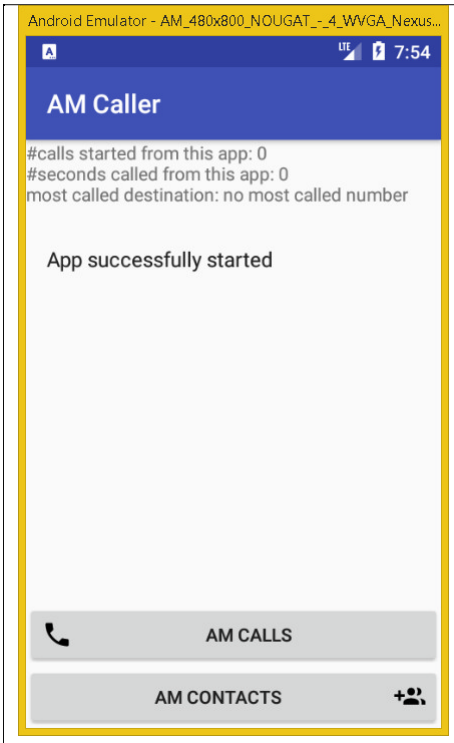

1. [Desenho de layout]
Escreva em XML o código de um RelativeLayout que permita a interface ilustrada pela imagem.

Existem, de cima para baixo, e da esquerda para a direita, com exceção da barra de título, as seguintes zonas:
Z0: uma zona de texto identificada por idTvDashboard, com mais do que uma linha, que em runtime receberá frase(s) com 3 linhas/informações;
Z1: uma ListView identificada por idLvOps, que em runtime poderá receber feedback de operações feitas com a app, a mais recente à cabeça;
Z2: um botão idBtnCalls, para encaminhar o utilizador para uma Activity "ActivityAmCalls";
Z3: um botão idBtnContacts", para encaminhar o utilizador para uma Activity "ActivityBasicContacts".

Note que esta descrição refere algumas situações de runtime, que o XML sozinho não consegue capturar - são referidas apenas para melhor contextualizar a app.

Saiba que os ícones que estão na imagem, nos botões, são Drawables vetoriais com ids "ic_calls" e "ic_contacts", que poderão ser embebidos nos Button via atributos "android:drawableLeft" e "android:drawableRight", para aparecerem à esquerda ou à direita, respetivamente.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.joythis.android.a20180114_simplecaller.SimpleCallerStart">

    <TextView
        android:layout_alignParentTop="true"
        android:singleLine="false"
        android:id="@+id/idTvDashboard"
        android:text="@string/strTvDashboard"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <ListView
        android:layout_below="@id/idTvDashboard"
        android:id="@+id/idLvOps"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:drawableLeft="@drawable/ic_calls_phone_black_24dp"
        android:layout_above="@id/idBtnContacts"
        android:layout_weight="1"
        android:id="@+id/idBtnCalls"
        android:text="@string/strBtnCalls"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:layout_alignParentBottom="true"
        android:drawableRight="@drawable/ic_contacts_group_add_black_24dp"
        android:layout_weight="1"
        android:id="@+id/idBtnContacts"
        android:text="@string/strBtnContacts"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

2. [padrão “init”]
Admita que o XML da questão anterior é o layout da Activity "SimpleCallerStart", que é a Activity de arranque da app.

Escreva, em Java, um método de nome "init", invocável em onCreate, que permita:
- inicializar os membros de dados Java da class, com nomes que correspondam aos ids do XML (por exemplo "mXpto" para o elemento que no XML tenha id "idXpto");
- confira, pelo mesmo listener, comportamentos diferentes "goCalls" e "goBasicContacts" aos botões em Z2 e Z3, respetivamente, devendo incluir na sua resposta a invocação das Activities adequadas;
- tenha código defensivo e reutilizável que evite quaisquer operações com objetos null;
- organize a abordagem em momentos de associação de variáveis a objetos XML e de atribuição de comportamentos;
- no final invoque "updateDashboard()", que deve admitir disponível e perfeito, tendo apenas que referir como escreveria String(s) com campos variáveis adequados em strings.xml.

```
public class SimpleCallerStart extends AppCompatActivity {
    Context mContext;
    TextView mTvDashboard;
    ListView mLvOps;
    Button mBtnCalls, mBtnContacts;
    View.OnClickListener mClickHandler;

    ArrayList<String> mAlStringOps;
    ArrayAdapter<String> mAdLvOps;

    Boolean mBCheckInitDataMembers, mBCheckAssignBehaviors;

    AmCallsStatistics mAmCallStatistics;

    void init(){
        this.setTitle(getString(R.string.strAppTitle));
        mBCheckInitDataMembers = initDataMembers();

        if (mBCheckInitDataMembers){
            mLvOps.setAdapter(mAdLvOps);
            newOp(getString(R.string.strInitDataMembers));

            mBCheckAssignBehaviors = assignBehaviors();
        }//if

        updateDashboard();
    }//init

    //_.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-
    Boolean initDataMembers(){
        mContext = this;
        mAlStringOps = new ArrayList<>();
        mAdLvOps = new ArrayAdapter<String>(
                this,
                android.R.layout.simple_list_item_1,
                mAlStringOps
        );

        mTvDashboard = (TextView)findViewById(R.id.idTvDashboard);
        mLvOps = (ListView) findViewById(R.id.idLvOps);
        mBtnCalls = (Button) findViewById(R.id.idBtnCalls);
        mBtnContacts = (Button) findViewById(R.id.idBtnContacts);

        Object[] aCheckThese =
        {
            mContext,
            mAlStringOps,
            mAdLvOps,

            mTvDashboard,
            mLvOps,
            mBtnCalls,
            mBtnContacts
        };

        Boolean bRet = checkIfAllNotNull (aCheckThese);

        return bRet;
    }//initDataMembers

    //_.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-.
    Boolean assignBehaviors(){
        Boolean bRet = false;

        mClickHandler = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int iClickedViewId = v.getId();
                // NÃO usar switch
```

```
                    // …
                     em certo caso goCalls();
                    no outro caso goContacts();
                }
            }//onClick
        };//mClickHandler

        Object[] aCheckThese = {mClickHandler};

        bRet = checkIfAllNotNull(aCheckThese);

        if (bRet){
            mBtnContacts.setOnClickListener(mClickHandler);
            mBtnCalls.setOnClickListener(mClickHandler);
        }//bRet

        return bRet;
    }//assignBehaviors

    //_.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-^-._.-
//Código defensivo
    Boolean checkIfAllNotNull (Object... pObjects){
        for (Object o : pObjects) if (o==null) return false;
        return true;
    }//checkIfAllNotNull

    void updateDashboard(){

// exemplo de string de 3 linhas, com campos variáveis a declarar em strings.xml
//<string name="strDashboard">#calls started from this app: %1$d\n#seconds called from this app: %2$d\nmost called destination: %3$s\n</string>

...         String strDash = String.format(
             getString(R.string.strDashboard),
             lHowManyCalls,
             lHowManySeconds,
             strMostCalledNumber
        );
        mTvDashboard.setText(strDash);
    }//updateDashboard

void goCalls(){
    Intent intentGoCalls = new Intent(this, ActivityAmCalls.class);
    startActivity(intentGoCalls);
}//goCalls

void goContacts(){
    Intent intentGoCalls = new Intent(this, ActivityBasicContacts.class);
    startActivity(intentGoCalls);
}//goContacts
```

3. [utilização de SQLite para Android]
Admita que quer categorizar contactos e chamadas, por exemplo com categorias como "family" e "work".

3.1. Escreva parcialmente as classes "Category" e "CategoryDB". Uma Category é apenas um "name" (String) e uma "dateStamp" (String); CategoryDB deve seguir o padrão SQLiteOpenHelper.

```
public class Category {
    private long mId;
    private String mName;
    private String mDateStamp;

          // constructor adequado a estes membros
          public Category(String pN, String pD){…}

          // o constructor escolhido é importante
          // porque terá que ser utilizado em 3.4, parar se criarem categorias
          // a chamada terá que ser compatível com a assinatura que aqui for respondida
…
}

class categoryDB extends SQLiteOpenHelper { …
// fazer onCreate com a constante CREATE_TABLE_CATEGORIES
// fazer onUpgrade com a constante DROP_TABLE_CATEGORIES
```

3.2. Escreva uma estrutura/tabela SQLite adequada para CategoryDB.

```
final static String CATEGORIES_DB_NAME = "CATEGORIES.DB";
final static int CATEGORIES_DB_VERSION = 1;

final static String TABLE_CATEGORIES = "categories";

final static String COL_ID ="_id";
final static String COL_NAME ="name";
final static String COL_DATE_STAMP ="date_stamp";

final static String CREATE_TABLE_CATEGORIES =
"CREATE TABLE IF NOT EXISTS "+
        TABLE_CATEGORIES +"("+
        COL_ID +" INTEGER PRIMARY KEY AUTOINCREMENT,"+
        COL_NAME +" TEXT,"+ //empty category accepted!
        COL_DATE_STAMP +" TEXT"+");";

final static String DROP_TABLE_CATEGORIES =
        "DROP TABLE IF EXISTS "+ TABLE_CATEGORIES +";";
```

3.3. Escreva
public long insertCategory (Category pC); //que deve retornar o id onde o insert foi feito, ou -1 se falhar.

```
public long insertCategory(
    Category pCategory
)
{
    long iIdForJustInsertedRecord=-1;

    try{
        SQLiteDatabase db=this.getWritableDatabase();

        ContentValues pairsColumnNameRecordValue = new ContentValues();
        pairsColumnNameRecordValue.put(COL_NAME, pCategory.getmName());
        pairsColumnNameRecordValue.put(COL_DATE_STAMP, pCategory.getmDateStamp());

        iIdForJustInsertedRecord = db.insert(
                TABLE_CATEGORIES,
                null,
                pairsColumnNameRecordValue
        );

        db.close();
    }
    catch (Exception e){
    }
    return iIdForJustInsertedRecord;
}//insertCategory
```

3.4. Escreva
ArrayList<Category> selectCategoriesWithNameMatching (String pExpression); //que deve retornar um ArrayList de Category objects cujo nome satisfaça exatamente a expressão recebida.

```
public ArrayList<Category> selectAllCategoriesWithNameMatching(
    String pExpression
)
{
    ArrayList<Category> ret = new ArrayList<>();
    try{
        SQLiteDatabase db=this.getWritableDatabase();

        String q="SELECT * FROM "+ TABLE_CATEGORIES +" where "+COL_NAME+" like
'%"+pExpression+"%'";
        Cursor allTheSelectResults = db.rawQuery(q, null);
        int iHowManyResults = allTheSelectResults.getCount();

        allTheSelectResults.moveToFirst();
        while(!allTheSelectResults.isAfterLast()){
            long idOfTheCurrentCategoryRecord =

allTheSelectResults.getLong(allTheSelectResults.getColumnIndex(COL_ID));
            String nameOfTheCurrentCategoryRecord =

allTheSelectResults.getString(allTheSelectResults.getColumnIndex(COL_NAME));
            String dateStampOfTheCurrentCategoryRecord =

allTheSelectResults.getString(allTheSelectResults.getColumnIndex(COL_DATE_STAMP));

            Category c = new Category(
                    nameOfTheCurrentCategoryRecord,
                    dateStampOfTheCurrentCategoryRecord
            );
            ret.add(c); //for an alternative return

            allTheSelectResults.moveToNext();
        }//while
        db.close();
    }//try
    catch(Exception e){

    }//catch
    return ret;
}//selectAllCategoriesWithNameMatching
```

4. [Execução assíncrona]
Assuma que é necessário fazer a importação de contactos "legacy", disponíveis para serem consumidos desde o URL: https://site/dados.TSV
Note que é um recurso remoto, a ser consumido por https; note também o formato TSV exemplificado:
John * 123456789 * family\n
Rita * 987654321 * work\n

Admita que tem disponível um método ler_https(String pUrl) que retorna a String do conteúdo lido por https.

4.1. Escreva, em SimpleCallerStart, em método “legacyLoader”, todo o código necessário para:
Ler os contactos e depois criar as categorias lidas, em base de dados, sem repetições;

TODO
Aspetos em apreciação:
- saber chamar ler_https,
- saber processar o retorno de ler_https (com split, por exemplo) para extração das categorias;
-- cuidado com split em * porque * é um símbolo com semântica própria em expressões regulares, por isso necessita de ser ESCaped “*”
- saber criar categorias com os dados lidos, usando métodos escritos nos grupo 3.

4.2. Escreva, na Activity principal, uma classe adequada a executar legacyLoader em thread própria.

```
class MinhaTarefaAssincronaModerna {
    private Executor executor = Executors.newSingleThreadExecutor(); // Can be static or application-wide
    private Handler handler = new Handler(Looper.getMainLooper()); // Used to post results back to the main thread

    public void execute(String url) {
        executor.execute(new Runnable() {
            @Override
            public void run() {
                // place for the background task
                String result =
                    legacyLoader();

                // Post your result back to the main thread
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        // This is equivalent to onPostExecute
                        mTvDashboard.setText(result);
                    }
                });
            }
        });
    }
}//MinhaTarefaAssincronaModerna
```

5. [Integração]
Admita que na Activity ActivityAmCalls está disponível um membro de dados mCalls, do tipo ArrayList<Call>, que em runtime descreve todas as chamadas já feitas a partir da aplicação.

Cada objeto do tipo Call, que pode assumir disponível e perfeito, representa:
- em membro mWhen, o momento de início de um telefonema;
- em membro String mDestination, o número de destino.

Com liberdade criativa, utilizando o padrão SAFR (Start Activity For Result), explique como faria para:
- chamar ActivityAmCalls a partir da Activity principal, deixando a "chamadora" atenta a um resultado, quando eventualmente chegar;
- na chamada deve seguir um "extra" inteiro em chave KEY_YEAR, correspondente a um ano (exemplo: 2023);
- ao terminar ActivityAmCalls, por algum mecanismo que decida, deve ser produzido como resultado o número de telefone de destino mais utilizado no ano indicado em KEY_YEAR, por análise de mCalls;
- ao receber o resultado (desde que não cancelado), a "chamadora" deve dar feedback, por algum mecanismo à sua escolha.

Na Activity principal, escreve-se um método de Callback que a framework chamará automaticamente, quando ActivityAmCalls tiver um resultado. No caso do velho padrão SAFR, poderá ser um override de onActivityResult;

Para que ActivityAmCalls tenha um resultado, tem que consultar, no Intent que a começa (e que se obtém por this.getIntent()) o valor do extra que está na chave KEY_YEAR, para saber o ano a considerar.
Conhecendo-se o ano, para calcular o destino (mDestination) mais telefonado, por pesquisa de mCalls, pode fazer assim:
- primeiro itera-se por mCalls e extrai-se (para uma lista auxiliar) todas as Call cujo mWhen tem o valor em KEY_YEAR;
- segundo, da lista auxiliar, forma-se o conjunto, sem repetições, dos valores de mDestination presentes;
- terceiro, para cada valor não repetido, conta-se quantas vezes ocorre na lista auxiliar;
- o valor maior corresponde ao número mais telefonado, no ano.

Se o resultado não for cancelado, ActivityAmCalls faz finish() com o resultado calculado.
A Activity principal, recebendo o resultado, escreve-o na dashboard baseada em TextView, ou apresenta um Toast.