>
        <item>@color/color_2</item>
        <item>@color/color_3</item>
        <item>@color/color_4</item>
        <item>@color/color_5</item>
        <item>@color/color_6</item>
        <item>@color/color_7</item>
        <item>@color/color_8</item>
        <item>@color/color_9</item>
        <item>@color/color_10</item>
        <item>@color/color_11</item>
    </array>
</resources>
```



Adicionamos ao arquivos 11 cores color_1, color_2 ...

Os valores das cores estão representados em hexadecimal entre as tags <color></color>

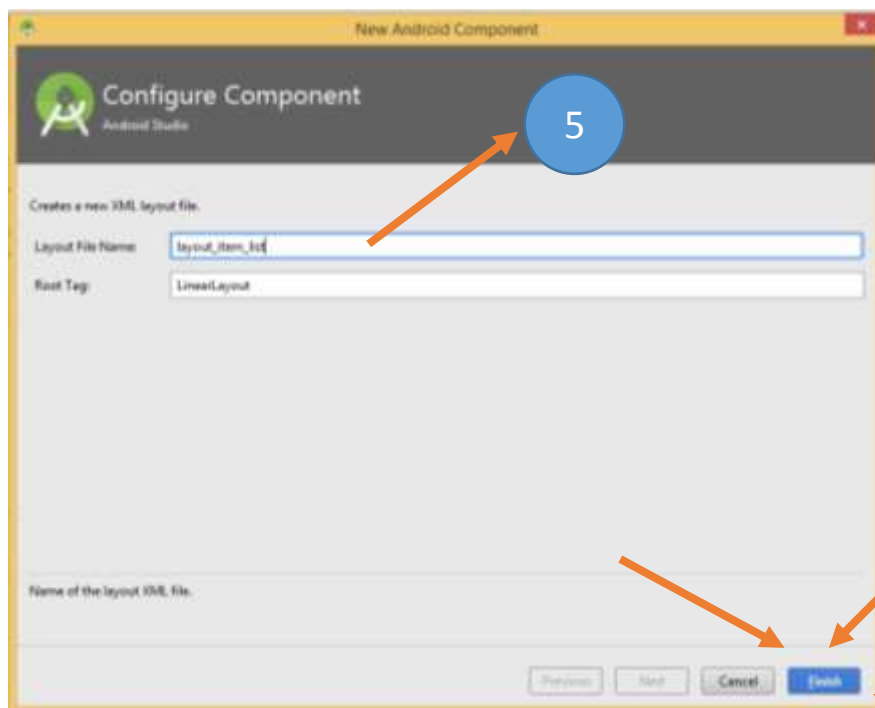
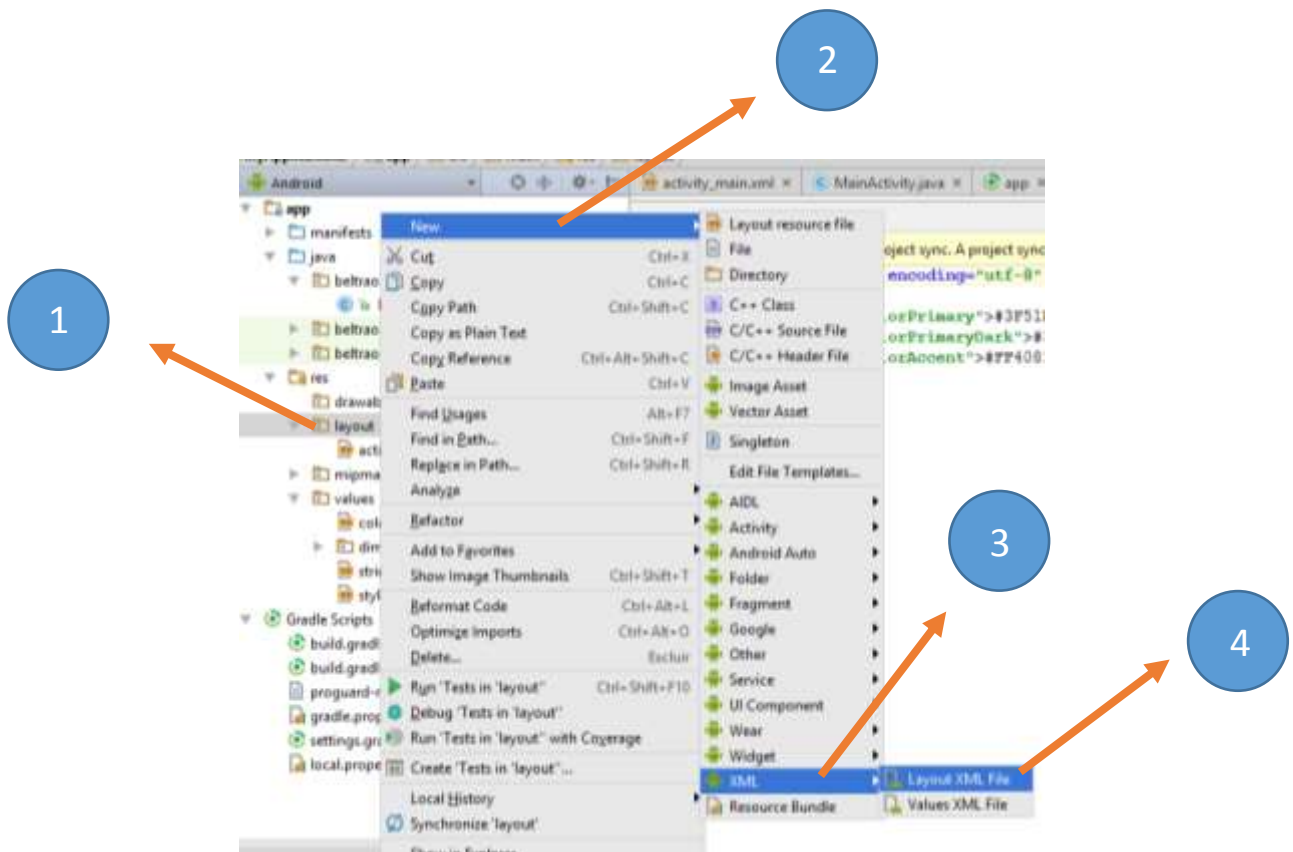


Criamos uma array com as 11 cores.

Nome do array: **colors**

4º passo: vamos criar um layout para os itens da lista. Então precisamos criar um arquivo no diretório layout.

1. Clique com botão direito do mouse em Layout.
2. Selecione a opção New.
3. Selecione XML.
4. Clique em layout XML file
5. Irá aparecer na tela um caixa de diálogo, informe o nome do arquivo como **layout_item_list** do layout e clique em finish



5º passo: Vamos editar o arquivo layout_item_list conforme indicado abaixo:

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

```
<android.support.v7.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="1dp"
android:elevation="10dp"
app:contentPadding="4dp"

>
```

1

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
```

2

```
<com.github.ivbaranov.mli.MaterialLetterIcon
    android:id="@+id/materialLetterCollorID"
    android:layout_width="60dp"
    android:layout_height="60dp"
    android:layout_margin="@dimen/letter_margin"
    app:mli_initials="false"
    app:mli_initials_number="2"
    app:mli_letter="S"
    app:mli_letter_color="@color/white"
    app:mli_letter_size="26"
    app:mli_letters_number="1"
    app:mli_shape_color="@color/black"
    app:mli_shape_type="circle" />
```

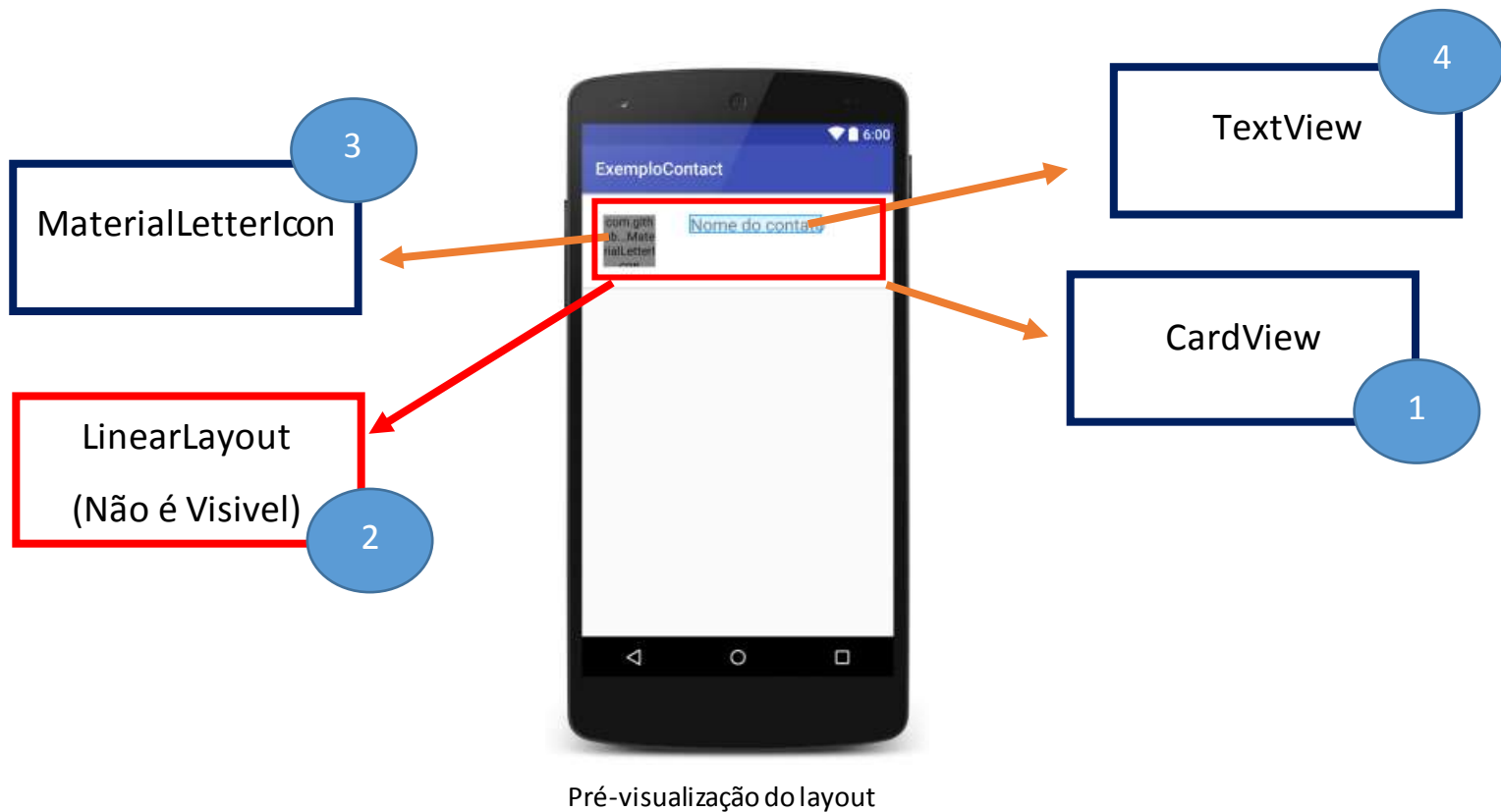
3

```
<TextView
    android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/letter_margin"
    android:maxLines="1"
    android:text="nome do contato"
    android:textSize="20dp" />
```

4

```
</LinearLayout>
```

```
</android.support.v7.widget.CardView>
```



Explicando os componentes:

1

<android.support.v7.widget.CardView>

Este componente é responsável pelo comportamento de agrupar as Views em formato de cartão.

2

<LinearLayout>

Este componente é necessário para organizar os elementos na tela

3

<com.github.ivbaranov.mli.MaterialLetterIcon>

Componente responsável por Capturar a Letra Inicial do Nome de Contato e inflar um View indicadora na tela.

android:id="@+id/materialLetterCollorID" é um identificador único do componente no layout

4

<TextView >

Exibe um elemento de Texto

android:id="@+id/name" é um identificador único do componente no layout

6º passo: Agora vamos montar o layout do XML para a Activity principal (MainActivity). Nela vamos inserir apenas um RecyclerView. O layout xml fica desta forma:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:orientation="vertical"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="beltrao.lab312.com.br.exemplocontact.MainActivity">

    <android.support.v7.widget.RecyclerView
        android:layout_width="match_parent"
        android:id="@+id/mRecycler"
        android:layout_height="match_parent">

    </android.support.v7.widget.RecyclerView>

</LinearLayout>
```

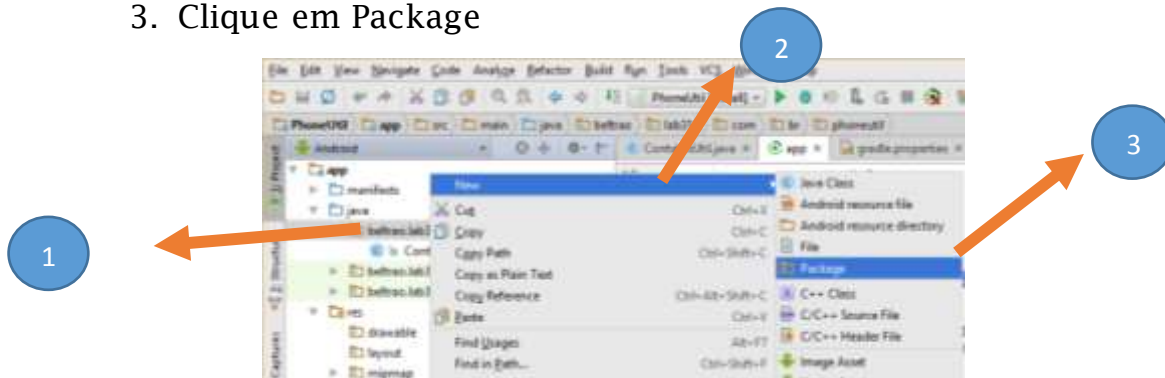
Fique atento ao `android:id="@+id/mRecycler"` que é como vamos referenciar este componente mais tarde. Abaixo a pre-visualização do Layout.



7º passo: Copie a classe **ContatosUtil** da Pasta MINICURSO ANDROID para o seu projeto.

8º passo vamos criar um classe adapter para inflar os itens para o recyclerView, mas primeiro vamos criar um pacote (package) para uma melhor organização do código, para isso:

1. Clique com o botão direito do mouse no pacote principal
2. Selecione New
3. Clique em Package

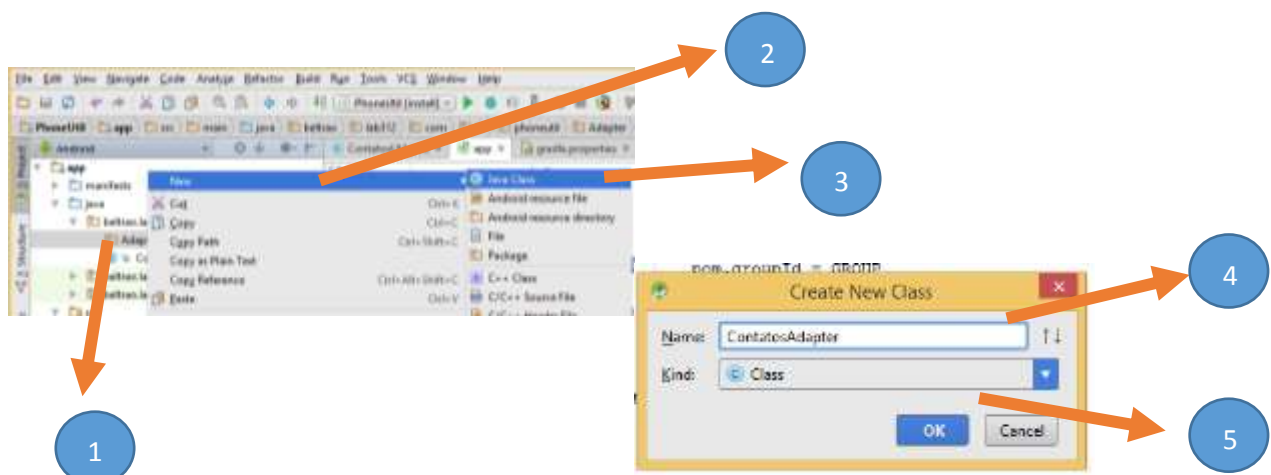


Irá ficar visível na tela uma caixa de diálogo, informe o nome do pacote como: **Adapter** e clique em ok.



9º passo: Feito isso vamos criar uma nova classe:

1. clique com botão direito do Mouse no pacote que você acabou de criar,
2. depois clique em **new**
3. Clique em **New Class**
4. Informe o **nome da Classe** e clique em **ok**.



10º passo: Agora vamos começar a codificar a classe ContactAdapter.

Primeiramente iremos criar um classe interna chamada holder responsável pela vinculação entre Java e o layout Xml que criamos. É necessário também que esta classe estenda a classe RecyclerView.ViewHolder para então possa ser usada no nosso adapter.

```
public class ContactAdapter {  
  
    class holder extends RecyclerView.ViewHolder{  
  
        public TextView name;  
        public MaterialLetterIcon mMaterialLetterIcon;  
  
        public holder(View itemView) {  
            super(itemView);  
  
            mMaterialLetterIcon = (MaterialLetterIcon)  
itemView.findViewById(R.id.materialLetterCollorID);  
  
            name = (TextView) itemView.findViewById(R.id.name);  
  
        }  
    }  
}
```

1

1 Neste trecho do código é quando realmente fazemos a referência dos objetos java com o layout Xml por meio do método findViewById e referenciando o ID que inserimos no layout xml de cada componente.

Feito isso vamos estender a classe ContactAdapter para que ela possa herdar atributos e métodos da classe RecyclerView.Adapter<T> como demonstrado no código abaixo:

```
public class ContactAdapter extends  
RecyclerView.Adapter<ContactAdapter.holder> { }
```

Note que só acrescentamos a seguinte linha no código.

```
extends RecyclerView.Adapter<ContactAdapter.holder>
```

A classe interna (holder) que criamos

Agora vamos criar um método Construtor para nossa classe, mas antes vamos precisar de alguns atributos são eles:

```
private List<ContatosUtil> nomes;
```

Precisamos de uma lista contendo o nome e telefone dos contatos existentes no dispositivo

```
private Context com;
```

Contexto da aplicação (Activity, fragmente etc)

```
private Typeface custom_font;
```

Fonte do texto

```
private AppCompatActivity mAppCompatActivity;
```

Referência de uma Activity

para criar nosso construtor vamos precisar de 4 dos 5 atributos acima, nosso método construtor ficará dessa forma:

```
public ContactAdapter(List<ContatosUtil>nomes, Context com,
AppCompatActivity mAppCompatActivity, Class<?> package) {

    this.nomes = nomes;
    this.com = com;
    this.mAppCompatActivity = mAppCompatActivity;
    this.package = package;
    // custom_font = Typeface.createFromAsset(com.getAssets(),
    "quicksand.ttf");

}
```

Agora vamos sobrescrever o método onCreateViewHolder, este método é responsável por inflar o layout que criamos (layout_item_lista) em uma **view** para ser passada como parâmetro para o objeto **mHolder**;

```
@Override
public holder onCreateViewHolder(ViewGroup parent, int viewType) {

    View v =
    LayoutInflater.from(com).inflate(R.layout.layout_item_lista, parent,
    false);

    holder mHolder = new holder(v);

    return mHolder;

}
```

Passamos a sobrescrever o método onBindViewHolder que é onde realmente a “magica” acontece, então, imagine uma lista de 20 nomes e seus respectivos números de telefone, este trecho do código irá se repetir para todos os 20 contatos e inserir-los um a um como um elemento da lista do recyclerView.

Para fins didático vamos dividir esse código em duas partes:

Parte 1 - Configurando valores para os elementos da lista.

Primeiramente observe a assinatura do Método.

```
@Override
public void onBindViewHolder(holder holder, final int position)
```

Bom, você deve se lembrar do método anterior ele infla o layout e instancia um objeto Holder, no entanto o próximo passo que o RecyclerView faz é chamar o onBindViewHolder passando por parâmetro o objeto Holder que você criou. E adivinha onde ele está agora? Isso garoto (a) espeto (a) na assinatura deste método.

Vamos usá-lo agora para fazer referência ao Layout [layout_item_lista](#) e assim inflar todos os itens na lista de contatos.

Agora vamos para o próximo passo, observe o código abaixo que será inserido dentro do método onBindViewHolder:

```
int[] mMaterialColors =  
com.getResources().getIntArray(R.array.colors);
```

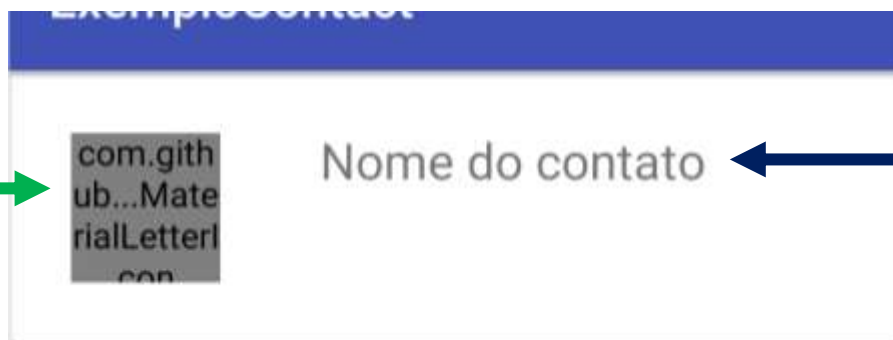
```
Random RANDOM = new Random();
```

Bom, nesta etapa do projeto, você não deve mais lembrar da primeira coisa que fizemos certo? Então eu te dou uma ajudinha, nós definimos no arquivo **res/values/colors.xml** 11 cores e seus respectivos valores e ainda mais, definimos um array (um vetor) contendo todas essas cores.

O código acima simplesmente **instancia um vetor** inteiro que faz **referência** ao array (vetor) contido no arquivo **colors.xml**. Deste modo temos todas as cores que precisamos armazenadas na variável **mMaterialColors**.

Mas, e o objeto Random? Para que server? A Classe Random server para sortear um número dentre um determinado limiter, usaremos ela para sortear uma cor aleatória do vetor de cores.

Feito isso vamos para a próxima etapa. Para melhor compreensão, observe a imagem abaixo antes de ler o código.



```
holder.mMaterialLetterIcon
```

```
holder.name
```

Quando você criou o Layout o AndroidStudio gerou esta Preview para você, estamos inflando este layout numeras vezes dentro de uma lista, em Java cada item deste layout está referenciado dentro do objeto Holder como ilustra a imagem.

A partir deste ponto já podemos escrever o código abaixo:

```

1 holder.name.setText(nomes.get(position).getNome());
2
3 holder.mMaterialLetterIcon.setLetterSize(30);
4 holder.mMaterialLetterIcon.setLettersNumber(1);
5 holder.mMaterialLetterIcon.setLetterTypeface(custom_font);
6 holder.mMaterialLetterIcon.setLetterColor(Color.WHITE);
7 holder.mMaterialLetterIcon
8 .setShapeColor(mMaterialColors[RANDOM.nextInt(mMaterialColors.length)
9 ]);
10 holder
11 .mMaterialLetterIcon.setLetter(nomes.get(position).getNome());

```

Na primeira linha informamos que o nome a ser exibido será um dos nomes contidos no arrayList (lista) de contatos através do metodo nomes.get(position).getNome();

Nas Linhas 3 à 11 configuramos o item que aparece a Inicial do nome do Contato, nelas configuramos tamanho (**setLetterSize(30)**) número de letras que aparecerão (**setLetterNumber(1)**), o tipo de fonte caso haja (**setLetterTypeface()**), a cor da letra em destaque (**setLetterColor()**), a cor de background (**setShaperColor()**) e quais letras serão usadas (**setLetter()**);

Feito isso terminamos a parte 1.

Parte 2 – Eventos de Clique.

Observe o código abaixo:

```

holder.itemView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Toast.makeText(com, "Nome: "+nomes.get(position).getNome(),
        Toast.LENGTH_LONG).show();

    }
});

```

Neste trecho de código ao clicar exibe uma mensagem para o usuário informando o nome do contato em que ele clicou.

O código completo ficará assim:

```
public class ContactAdapter extends RecyclerView.Adapter<ContactAdapter.holder> {

    private List<ContatosUtil> nomes;
    private Context com;
    private Typeface custom_font;
    private AppCompatActivity mAppCompatActivity;

    public ContactAdapter(List<ContatosUtil> nomes, Context com, AppCompatActivity
mAppCompatActivity) {

        this.nomes = nomes;
        this.com = com;
        this.mAppCompatActivity = mAppCompatActivity;
        custom_font = Typeface.createFromAsset(com.getAssets(), "quicksand.ttf");
    }

    @Override
    public holder onCreateViewHolder(ViewGroup parent, int viewType) {

        View v = LayoutInflater.from(com).inflate(R.layout.layout_contatos, parent,
false);
        holder mHolder = new holder(v);
        return mHolder;
    }

    @Override
    public void onBindViewHolder(holder holder, final int position) {
        int[] mMaterialColors = com.getResources().getIntArray(R.array.colors);
        Random RANDOM = new Random();

        holder.name.setText(nomes.get(position).getNome());
        holder.mMaterialLetterIcon.setLetterSize(30);
        holder.mMaterialLetterIcon.setLettersNumber(1);
        holder.mMaterialLetterIcon.setLetterTypeface(custom_font);
        holder.mMaterialLetterIcon.setLetterColor(Color.WHITE);

        holder.mMaterialLetterIcon.setShapeColor(mMaterialColors[RANDOM.nextInt(mMaterialColors.1
ength)]);
        holder.mMaterialLetterIcon.setLetter(nomes.get(position).getNome());

        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

            }
        });
    }

    @Override
    public int getItemCount() {

        return nomes.size();
    }
}
```

```
class holder extends RecyclerView.ViewHolder {
    public TextView name;
    public MaterialLetterIcon mMaterialLetterIcon;

    public holder(View itemView) {
        super(itemView);

        mMaterialLetterIcon = (MaterialLetterIcon)
itemView.findViewById(R.id.materialLetterCollorID);
        name = (TextView) itemView.findViewById(R.id.name);
    }
}
```

X Passo: MainActivity .

Agora que já concluímos o adapter do RecyclerView está na hora de vê-lo funcionando. Para isso abra o arquivo MainActivity.java e declare as seguinte variáveis.

```
1. private List<ContatosUtil> contatos;  
2. private ContactAdapter adp;  
3. private RecyclerView r;
```

1. precisamos de uma lista de contatos, cada item será uma linha da lista a ser exibida e inflada como vimos na criação do adapter.
2. Declaramos o adapter que criamos.
3. E declaramos um RecyclerView para fazer referencia ao recyclerView do layout xml que criamos.

Agora o passo seguinte consiste em sobrescrever o método onCreate() e vincular o RecyclerView java com o RecyclerView do nosso xml através do método findViewById() como demonstrado abaixo:

```
r = (RecyclerView) findViewById(R.id.mRecycler);
```

Ainda no onCreate vamos inicializar nossa lista de Contatos.

```
contatos = new ArrayList<>();
```

O recyclerView é um componente que gráfico bastante poderoso e pode assumir o comportamento de Lista (nosso caso) ou de grid (linhas e colunas) o responsável por esse comportamento é outro componente chamado LayoutManager que pode ser um LinearLayoutManager ou GridLayoutManager.

O LinearLayout Manager assume o comportamento de agrupar seus componentes um abaixo do outro (Orientação Vertical - ORIENTATION VERTICAL) ou um ao lado do outro (Orientação Horizontal - ORIENTATION HORIZONTAL).

O `GridLayoutManager` tem o comportamento de agrupar seus componentes na forma de linhas e colunas como uma tabela.

No nosso caso queremos uma lista de contatos, para isso vamos usar o `LinearLayoutManager` na orientação vertical, segue código abaixo:

```
LinearLayoutManager ll = new  
LinearLayoutManager(getApplicationContext());  
ll.setOrientation(LinearLayoutManager.VERTICAL);  
r.setLayoutManager(ll);
```

Na primeira e segunda linha instanciamos um novo `LinearLayoutManager` passando para o construtor um context.

Na terceira linha definimos sua orientação, neste caso vertical.

Na ultima linha definimos que o `layoutManager` a ser usado pelo `recyclerView` será o nosso `LinearLayoutManager`.

Pronto!! Quase tudo feito. Talvez você não saiba mas a classe que copiamos da área de trabalho chamada `ContatosUtil` contém métodos encapsulados para listar os contatos do dispositivo, ela retorna uma Lista de `ContatosUtil` o qual já inicializamos no início da `MainActivity`.

No entanto, imagine que você é uma pessoa bastante popular e conhece mil pessoas, ou melhor 10 mil pessoas, nossa são muitas informações para serem processadas não acha? Então vamos processa-las separadamente em outro processo, para fazer isso vamos precisar de uma `Thread`. Observe o código abaixo para criar uma nova `Thread`.

```
new Thread(new Runnable() {  
    @Override  
    public void run() {  
  
    }  
    });  
}).start();
```

Cria uma nova Thread

Todas as tarefas a serem executadas dentro desta thread tem que estar dentro do método `run()`{ }

Inicia a Thread.

Pronto já criamos uma nova Thread, isso significa que os códigos que inserirmos dentro do método run() serão realizados fora da Thread principal, ou seja, fora da activity.

Agora vamos inserir o código para listar todos os contatos do dispositivo, é bastante simples!!

```
contatos = ContatosUtil.build(MainActivity.this).getContatos();
```

Pronto!! Viu como foi fácil? Já temos todos os contatos, agora precisamos voltar para MainActivity e inflar no recyclerView todos os Contatos para isso vamos usar o método runOnUiThread() ele volta a thread principal.

```
runOnUiThread(new Runnable() {  
    @Override  
    public void run() {  
  
    }  
});
```

Chamamos a Thread da MainActivity ou seja voltamos para MainActivity

Tudo o que fizemos dentro do método run() será executado na MainActivity

Agora que já estamos de Volta na MainActivity ja podemos instanciar nosso adapter para inflar todos os element da lista.

1

```
adp = new ContactAdapter(contatos,  
    MainActivity.this, MainActivity.this);
```

2

```
adp.notifyDataSetChanged();  
r.setAdapter(adp);
```

1

Inicializamos o adapter passando como parâmetro nossa lista de contatos, nosso contexto de aplicação e nossa Activity;

2

Informamos que os dados do adapter sofreram alterações.
E definimos que o adapter do recyclerView é o adapter que criamos

O código completo da MainActivity fica assim:

```
public class MainActivity extends AppCompatActivity {

    private List<ContatosUtil> contatos;
    private ContactAdapter adp;
    private RecyclerView r;

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

        contatos = new ArrayList<>();

        r = (RecyclerView) findViewById(R.id.mRecycler);

        LinearLayoutManager ll = new LinearLayoutManager(getApplicationContext());
        ll.setOrientation(LinearLayoutManager.VERTICAL);
        r.setLayoutManager(ll);

        new Thread(new Runnable() {
            @Override
            public void run() {

                contatos = ContatosUtil.build(MainActivity.this).getContatos();

                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {

                        adp = new ContactAdapter(contatos, MainActivity.this,
MainActivity.this);
                        adp.notifyDataSetChanged();
                        r.setAdapter(adp);

                    }
                });
            }
        }).start();

    }

}
```

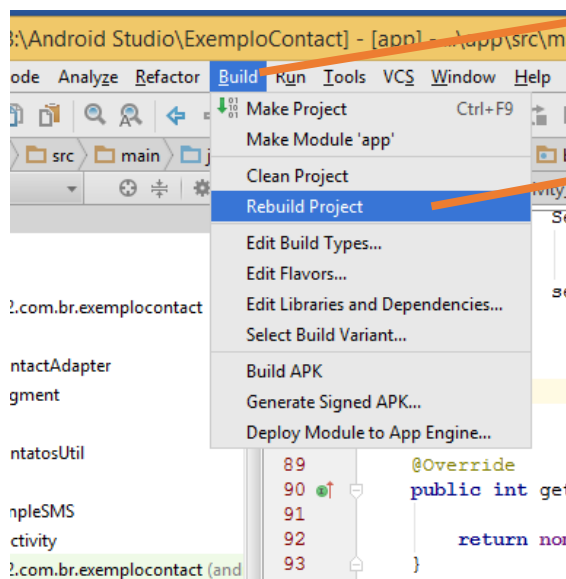

O Resultado final será algo parecido com isso:



Para executar o projeto clique em Build, Rebuild Project. Depois clique em executar.

Se você tiver um celular é preciso dar permissão em depuração USB no dispositivo.

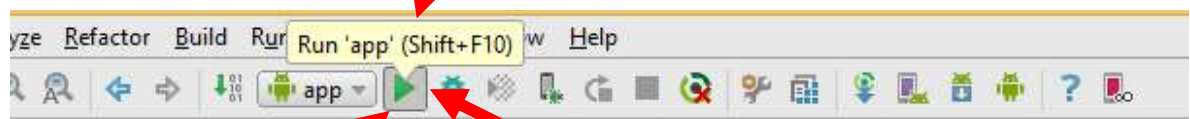
Caso você esteja usando um Emulador isso não será necessário, mas fico atento, inicie-o o quanto antes, pois seu carregamento é um pouco demorado.



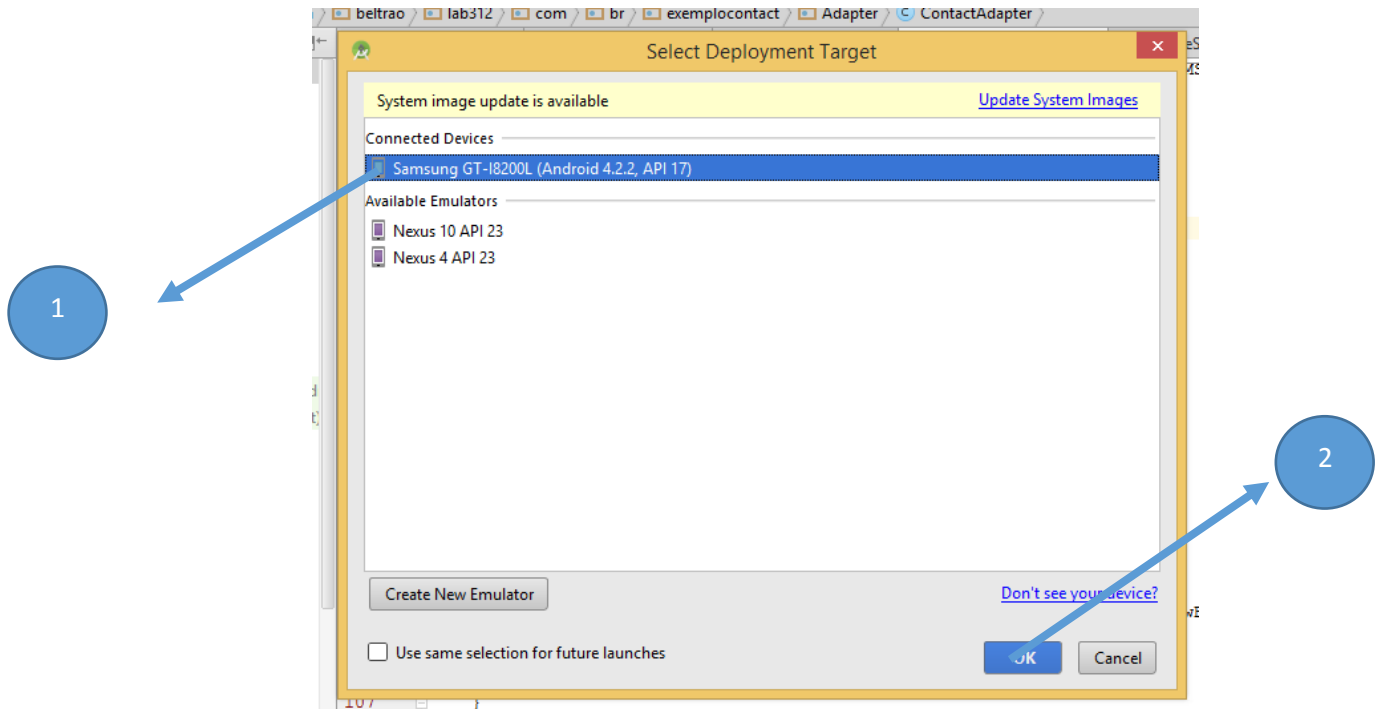
Clique em Build

Clique em Rebuild Project e aguarde termino da tarefa

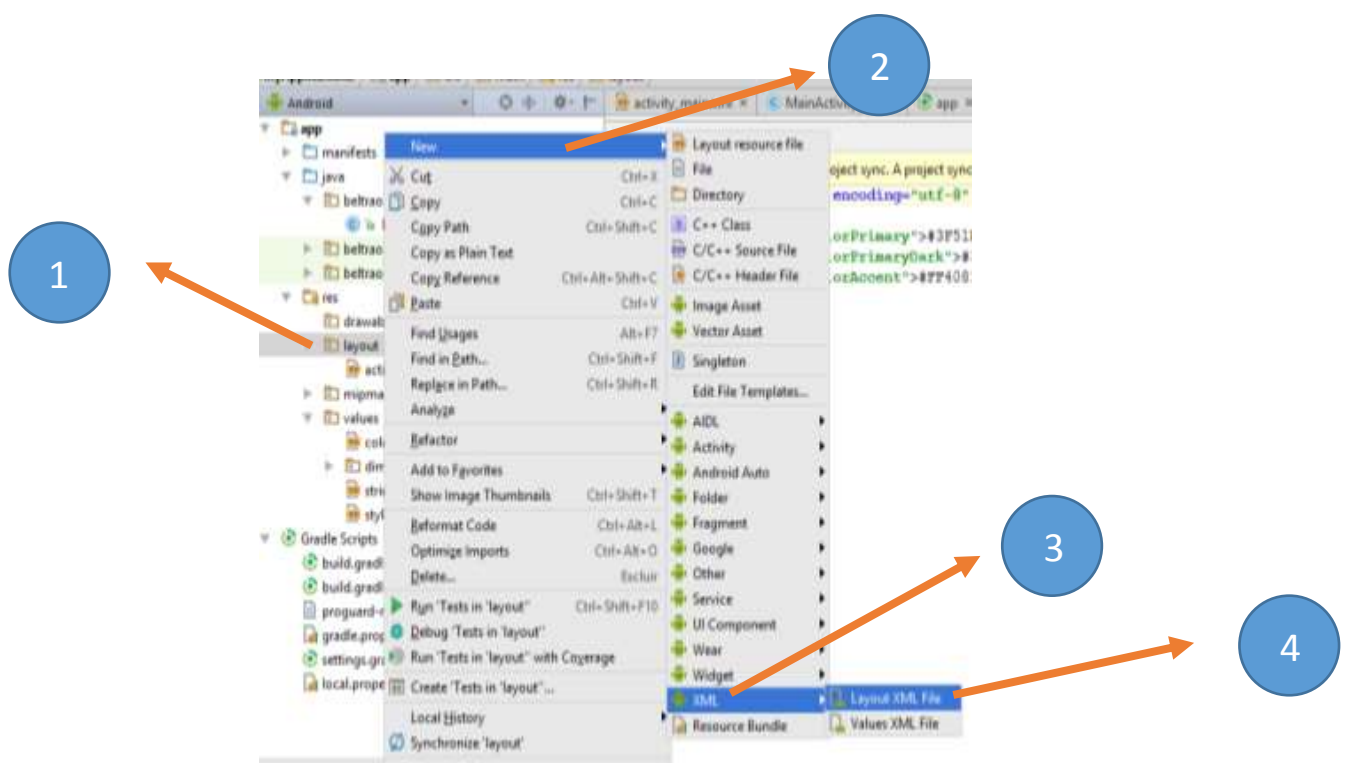
Para executar ...

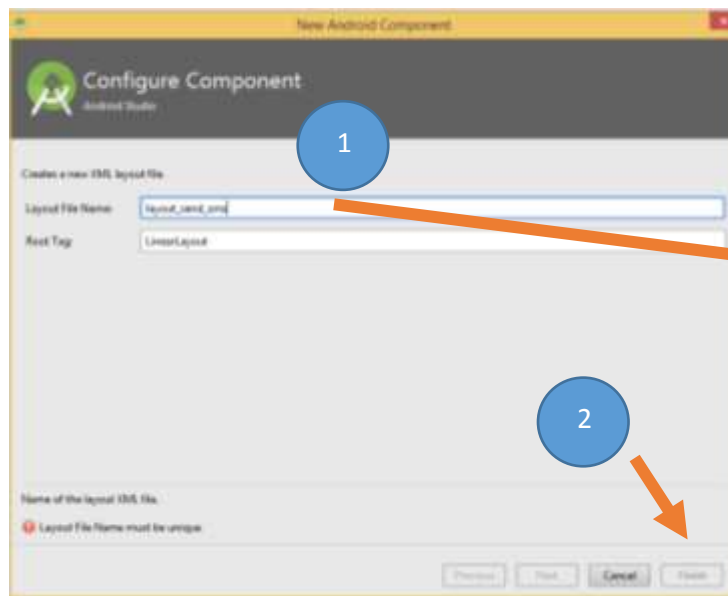


Selecione um dispositivo ou Emulado e clique em OK.



Agora que criamos a lista de contatos vamos criar a interface que o usuário vai enviar o sms, para isso vamos criar o outro layout xml. Conforme o Passo 4 deste Tutorial

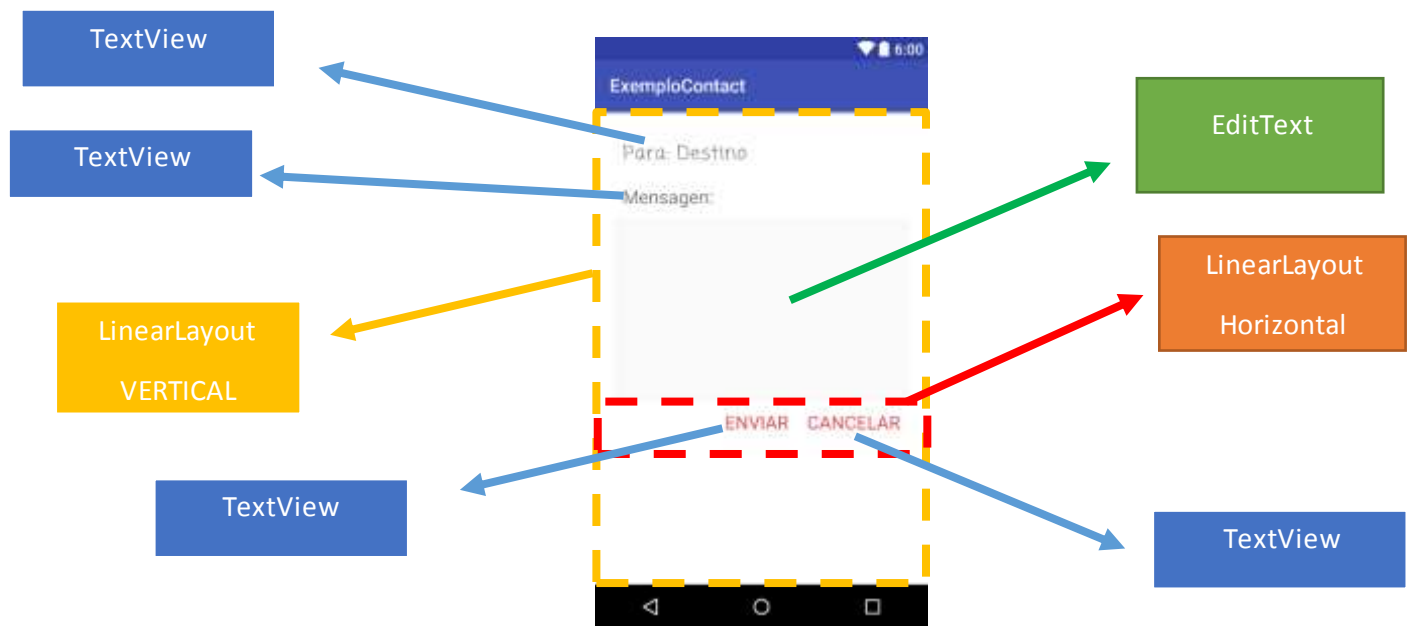




Em Layout file name

Informe o nome
layout_send_sms e clique
em finish

Agora vamos editar o Layout para que se ajuste a esse formato:



O Arquivo xml completo fica assim:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="20dp"
    android:background="#ffffff"
    android:orientation="vertical"
    >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Para: Destino"
        android:layout_margin="10dp"
        android:id="@+id/destino"
        android:fontFamily="casual"
        android:textStyle="bold"
        android:textSize="20dp"
    />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mensagen:"
        android:layout_margin="10dp"
        android:textSize="20dp"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:id="@+id/msg"
        android:gravity="top|left"
        android:background="#fafafa"

    />

    <LinearLayout
        android:layout_width="wrap_content"
        android:orientation="horizontal"
        android:layout_gravity="right"
        android:layout_height="wrap_content">

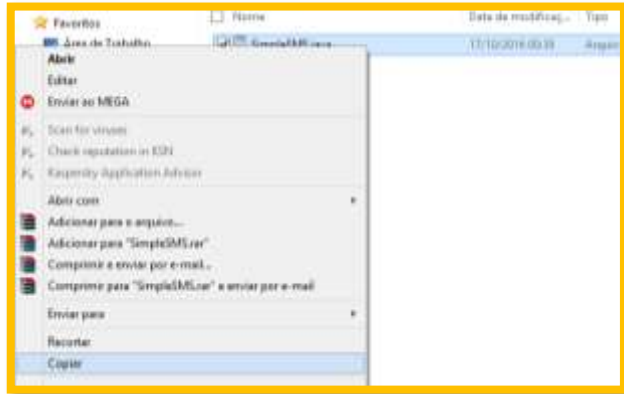
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="ENVIAR"
            android:id="@+id/enviar"
            android:textSize="20dp"
            android:textColor="#e2424a"
            android:layout_margin="10dp"
        />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="CANCELAR"
            android:textSize="20dp"
            android:id="@+id/cancelar"
            android:textColor="#e2424a"
            android:layout_margin="10dp"

        />

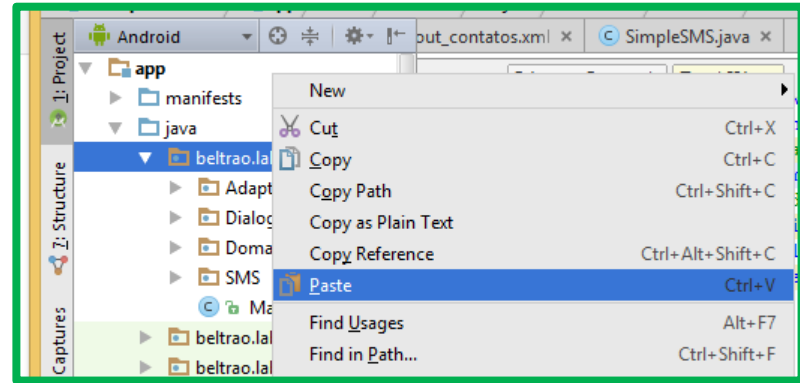
    </LinearLayout> </LinearLayout>
```

O próximo passo é copiar a classe SimpleSMS da pasta da área de trabalho e passar para nosso projeto, se achar necessário você pode criar um pacote.



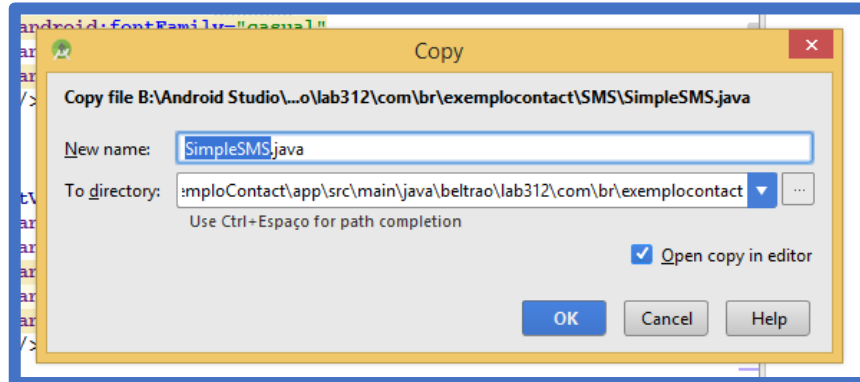
1

Copie a classe da pasta na área de trabalho



2

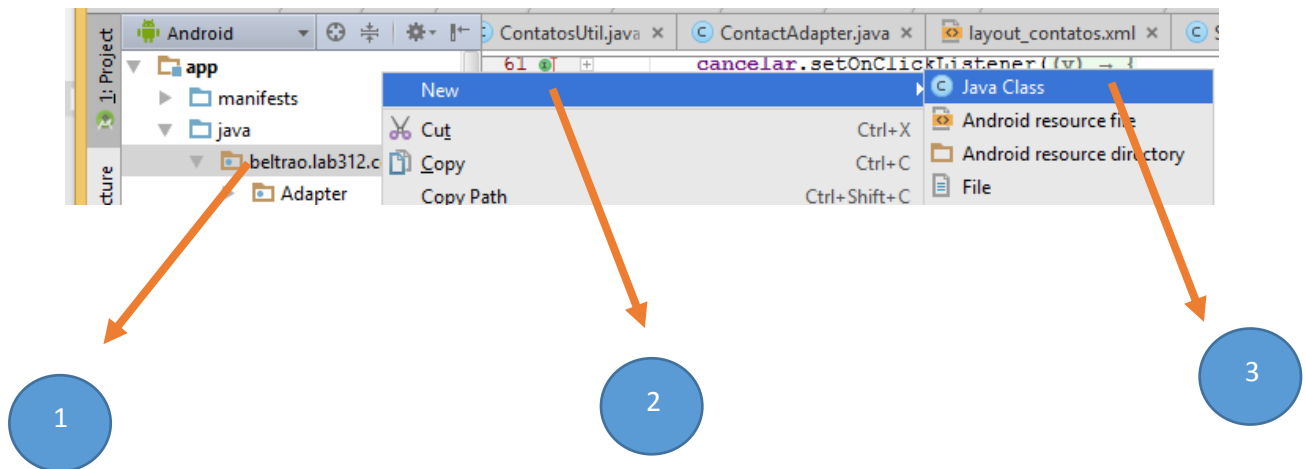
Cole dentro de seu projeto, dentro do pacote principal ou de sua preferencia



3

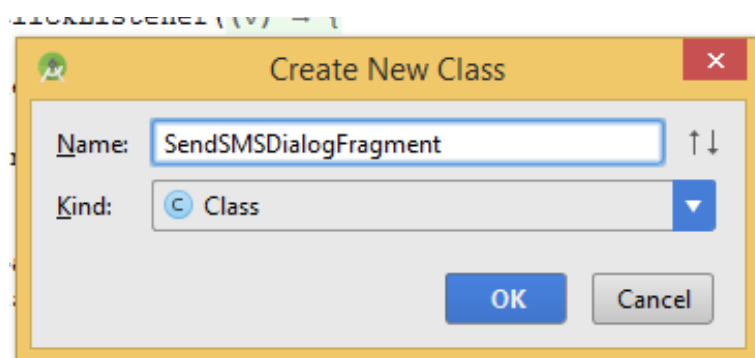
Apenas Clique em OK

Feito isso o próximo passo é criar a classe java para nossa interface.



1. Clique com o Botão direito do mouse no pacote principal
2. Clique em NEW
3. Clique em Java Class

Irá aparecer uma caixa de dialogo pedindo o nome da classe informe SendSMSDialogFragment e clique em OK.

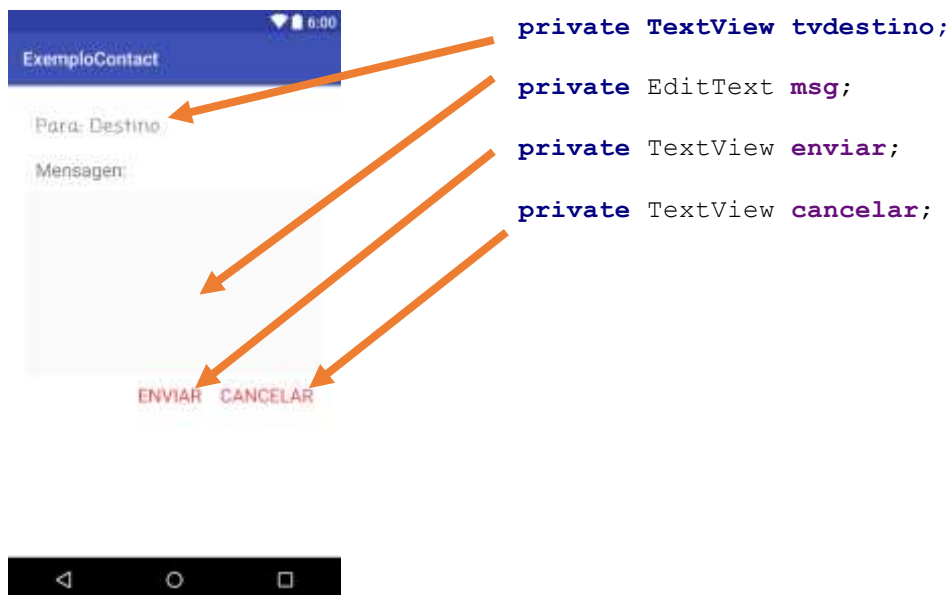


Dentro da classe SendSMSDialogfragment vamos criar algumas variáveis.

```
private Context com;  
private String destino;  
private String phone;
```

Nome do Destinatário

Numero de telefone do
destinatário



Próximo passo consiste em criar um método construtor para que sejam passados os atributos quando instanciarmos a classe.

```
public SendSMSDialogFragment( String phone, String destino, Context
com) {

    this.phone = phone;
    this.destino = destino;

    this.com = com;

}
```

A classe `SendSMSDialogFragment` é uma classe que consiste em tratar os eventos da interface, no entanto para que ela assuma o comportamento de `DialogFragment` ou seja para que ela apareça na tela em forma de pop-up é necessário estender a classe

chamada DialogFragment, para que então possamos sobrescrever o método onCreateView. Nossa classe até esta etapa deve ficar assim

```
@SuppressWarnings("ValidFragment")
public class SendSMSDialogFragment extends
    android.support.v4.app.DialogFragment{

    private TextView enviar;
    private TextView cancelar;
    private TextView tvdestino;

    private Context com;
    private String destino;
    private String phone;
    private EditText msg;

    public SendSMSDialogFragment( String phone, String destino, Context com)
    {

        this.phone = phone;
        this.destino = destino;

        this.com = com;

    }
}
```

Agora vamos sobrescrever o método onCreateView.

```
@Nullable
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {

    View v = inflater.inflate(R.layout.layout_send_sms, container);

}
}
```

O primeiro passo é inflar o Layout que criamos **layout_send_sms**, a partir de agora ele é referenciado pela variável v;

Agora através de `v` vamos referenciar os botões enviar e cancelar, o `textView` destino e o `EditText` que tem o conteúdo da mensagem informada pelo usuário.

```
enviar = (TextView) v.findViewById(R.id.enviar);
cancelar = (TextView) v.findViewById(R.id.cancelar);
tvdestino = (TextView) v.findViewById(R.id.destino);
msg = (EditText) v.findViewById(R.id.msg);
```

A seguir vamos:

1. definir que ao criar a tela o `textView` de Destino será o nome do contato mais o seu número de telefone.
2. Também vamos criar um evento de clique cancelar que irá fechar a tela com o método `dismiss()`.

```
tvdestino.setText(""+destino+" "+phone);
```

1

```
cancelar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        dismiss();
    }
});
```

2

Por fim, vamos criar o evento de clique de Enviar. A primeira coisa que fazemos é instanciar a classe `SimpleSMS` conforme mostra abaixo:

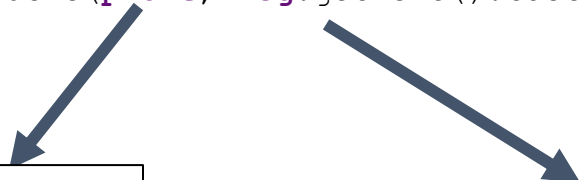
```
enviar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        SimpleSMS simpleSMS = new SimpleSMS(com);

    }
});
```

Em seguida usamos o metodo `send(numero, mensagem)` para enviar uma mensagem, este metodo retorna um valor booleano (true ou false) caso a mensagem seja enviada retorna true caso exista erro no número de telefone informado então retorna false.

```
simpleSMS.sendSMS(phone, msg.getText().toString());
```



Numero de telefone passado
na instancia da classe

Mensagem extraída do editText
da interface

O codigo de clique para enviar a mensagem fica assim:

```
enviar.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        SimpleSMS simpleSMS = new SimpleSMS(com);  
  
        boolean enviou = simpleSMS.sendSMS(phone,  
msg.getText().toString()+"\n MINICURSO-ANDROID-ICET");  
  
        if( enviou ){  
  
            Toast.makeText(getActivity(), "Enviando SMS ...",  
Toast.LENGTH_SHORT).show();  
            dismiss();  
        }  
  
        else{  
  
            Toast.makeText(getActivity(), "Falha ao enviar SMS ...",  
Toast.LENGTH_SHORT).show();  
  
        }  
  
    }  
});
```

O código completo da classe ficará assim:

```

@SuppressLint("ValidFragment")
public class SendSMSDialogFragment extends android.support.v4.app.DialogFragment{

    private TextView enviar;
    private TextView cancelar;
    private TextView tvdestino;

    private Context com;
    private String destino;
    private String phone;
    private EditText msg;

    public SendSMSDialogFragment( String phone, String destino, Context com) {

        this.phone = phone;
        this.destino = destino;
        this.com = com;
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {

        View v = inflater.inflate(R.layout.layout_send_sms, container);

        enviar = (TextView) v.findViewById(R.id.enviar);
        cancelar = (TextView) v.findViewById(R.id.cancelar);
        tvdestino = (TextView) v.findViewById(R.id.destino);
        msg = (EditText) v.findViewById(R.id.msg);

        tvdestino.setText(""+destino+" "+phone);

        cancelar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                dismiss();
            }
        });

        enviar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                SimpleSMS simpleSMS = new SimpleSMS(com);

                boolean enviou = simpleSMS.sendSMS(phone, msg.getText().toString()+"\n
MINICURSO-ANDROID-ICET");

                if( enviou ){

                    Toast.makeText(getActivity(), "Enviando SMS ...",
Toast.LENGTH_SHORT).show();
                    dismiss();
                }

                else{

                    Toast.makeText(getActivity(), "Falha ao enviar SMS ...",
Toast.LENGTH_SHORT).show();
                }
            }
        });
        return v;
    }
}

```

Para concluir o projeto so devemos mudar o comportamento de clique da lista para chamar nossa tela de enviar mensagem. Para isso abra o ContatosAdapter.java e edite o seguinte trecho de código:

```
holder.itemView.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        Toast.makeText(com, "nome "+nomes.get(position).getName(),  
        Toast.LENGTH_LONG).show();  
  
    }  
});
```

Mude para o seguinte código:

```
holder.itemView.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        FragmentTransaction ft =  
        mAppCompatActivity.getSupportFragmentManager().beginTransaction();  
        SendSMSDialogFragment sendSMSDialogFragment = new  
        SendSMSDialogFragment(nomes.get(position).getNumber(),  
        nomes.get(position).getNome(),  
        com);  
        sendSMSDialogFragment.show(ft, "Enviar SMS");  
  
    }  
});
```

A classe de ContatosAdapter completa ficará assim:

```
public class ContactAdapter extends RecyclerView.Adapter<ContactAdapter.holder> {

    private List<ContatosUtil> nomes;
    private Context com;
    private Typeface custom_font;
    private AppCompatActivity mAppCompatActivity;

    public ContactAdapter(List<ContatosUtil> nomes, Context com, AppCompatActivity
mAppCompatActivity) {

        this.nomes = nomes;
        this.com = com;
        this.mAppCompatActivity = mAppCompatActivity;
        custom_font = Typeface.createFromAsset(com.getAssets(), "quicksand.ttf");
    }

    @Override
    public holder onCreateViewHolder(ViewGroup parent, int viewType) {

        View v = LayoutInflater.from(com).inflate(R.layout.layout_contatos, parent,
false);
        holder mHolder = new holder(v);
        return mHolder;
    }

    @Override
    public void onBindViewHolder(holder holder, final int position) {
        int[] mMaterialColors = com.getResources().getIntArray(R.array.colors);
        Random RANDOM = new Random();

        holder.name.setText(nomes.get(position).getNome());
        holder.mMaterialLetterIcon.setLetterSize(30);
        holder.mMaterialLetterIcon.setLettersNumber(1);
        holder.mMaterialLetterIcon.setLetterTypeface(custom_font);
        holder.mMaterialLetterIcon.setLetterColor(Color.WHITE);

        holder.mMaterialLetterIcon.setShapeColor(mMaterialColors[RANDOM.nextInt(mMaterialCo
lors.length)]);
        holder.mMaterialLetterIcon.setLetter(nomes.get(position).getNome());

        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                FragmentTransaction ft =
mAppCompatActivity.getSupportFragmentManager().beginTransaction();
                SendSMSDialogFragment sendSMSDialogFragment = new
SendSMSDialogFragment(nomes.get(position).getNumber(),
                        nomes.get(position).getNome(),
                        com);
                sendSMSDialogFragment.show(ft, "Enviar SMS");

            }
        });
    }
}
```

```

@Override
public int getItemCount() {

    return nomes.size();
}

class holder extends RecyclerView.ViewHolder {
    public TextView name;
    public MaterialLetterIcon mMaterialLetterIcon;

    public holder(View itemView) {
        super(itemView);

        mMaterialLetterIcon = (MaterialLetterIcon)
itemView.findViewById(R.id.materialLetterCollorID);
        name = (TextView) itemView.findViewById(R.id.name);
    }
}
}

```

Feito isso está pronto nosso App:

Clique em Build, Rebuild

Espere o fim do processo.

Então Execute!!!

Contato:

Ruddá Beltrão: beltrao.rudah@gmail.com

Cristian Reis: christianreisoffice@gmail.com