VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELGAVI – 590018, KARNATAKA

KALPATARU INSTITUTE OF TECHNOLOGY,
TIPTUR-572201.

MOBILE APPLICATION DEVELOPMENT – 18CSMP68

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELGAVI – 590018, KARNATAKA

KALPATARU INSTITUTE OF TECHNOLOGY,

TIPTUR-572201.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,

LAB MANUAL PPT

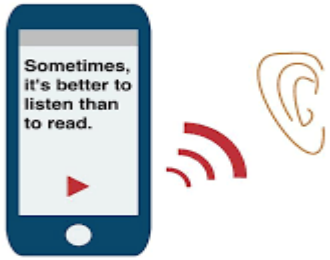
CHETHAN CHANDRA S BASAVARADDI,
ASSISTANT PROFESSOR,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
KALPATARU INSTITUTE OF TECHNOLOGY, TIPTUR-572201.
raddi04@yahoo.com

PREREQUISITE FOR THIS FDP SESSION

- Familiarity with the basics of Java Programming Language
- Basic concepts of XML

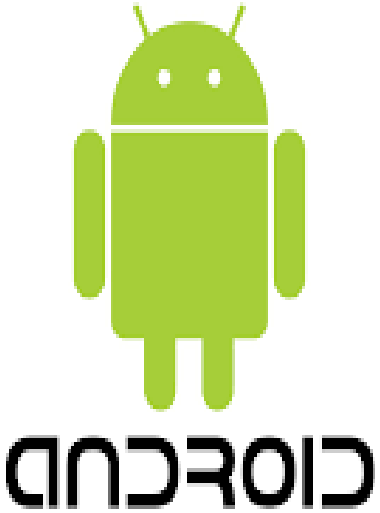
FLOW OF THIS FDP

- Basics of Android
- Parsing of JSON and XML files
- Counter Application
- Call Dialler Application
- Text to Speech Application
- MySQLite Database Application
- Content Provider Application



WHAT IS ANDROID?

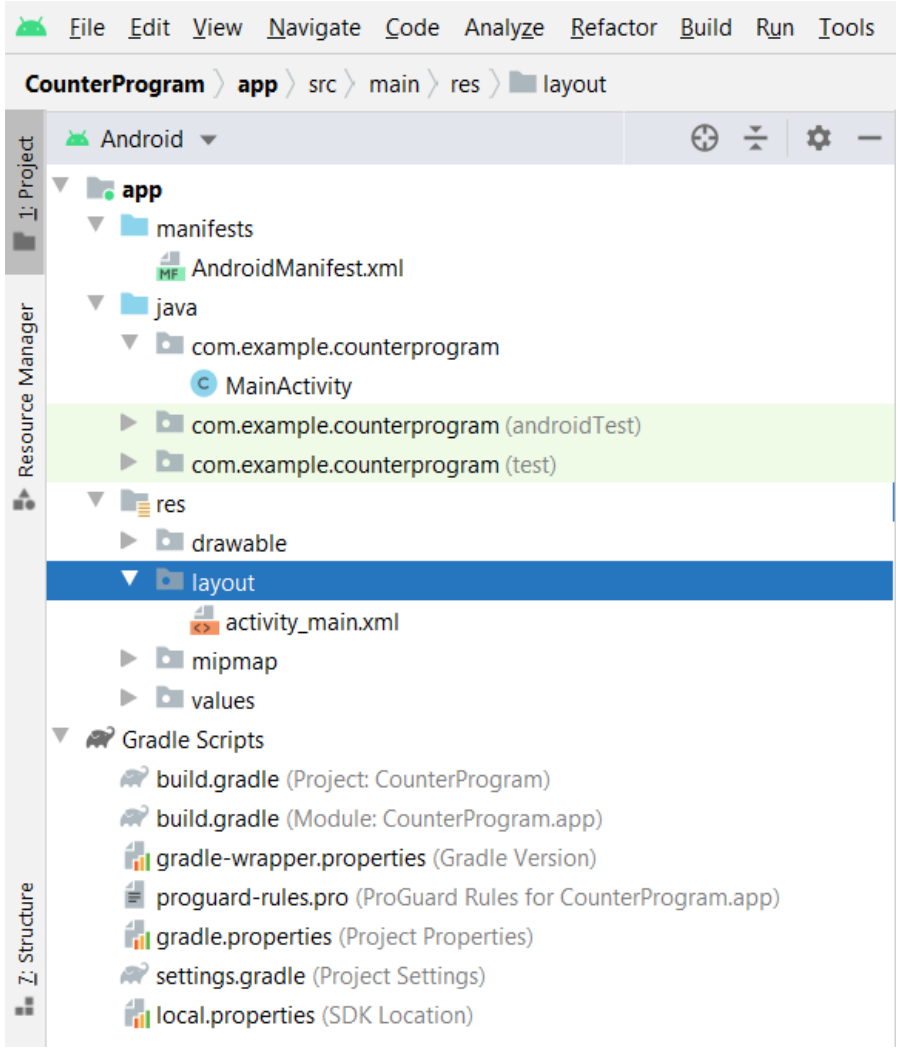
- Open software platform for Mobile Development
- A Complete Stack – OS, Middleware, Applications
- Powered by Linux
- Fast Application Development in Java
- Open Source under the Apache 2 License



ANDROID PROJECT STRUCTURE

Within each Android app module, files are shown in the following groups:

- manifests → contains the AndroidManifest.xml
- java → contains the java source code files, separated by package names, including Junit test code.
- res → contains all non – code resources, such as XML layout, UI strings and bitmap images, divided into corresponding sub-directories.



PROGRAM - 6

- Create two files of XML and JSON type with values for City_Name, Latitude, Longitude, Temperature and Humidity. Develop an application to create an activity with two buttons to parse the XML and JSON files which when clicked should display the data.

XML

- XML stands for Extensible Markup Language
- XML was designed to store, carry, and exchange data.
- XML was not designed to display data.
- XML makes the data more useful
- XML can be used to create new *ML .

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

- XML document has one and only one root element
- All XML Elements Must have Closing Tags

XML EXAMPLE - CITY.XML

```
<?xml version="1.0"?>
<records>
    <place>
        <name>Mysore</name>
        <lat> 12.295 </lat>
        <long>76.639 </long>
        <temperature> 22 </temperature>
        <humidity> 90 % </humidity>
    </place>
</records>
```

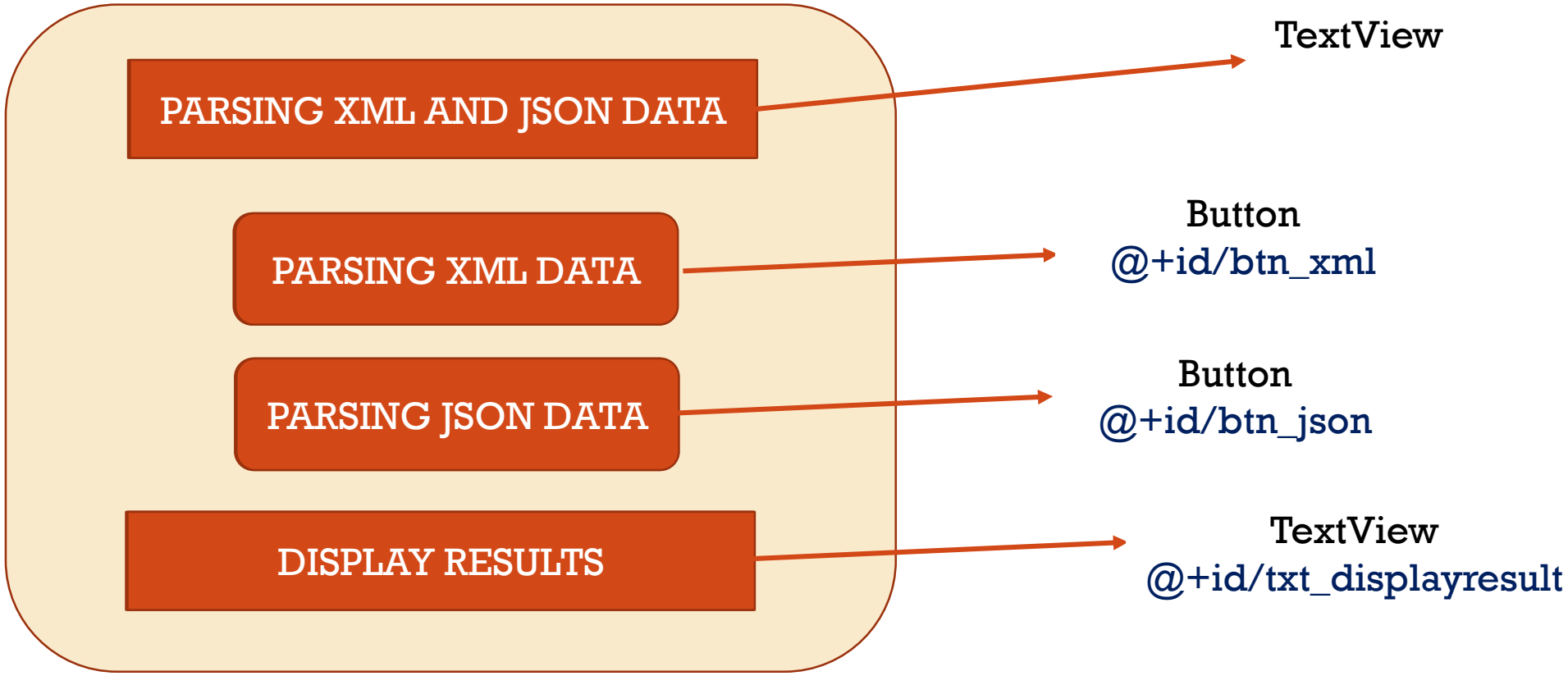
JSON – JAVASCRIPT OBJECT NOTATION

- JSON is a Data Representation Format
- Lightweight and Easy to Read / Write
- Integrates Easily with Most Languages
- JSON Types
  - Strings
  - Numbers
  - Booleans
  - Null
  - Arrays
  - Objects

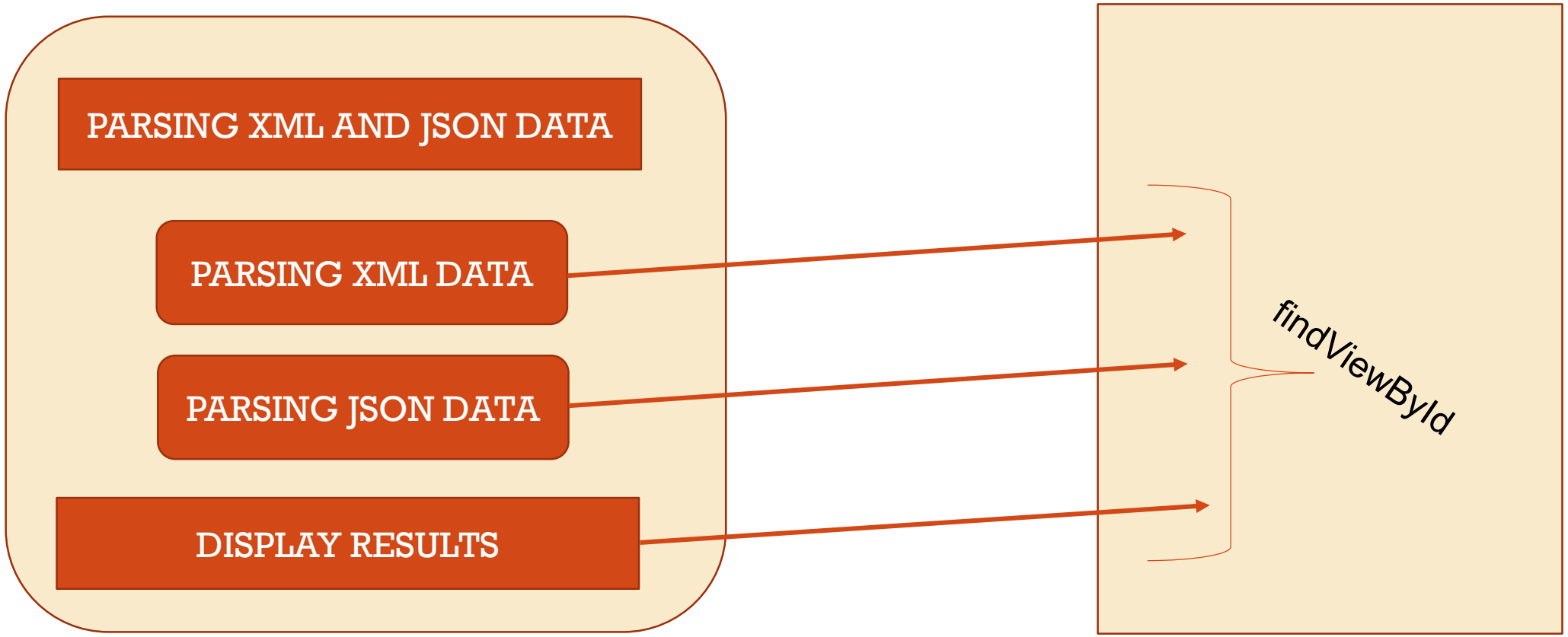
JSON EXAMPLE – SAMPLECITY.JSON

```
[
  { "name": "Gulbarga ",
    "lat": "12.295 ",
    "long": "76.639 ",
    "temperature":"38 ",
    "humidity": "92 %"
  }
]
```

PROGRAM – 7: GRAPHICAL LAYOUT OF THE APPLICATION



GETTING THE CONTROLS TO JAVA FILE



READ CONTENT FROM THE XML FILE

- InputStream – is useful when reading the data from a file, or received over a network and reads the data as an ordered sequence of bytes.
- getAssets() – returns an Assets Manager for the application's package
- getAssets().open(filename) → opens a file using the Assets Manager
- DocumnetBuilderFactory.newInstance() → creates a new instance of DocumentBuilderFactory class
  - DocumentBuilderFactory enables applications to obtain a parser that produces object trees from XML documents.
- DocumentBuilder() → used to obtain DOM Document Instances from an XML Document.
- Parse() → parse the content of the given InputStream as an XML and returns a new DOM Document object

READ CONTENT FROM THE XML FILE

- getDocumentElement() → this method searches an XML document for an root element and returns XML object containing the document element and all of its child elements.
- normalize() → puts all text nodes underneath a node into a normal form.
- getElementsByTagName() → method returns a collection of an elements's child elements with the specified tag name, as a NodeList object.
- The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

READ CONTENT FROM JSON FILE

- Use InputStream and Assets Manager to open the JSON file
- InputStream.available() → returns the number of remaining bytes that can be read from this input stream without blocking by the next method call for this input stream
- Byte[] buffer = new byte[n]; → creates a byte array of size 'n'
- InputStream.read(buffer) → this method will attempt to read as many bytes into the byte array given as parameter as the array has space for.
- json = new String(buffer, "UTF-8"); → constructs a new string object named json by decoding the specified array of bytes using the specified charset
- JSONArray jsonArray = new JSONArray(json) → creates a new JSONArray with the values from the "json" String

READ CONTENT FROM JSON FILE

- JSONObject obj = jsonArray.getJSONObject(i); → returns the value at index if it exists and is a JSONObject
- obj.getString(String) → returns the value mapped by name if it exists, coercing it if necessary, or throws if no such mapping exists.

Creating Counter Application

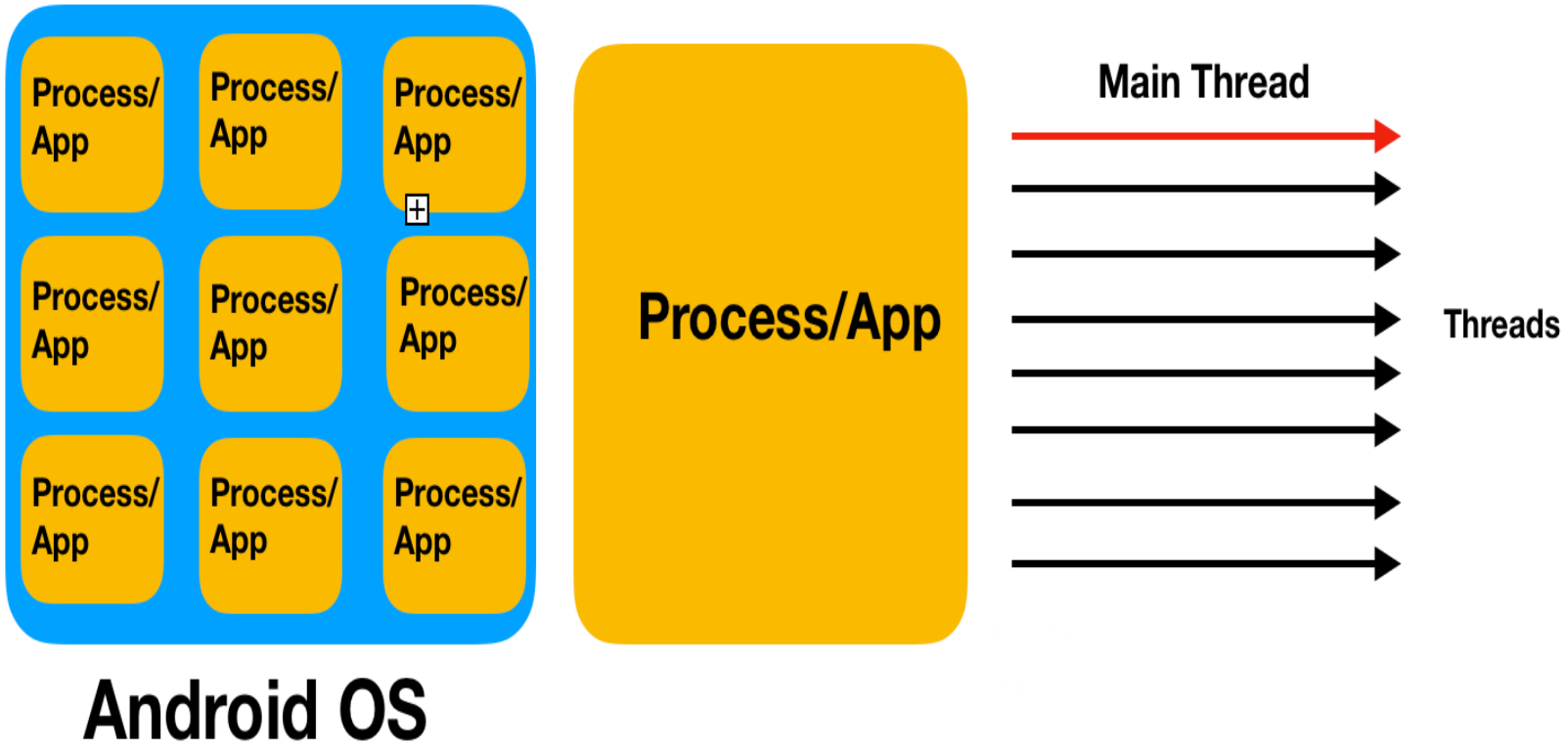
ABOUT ME

- I am Lavanya Santhosh., working as Assistant Professor in the Department of Computer Science and Engineering at Dr. Ambedkar Institute of Technology, Bengaluru.
- Research Interests include Machine Learning, Deep Learning and Big Data Analytics
- You can contact me at:

lavanyasanthosh.cs@drait.edu.in

www.linkedin.com/in/lavanya-basavaraj

THREADS


Process/ App
Process/ App
Process/ App
Process/ App
Process/ App
Process/ App
Process/ App
Process/ App
Process/ App
Android OS
Process/App
Main Thread
Threads

IMPORTANT RULES WHEN WORKING WITH THREADS

- Don’t block the Main thread
- Don’t try and access the UI directly from the worker thread

CREATING NEW THREAD

I. Extending the Thread class

- The class should extend the Thread class.
- The class should override the run() method.
- The functionality that is expected by the Thread to be executed is written in the run() method.
- To start the thread, create a new Thread object and then call start().

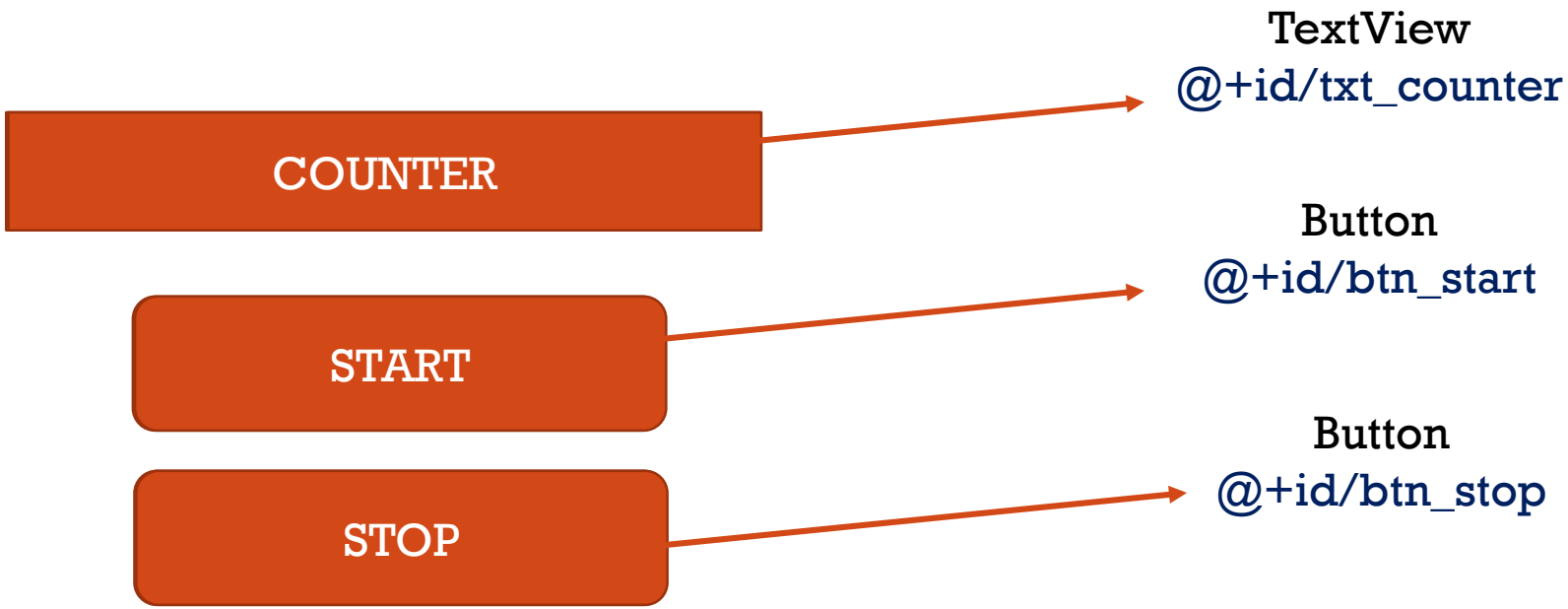
- **void start():** Creates a new thread and makes it runnable.
- **void run():** The new thread begins its life inside this method.

CREATING NEW THREAD

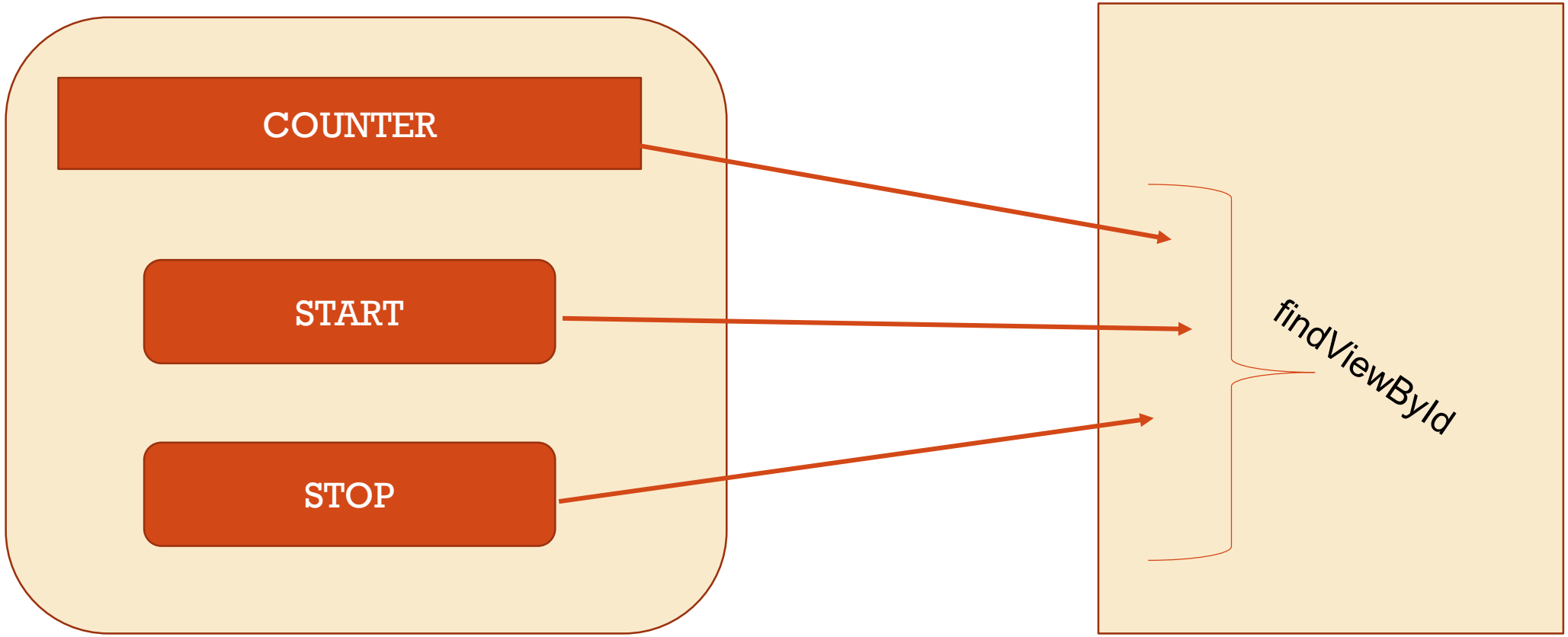
II. Implementing the Runnable Interface

- The class should implement the Runnable interface
- The class should override the **run()** method.
- The functionality that is expected by the Thread to be executed is put in the run() method
- To start the thread, pass the runnable to a new Thread object and then call **start()**.

PROGRAM – 5: GRAPHICAL LAYOUT OF THE COUNTER APPLICATION



GETTING THE CONTROLS TO JAVA FILE



XML FILE

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text="0"
    android:gravity="center"
    android:textSize="120dp"
    android:id="@+id/txt_counter“/>

<Button
    android:id="@+id/btn_start"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Start"/>

<Button
    android:id="@+id/btn_stop"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Stop"/>
```

USING AN ONCLICKLISTENER

Two ways

- Implementing View.OnClickListener in your Activity or fragment.
  - link the button in xml and give the reference (findViewById, R.id.btn_start)
  - mButton.setOnClickListener(this);

- Creating new anonymous View.OnClickListener.
  - Link the button from the XML by calling findViewById() method and set the onClick listener by using setOnClickListener() method.
  - setOnClickListener takes an OnClickListener object as the parameter.

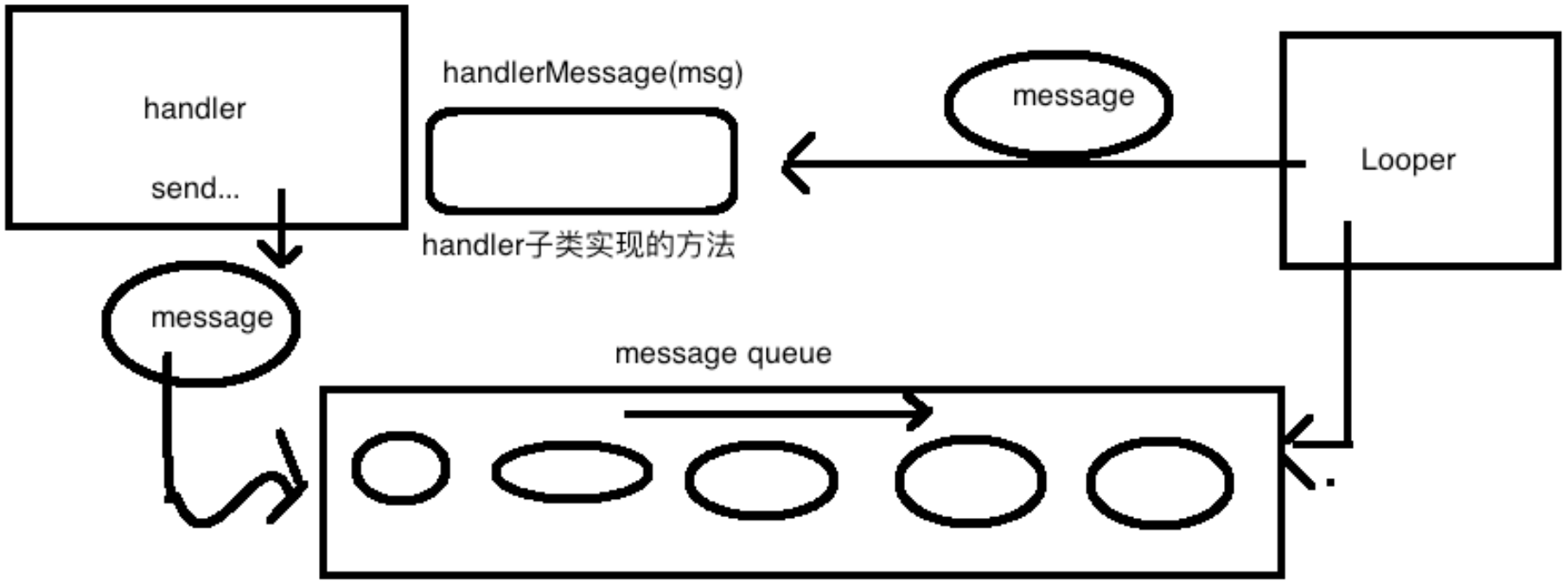
@OVERRIDE ONCLICK()

```
stop.setOnClickListener(new View.OnClickListener(){

    @Override
    public void onClick(View v) {
        running = false;
    }
});

start.setOnClickListener(new View.OnClickListener(){

    @Override
    public void onClick(View v) {
        running = true;
        mythread = new Thread(runnable);
        mythread.start();
    }
```

HANDLER MECHANISM

- **Handler**
  - **Handler.sendMessage()**
  - **Handler.handleMessage()**
- **Message**: Message entity, the message sent is of type Message.
- **MessageQueue**: Message queue, used to store messages. When sending a message, the message enters the queue, and then Looper will take the message from this MessageQueue for processing.
- **Looper**: The loop() method is an infinite loop, which always takes messages from MessageQueue for processing.

SCHEMATIC DIAGRAM



HANDLER

```
Handler h = new Handler(){
    @Override
    public void handleMessage(@NonNull Message msg) {
        super.handleMessage(msg);
        counter.setText("" + msg.what);
    }
};
```

RUNNABLE INTERFACE

- Implement runnable interface

```
Runnable runnable = new Runnable()
```

- @override run();

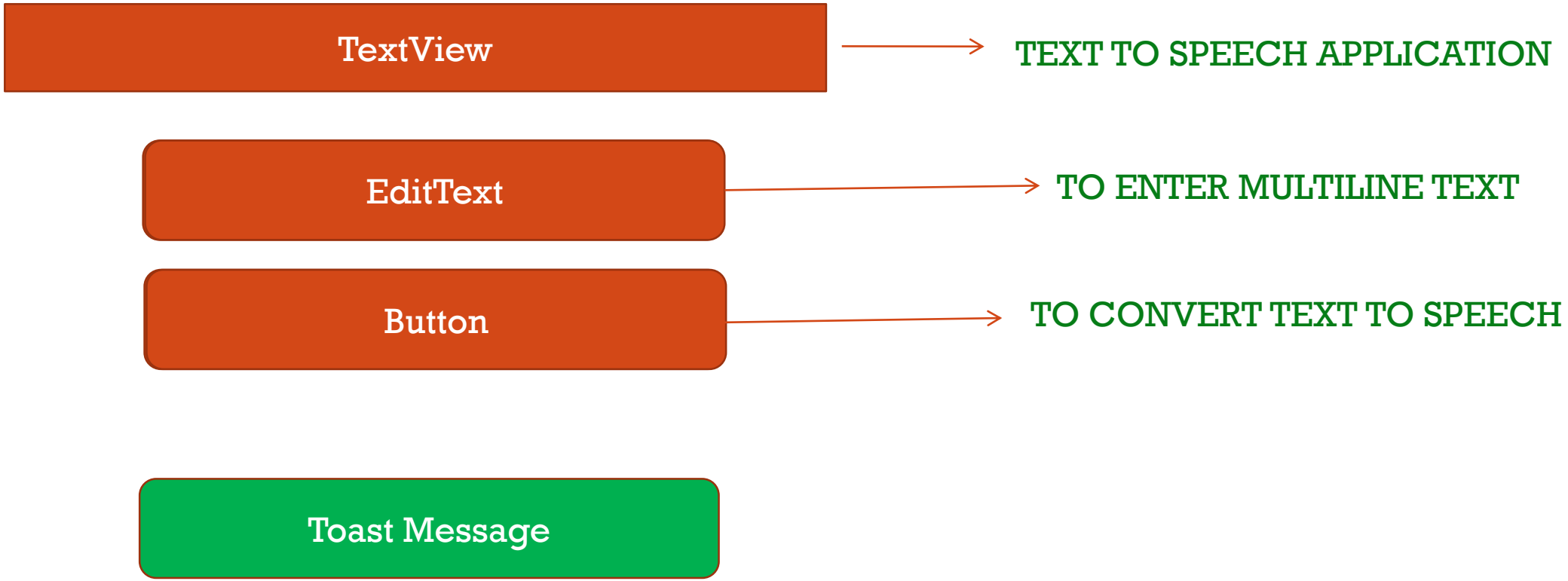
```
@Override
public void run()
      {
   try{ int i=0;

      while (running){
         i += 1;
         mythread.sleep(1000);
         h.sendEmptyMessage(i);
      }
   }catch(Exception e){
      e.printStackTrace();
   }

}
```

Creating Text-to-Speech Application

XML FILE

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="TEXT TO SPEECH APPLICATION"
   android:textSize="18sp"/>

<EditText
   android:id="@+id/txt_input"
   android:layout_width="237dp"
   android:layout_height="177dp"
   android:layout_marginStart="100dp"
   android:layout_marginTop="209dp"
   android:layout_marginEnd="77dp"
   android:layout_marginBottom="437dp"
   android:inputType="textMultiLine"" />

<Button
   android:id="@+id/btn_txt2spch"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_marginStart="75dp"
   android:layout_marginTop="399dp"
   android:layout_marginEnd="51dp"
   android:layout_marginBottom="284dp"
   android:text="Convert Text to Speech"/>
```

TEXT-TO-SPEECH APP



STEPS TO CREATE TEXT-TO-SPEECH APP

- **TextToSpeech** class
- Instantiate an object and specify the **initListener.**
- Syntax

```
private EditText write;
t1=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
@Override public void onInit(int status) { } });
```

- Specify the properties for TextToSpeech object , such as its language.

```
ttobj.setLanguage(Locale.UK);
```

- Call **speak** method of the class to speak the text.

```
ttobj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
```

ONCLICK()

- **Override onClick()**

```
@Override
  public void onClick(View view) {
  String tospeak= txtinput.getText().toString();
  Toast.makeText(getBaseContext(),tospeak,Toast.LENGTH_SHORT).show();

  t1.speak(tospeak,TextToSpeech.QUEUE_FLUSH,null);
          }
      });
   }
```

ONPAUSE()

- **Call onPause()**

```
public void onPause()
    {
  if(t1 != null)
        {
  t1.stop();
  t1.shutdown();
        }
  super.onPause();
    }
```

Creating Call-Dialer Application

INTENTS

- An intent is a messaging object you can use to request an action from another app component.

- Intent facilitates communication between different components

- Intents are of two types
  - **Implicit Intent**
  - **Explicit Intent**

THREE USE CASES

- Starting an activity
- Starting a service
- Delivering a Broadcast receiver.

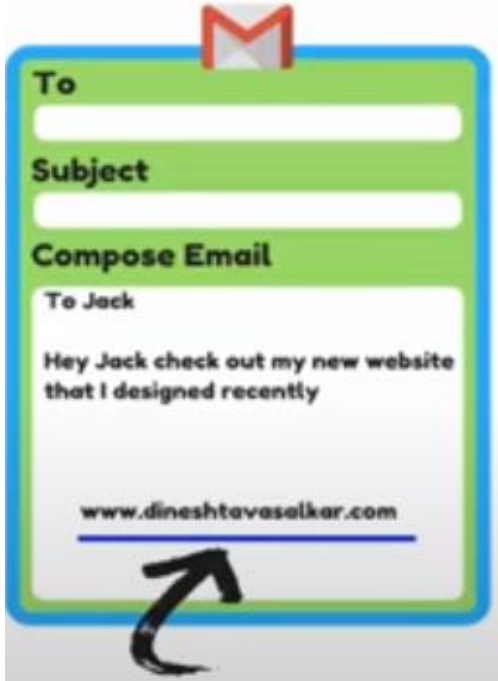
IMPLICIT INTENT

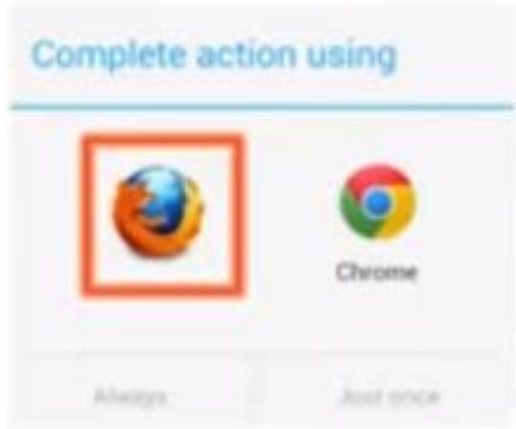
- Implicit intents do not declare the class name of the component to start, but instead declare an action to perform. The action specifies the thing you want to do, such as view, edit, send. Intents often also include data associated with the action, such as the address you want to view, or the email message you want to send.

- **Communicates between 2 activities inside of different application.**

Ex: Intent = new Intent (Intent.ACTION_VIEW, Uri. parse ("http://www.google.com"));

EXAMPLES OF IMPLICIT INTENT





EXPLICIT INTENT

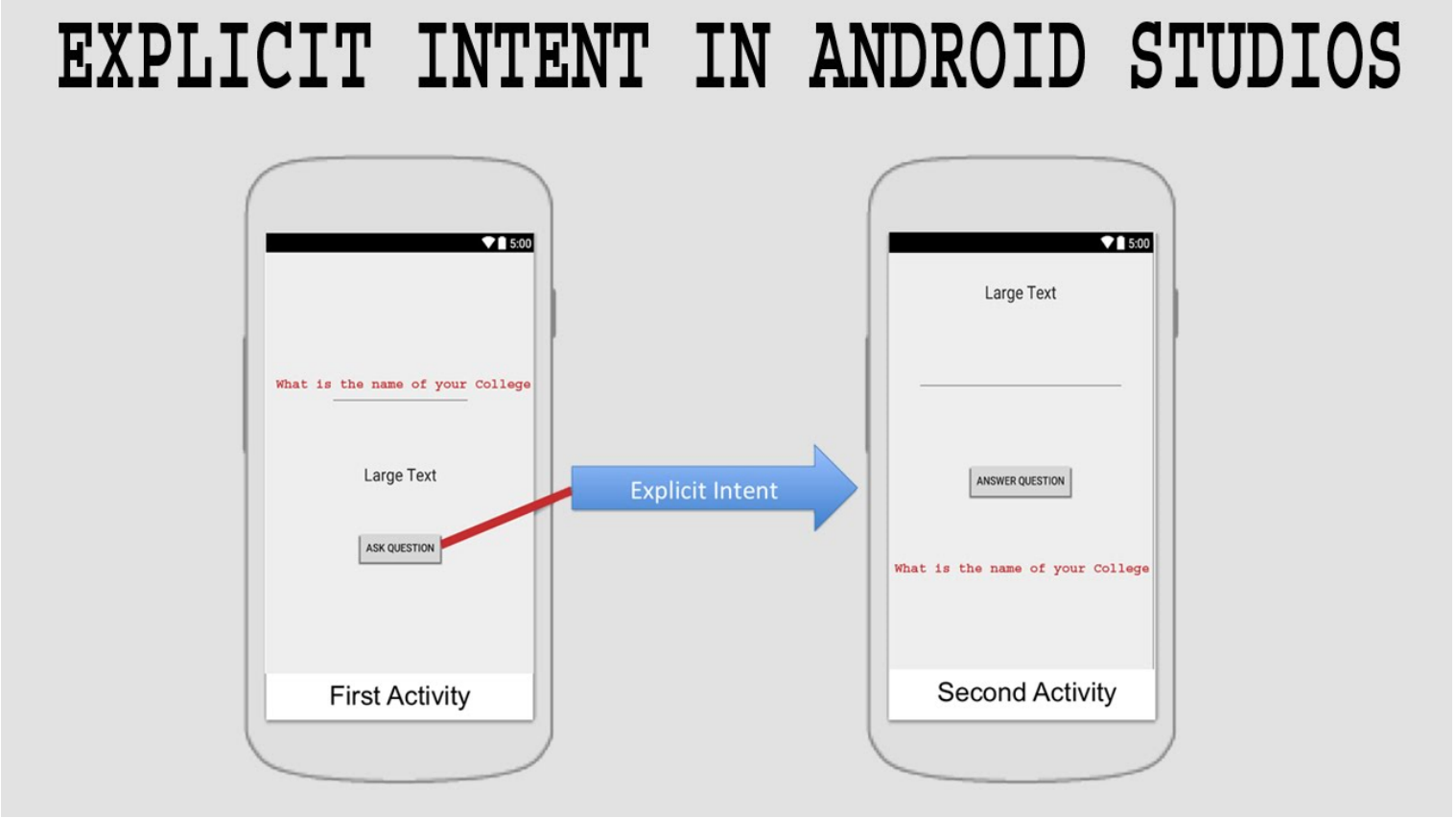
- **We specify which activity should get active on receiving the intent. These are usually used for application's internal communications. They will specify a component which provides the exact class to be run.**

- **Communicates between 2 activities inside the same application.**

Ex: **Intent it = new Intent (this, newactivity.class);**

startActivity(it);

EXAMPLES OF EXPLICIT INTENT



XML FILE

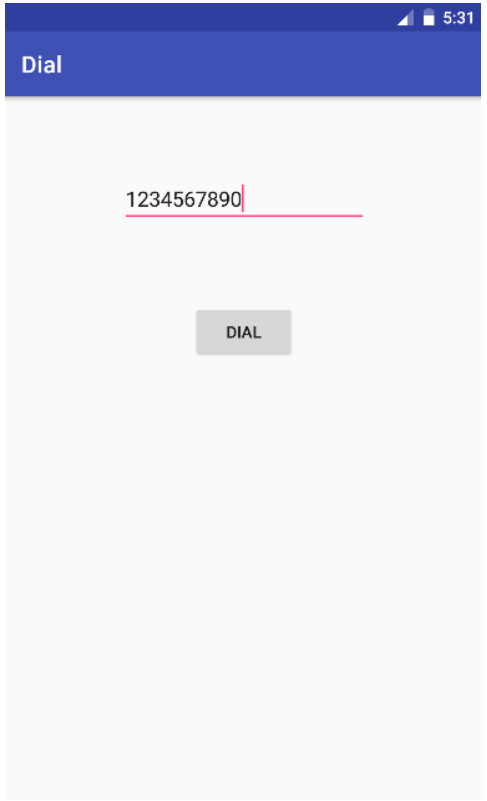
```
<LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:orientation="horizontal" >

      <EditText
          android:id="@+id/num"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:hint="Enter number"
          />
      <Button
          android:id="@+id/del"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Del"
          />
  </LinearLayout>
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/one"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1"
        />
    <Button 2/>
    <Button 3/>
</LinearLayout>
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/one"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="#"
        />
    <Button CALL/>
    <Button SAVE/>
</LinearLayout>
```

TO APPEND THE NUMBER IN THE DIALER

```
final EditTextnum= (EditText) findViewById(R.id.num);

  findViewById(R.id.one).setOnClickListener(new
  View.OnClickListener() {
  @Override
  public void onClick(View arg0) {
  num.append("1");
```



DELETING THE NUMBER IN THE DIALER

```
1. findViewById(R.id.del).setOnClickListener(new View.OnClickListener() {


  @Override
  public void onClick(View arg0) {
  String number = num.getText().toString();
              number = number.substring(0, number.length()-1);
  num.setText(number);
          }
      });
```

PUTEXTRA()

- **putExtra()** adds extended data to the intent.

- It has two parameters, first one specifies the name of the extra data and the second parameter is the data itself.

IMPLICIT INTENT FOR CALL

- Intent it = new Intent(Intent.*ACTION_CALL*);
  it.setData(Uri.*parse*("tel:"+number));
  startActivity(it);

- ContactsContract.Data : If you need to look up a contact by some data element like email address, nickname, etc, use a query against the ContactsContract.Data table. The result will contain contact ID, name etc.
- *CONTENT_URI*
- Intent it = new Intent(Intent.*ACTION_INSERT*,
  ContactsContract.Contacts.*CONTENT_URI*);
  it.putExtra(ContactsContract.Intents.Insert.*PHONE*, number);

ANDROID MANIFEST FILE

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

CREATING DATABASE APPLICATION

ABOUT ME

- I am Uma K.M., working as Assistant Professor in the Department of Computer Science and Engineering at Dr. Ambedkar Institute of Technology, Bengaluru.
- Research Interests include Cloud Computing
- You can contact me at: uma.km@dr-ait.org

SQLITE

- **Android** comes in with built In **SQLite** database implementation. So, there is no need to perform any **database** setup or administration task.

- **SQLite** is an open-source relational database. it is used to perform database operations on Android devices such as storing, manipulating or retrieving persistent data from the database.

- It is
  - Open source
  - Lightweight
  - Single-tier

SQLITEOPENHELPER CLASS

- You will use SqLiteOpenHelper to create and upgrade your SQLite database.
- SqLiteOpenHelper removes the effort required to install and configure database in other systems.
- The database is not actually created or opened until one of getWritableDatabase() or getReadableDatabase() is called.

STEPS TO CREATE DATABASE APPLICATION IN ANDROID

- Create a class that extends SQLiteOpenHelper
- Override **onCreate()** and **onUpgrade()** method of SQLiteOpenHelper.
- Create a constructor that will make superclass constructor call.

CREATE A CLASS THAT EXTENDS SQLITEOPENHELPER

```
//Initial Class
public class MyDatabase
{
   ---
   ---
   ---
}

//Extend with SQLiteOpenHelper
public class MyDatabase extends SQLiteOpenHelper
{
   ---
   ---
   ---
}
```

OVERRIDE ONCREATE() AND ONUPGRADE() METHOD OF SQLITEOPENHELPER

onCreate()

- When we call constructor of an SQLiteOpenHelper class you must pass Database name and version. SQLiteOpenHelper will check whether in this class in database with that name and version exists or not. If does not exists, SQLiteOpenHelper will call **onCreate()** method.

onUpgrade()

- If database with name and version exists it will call **onUpgrade()** method.

CREATE A CONSTRUCTOR THAT WILL MAKE SUPERCLASS CONSTRUCTOR CALL

```
public MyDatabase( Context context, String name, CursorFactory
  factory, int version)
 {
 super(context, name, factory, version);
 // TODO Auto-generated constructor stub
 }
```

1) Context: It is the **context** of the current state of the application. It can be used to get information regarding the activity and application.
2) Name: specifies database name
3) Factory: used to allow returning sub-classes of **Cursor** when calling query.
4) Version: specifies version

STEPS TO INSERT DATA INTO THE DATABASE

1. Create object of your Database class. In that pass the context db name and version using constructor.

2. Open Database in writable mode using **getWritableDatabase()** object.

3. Create object of type **ContentValues** and add key/value to the columns in the table.

4. Insert data to Database using **insert()** method of SQLiteDataBase .

STEP1: TO INSERT DATA INTO THE DATABASE

MyDatabase **d**= new MyDatabase(this, MyDatabase.*DATABASE_NAME, null, 1);*

d:

1) ***object of*** MyDatabase class.
2) Used to open the database in either getReadableDatabase() or getWritableDatabase() mode

STEP2: TO INSERT DATA INTO THE DATABASE

SQLiteDatabase db= d. getWritableDatabase();

db:

1) specifies object of SQLiteDatabase.

2) Used to insert data into the database & close database.

STEP3: TO INSERT DATA INTO THE DATABASE

ContentValues

- ContentValues are used to insert new rows into tables. Each ContentValues object represents a single table row as a map of column names to values

- This class is used to store a set of values that the ContentResolver can process.

ContentValues cv= **new ContentValues();**

Methods of ContentValues:

1) Put(String, String)--Adds a value to the set.

Parameters

- Key:String--the name of the value to put
- Value:String--the data for the value to put

STEP4: TO INSERT DATA INTO THE DATABASE

- **insert – It is a method for inserting a row into the database.**

insert(String table, String nullColumnHack,
ContentValues values)

Ex→ db.insert("Employee", **null, cv);**
db.close();

1) **First parameter→ Table name**
2) **Specifies the column that will be set to null if the ContentValues argument contains no data.**
3) **Contains the data that will be inserted into the table.**

CODE TO INSERT VALUES INTO THE TABLE

- MyDatabase d = new MyDatabase(this, MyDatabase.*DATABASE_NAME, null, 1);*

- SQLiteDatabase db= d.getWritableDatabase();

- ContentValues cv= **new ContentValues();**

```
cv.put("id", sid);
cv.put("name", sname);
cv.put("age",sage );
cv.put("addr", saddr);
db.insert("Employee", null, cv);
```

STEPS TO QUERY DATA FROM DB

1) Create object of your DataBase class.

2) Open DataBase in readable mode using getReadableDatabase() method of your DataBase object.

3) Use query method of SQLiteDataBase object and pass table name , columns to fetch "where" condition and values for "where" condition.

4) The **query** method will return the object of type **Cursor.** Use **move-to-next** method of cursor to go to each row to get the value using getstring, getint, getfloat etc.

STEP1: TO QUERY DATA FROM DB

- MyDatabase d = new MyDatabase(this, MyDatabase.*DATABASE_NAME, null, 1);*

d:

1) ***object of*** MyDatabase class.
2) Used to open the database in either getReadableDatabase() or getWritableDatabase() mode

STEP2: TO QUERY DATA FROM DB

SQLiteDatabase db= d. getWritableDatabase();

db:

1) specifies object of SQLiteDatabase.

2) This object is used with query() method to query data from the database.

STEP3: TO QUERY DATA FROM DB

Query():

- Retrieve data from your database. Use the arguments to select the table to query, the rows and columns to return, and the sort order of the result.

- Return the data as a Cursor object.

```
public Cursor query (String  table,
                     String[] columns,
                     String selection,
                     String[] selectionArgs,
                     String sortOrder)
```

STEP4: TO QUERY DATA FROM DB

Cursor: provides random read-write access to the result set returned by a database query.

1) moveToFirst(): Move the cursor to the first row.

2) moveToLast(): Move the cursor to the last row.

3) moveToNext(): Move the cursor to the next row.

4) moveToPrevious(): Move the cursor to the previous row.

5) moveToPosition(int position): Move the cursor to an absolute position.

6) getString(int columnIndex): Returns the value of the requested column as a String.

Creating Content Provider

CONTENT PROVIDER

- A **content provider** manages access to a central repository of data. A **provider** is part of an **Android** application, which often provides its own UI for working with the data.

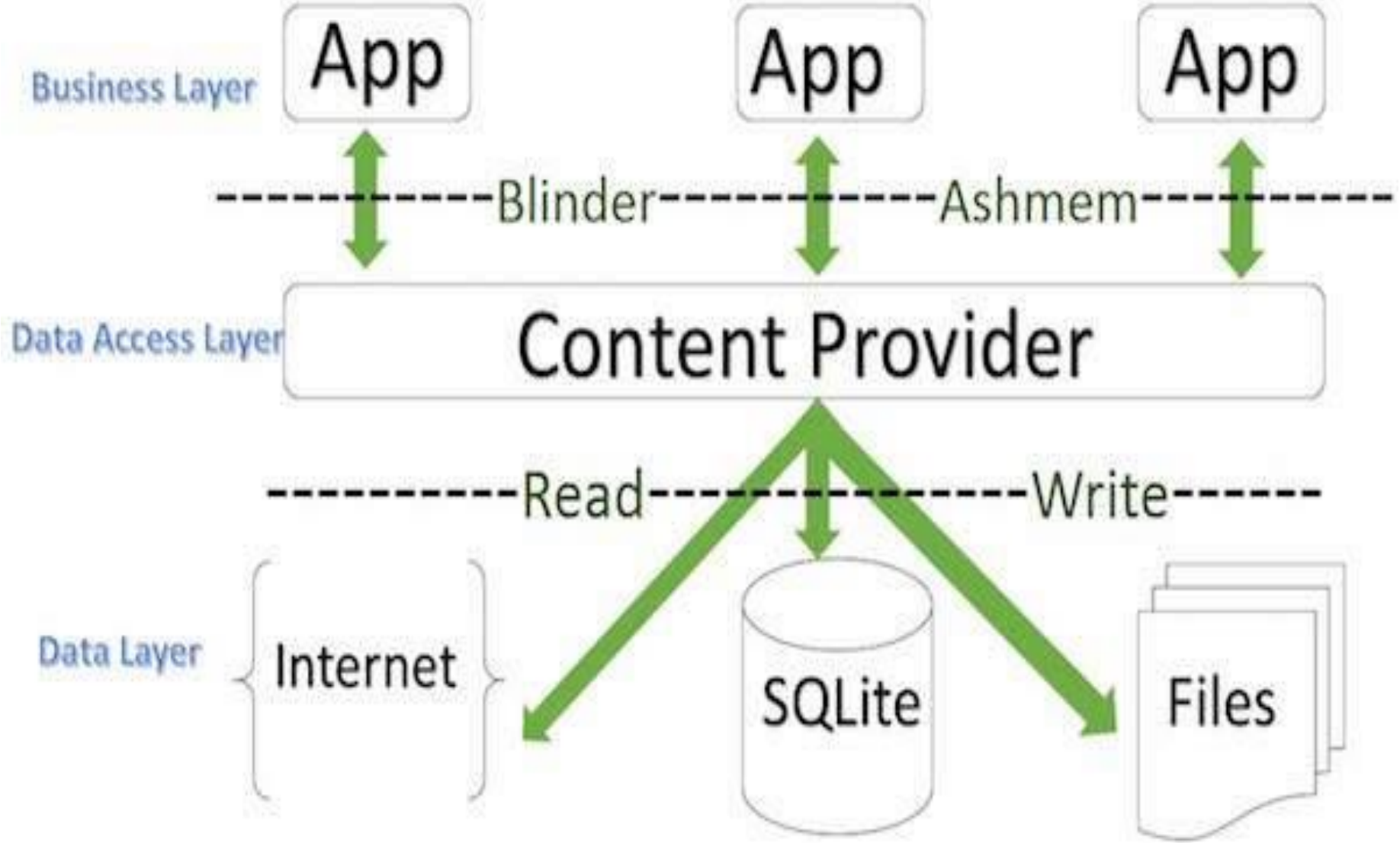
- Android system allows the content provider to store the application data in several ways. Users can manage to store the application data like images, audio, videos, and personal contact information by storing them in **SQLite Database**, **in files**, **or even on a network**.

WHY??

- Sometimes it is required to share data across applications. This is where content providers become very useful.

- In order to share the data, content providers have certain permissions that are used to grant or restrict the rights to other applications to interfere with the data.

CONTENT PROVIDER

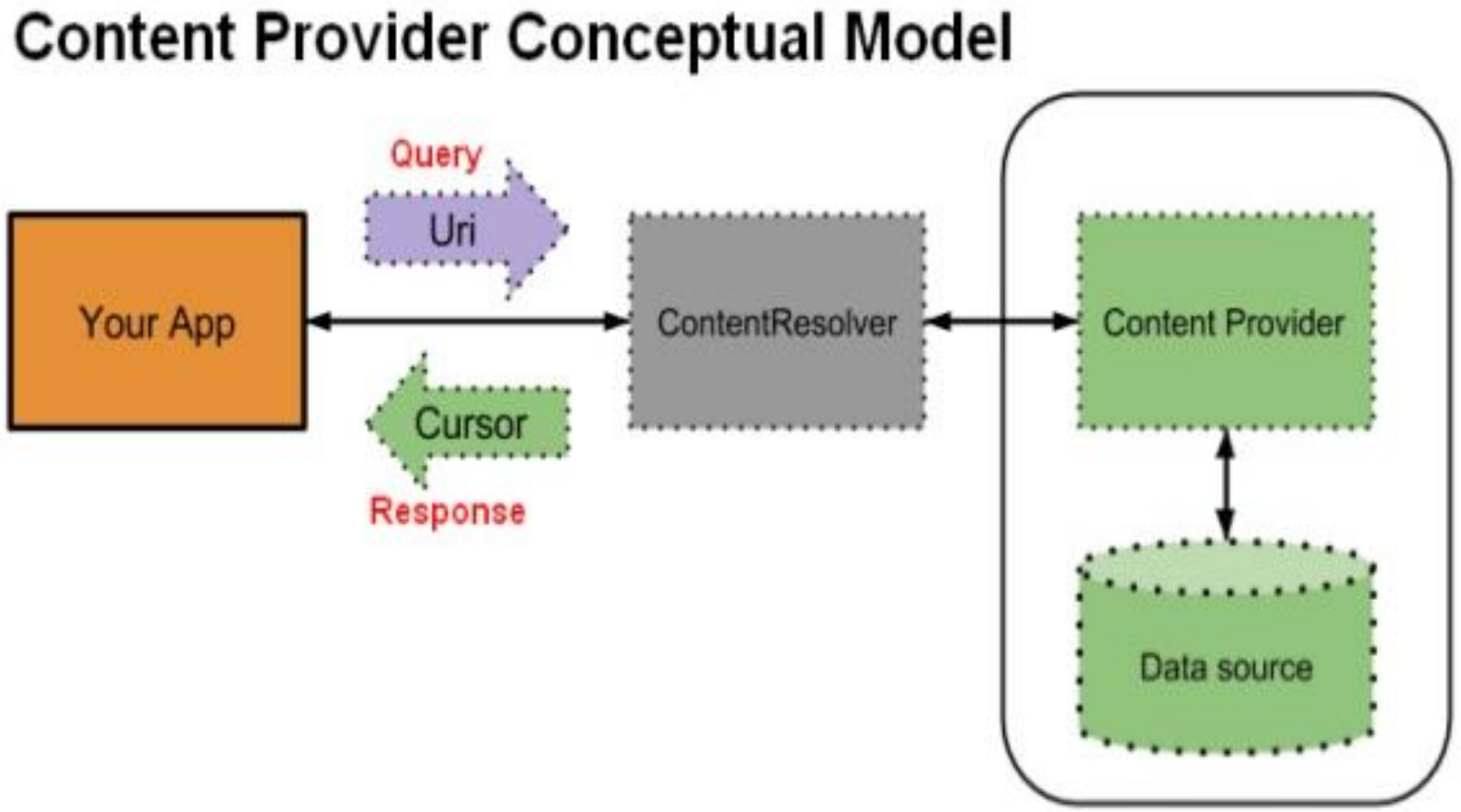


IMPLEMENTING THE CONTENTPROVIDER CLASS

- The ContentProvider instance manages access to a structured set of data by handling requests from other applications.
- All forms of access eventually call ContentResolver, which then calls a concrete method of ContentProvider to get access.

```
public class xyz extends ContentProvider
{
    --
    --
}
```

CONTENT PROVIDER



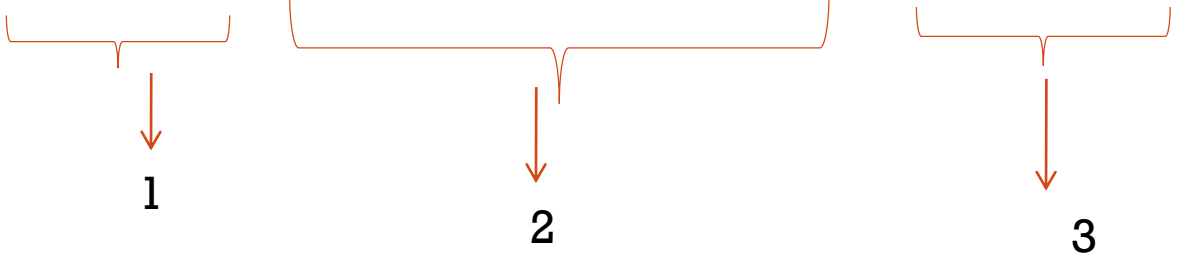
CONTENT URIS

- A **content URI** is a **URI** that identifies data in a provider.
- **Content URIs** include the symbolic name of the entire provider (its authority) and a name that points to a table (a path).
- When you call a client method to access a table in a provider, the **content URI** for the table is one of the arguments.

CONTENT URIS

Defining content URI

- static final String URL = "content://com.example.notesprovider/notes";



1) Standard prefix indicating that the data is controlled by a content provider. Its never modified.

2) The authority part of the URI; It identifies the Content Provider.

3) The path that the content provider uses to determine what kind of data(which table) is being requested.

CONTENT RESOLVER

URI

1) URI constant is used in all interactions with the content Provider.
   - Every ContentResolver method takes the URI as its first argument.

2) It identifies to which Provider the ContentResolver should talk to and which table of the provider is being targeted.

CONTENT RESOLVER

- The Content Resolver is the single, global instance in your application that provides access to your (and other applications') content providers.

- Clients access content providers indirectly through Content Resolver

```
ContentResolver  cr = getContentResolver();
```

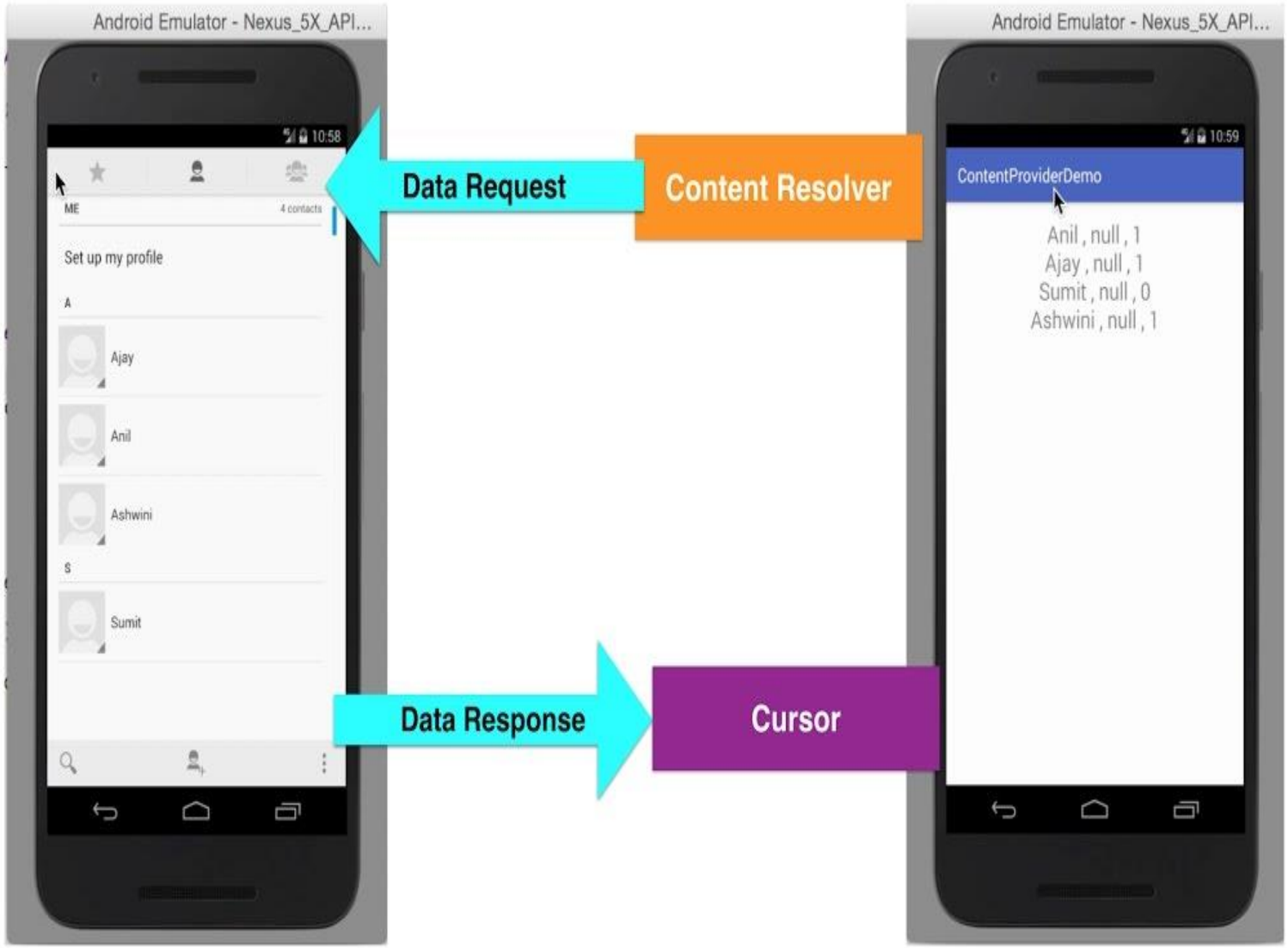
Provider

```
cr.insert(Uri.parse("content://com.example.notesprovider/notes"), values);
```

Providerclient

```
Cursor cursor =  cr.
  query(Uri.parse("content://com.example.notesprovider/notes"),
  new String[]{"note_date","content"},        where, new
  String[]{searchDate},  null);
```

CONTENTRESOLVER AND CURSOR



CURSOR

- This interface provides random read-write access to the result set returned by a database query.

1) moveToFirst(): Move the cursor to the first row.
2) moveToLast(): Move the cursor to the last row.
3) moveToNext(): Move the cursor to the next row.
4) moveToPrevious(): Move the cursor to the previous row.
5) moveToPosition(int position): Move the cursor to an absolute position.
6) getString(int columnIndex): Returns the value of the requested column as a String.

METHODS OF CONTENTPROVIDER CLASS

UPDATE()

- Update existing rows in your provider. Use the arguments to select the table and rows to update and to get the updated column values.
- Return the number of rows updated.

```
@Override
  public int update(Uri arg0, ContentValues arg1, String arg2, String[] arg3)
  {
        // TODO Auto-generated method stub
        return 0;
  }
```

DELETE()

- Delete rows from your provider. Use the arguments to select the table and the rows to delete.
- Return the number of rows deleted.

```
@Override
 public int delete(Uri arg0, String arg1, String[] arg2)
  {
        // TODO Auto-generated method stub
        return 0;
  }
```

GETTYPE()

- Return the MIME type corresponding to a content URI.

```
@Override
  public String getType(Uri arg0)
  {
        // TODO Auto-generated method stub
        return null;
  }
```

INSERT()

- Insert a new row into your provider. Use the arguments to select the destination table and to get the column values to use.
- Implement this to handle requests to insert a new row.
- Return a content URI for the newly-inserted row.

public abstract **Uri** insert (**Uri** uri, **ContentValues** values)

1) Uri: The content:// URI of the insertion request.

2) ContentValues: A set of column_name/value pairs to add to the database.

ONCREATE()

- Initialize your provider.

- The Android system calls this method immediately after it creates your provider. Notice that your provider is not created until a ContentResolver object tries to access it.

- Returns Boolean value:
  - ✓ true if the provider was successfully loaded
  - ✓ false otherwise

QUERY()

- Retrieve data from your provider. Use the arguments to select the table to query, the rows and columns to return, and the sort order of the result.

- Return the data as a Cursor object.

public **Cursor** query (**Uri** ,

String[] projection,

String selection,

String[] selectionArgs,

String sortOrder)

ARGUMENTS OF QUERY() METHOD

Parameters	Meaning
uri	Uri: The URI to query. This will be the full URI sent by the client; if the client is requesting a specific record, the URI will end in a record number that the implementation should parse and add to a WHERE or HAVING clause, specifying that _id value. This value cannot be null.
projection	String: The list of columns to put into the cursor. If null all columns are included. This value may be null.
selection	String: A selection criteria to apply when filtering rows. If null then all rows are included. This value may be null.
selectionArgs	String: You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings. This value may be null.
sortOrder	String: How the rows in the cursor should be sorted. If null then the provider is free to define the sort order. This value may be null.

Thank you