Activity

Application Entry Point

- Android App requires at least three things.
  1. an Activity class
  2. a layout file
  3. a Manifest file

```
<activity android:name=".MainActivity">
 <intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
</activity>
```

- When an application is launched,
  - the Android runtime creates an **Intent object** and inspects the **manifest** file.
  - It's looking for a specific value of the intent-filter node;

Starting Activities

- Activities can be started in one of two ways:
  - activity is marked as the **launchable** inside the AndroidManifest.xml
  - an activity can be started by an **intent** from the same app or any other app.

```
<activity android:name="com.example.myapp.ExampleActivity"
android:exported="true">
<intent-filter>
  <action android:name="com.example.myapp.ExampleActivity.START_ME" />
  <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
```

explicit intent

```
val intent = Intent()
intent.action = "com.example.myapp.ExampleActivity.START_ME"
startActivity(intent)
```

Activities

- **Activities** are one of the fundamental **building blocks** of apps on the Android platform.
- They serve as the **entry point** for a user's interaction with an app
- Activities are application's **presentation layer**:
  - An activity corresponds to a single screen of the application
  - The *home activity* is shown when the user launches an application
- Different activities can **exchange information** one with each other.

Activities and UI components

- Activities use **Fragments** and **Views** to layout and display information, and to respond to user actions.
- Some of these components can interact with the user by handling **events** (e.g. Buttons).
- Two ways to build UI

Programmatic approach

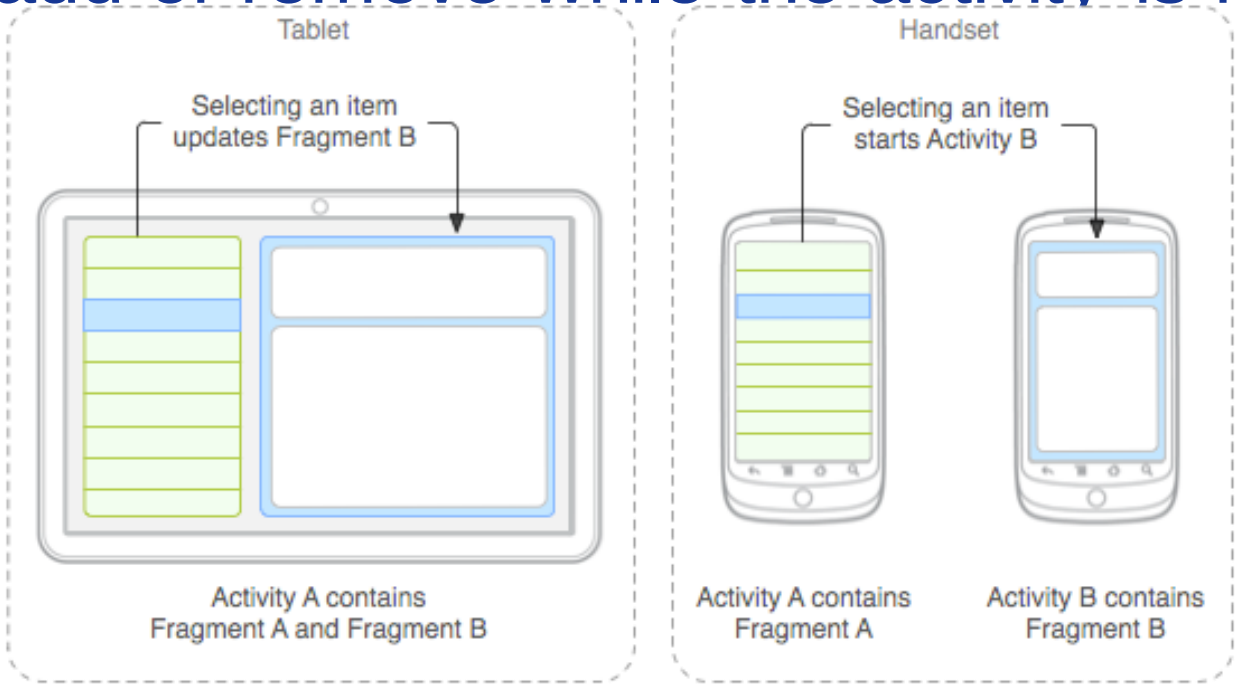
```
val button = Button(this)
val text= TextView(this)
text.text = "Hello world"
```

Declarative approach

```
< TextView android.text=@string/hello” android:textcolor=@color/blue
android:layout_width=“fill_parent” android:layout_height=“wrap_content” />
< Button android.id=“@+id/Button01” android:textcolor=“@color/blue”
android:layout_width=“fill_parent” android:layout_height=“wrap_content” />
```

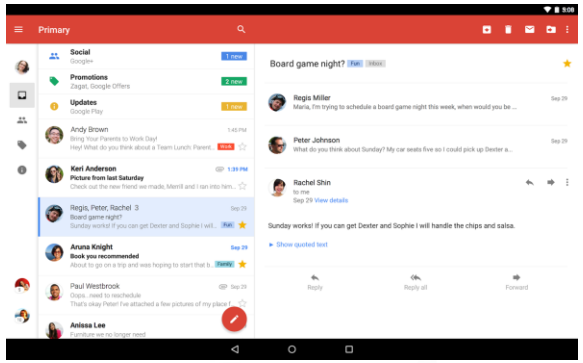
Fragments

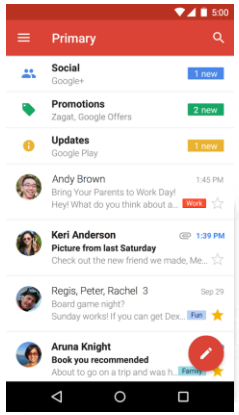
- A Fragment represents a behavior or a portion of user interface in an **Activity**.
- is a modular section of an **Activity**, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running



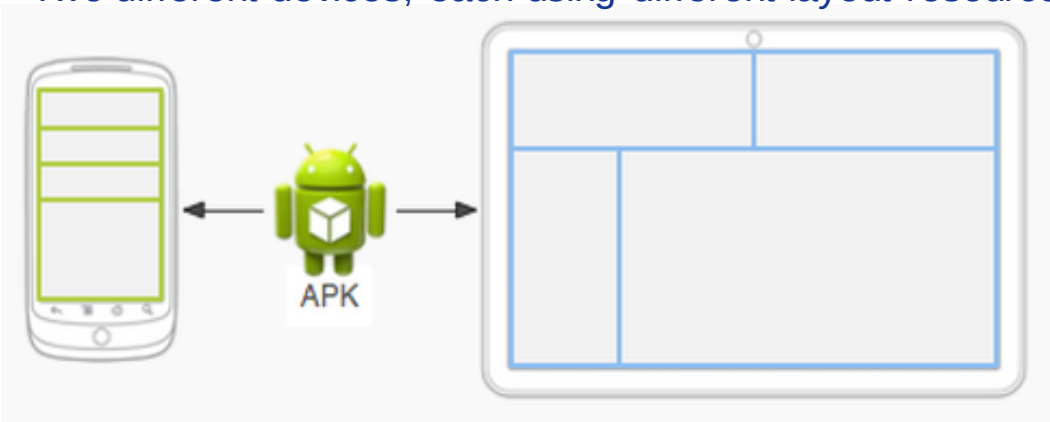
An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

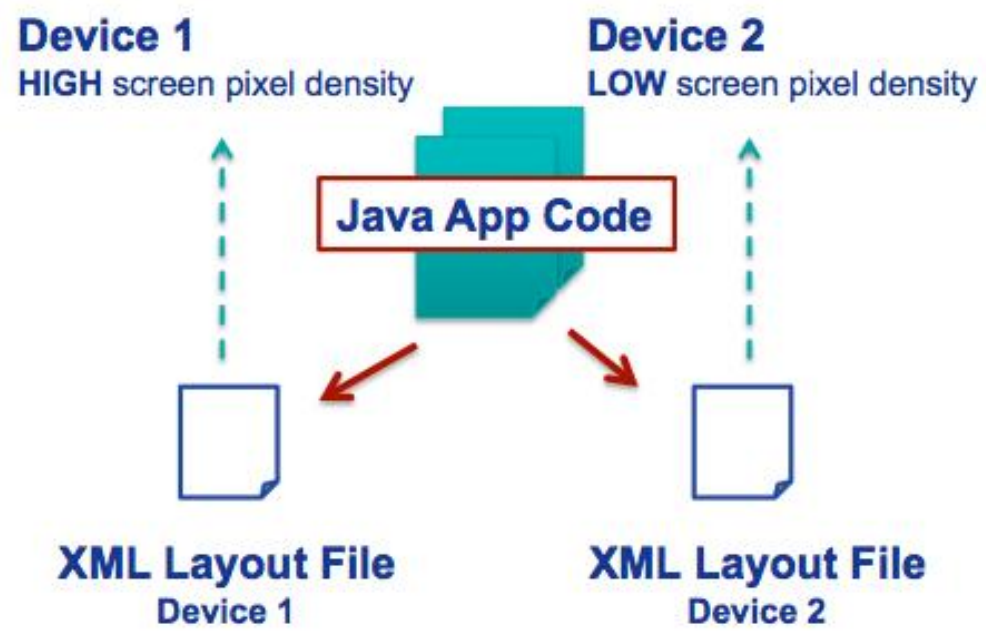
Layout automatic selection





Two different devices, each using different layout resources.





- **Runtime:** Android detects the current device configuration and loads the appropriate resources for the application
- Add a new XML file if you need to support a new device

Providing Alternative Resources

- Every application should provide alternative resources to support specific device configurations.
- For instance, you should include alternative drawable resources for **different screen densities** and alternative string resources for **different languages**.

```
res/
   drawable/
      icon.png
      background.png
   drawable-hdpi/
      icon.png
      background.png
```

Screen pixel density (dpi)
hdpi=High-density screens; approximately 240dpi.

for details visit
developer.android.com

ar=Arabic
Specific layout for Arabic

```
res/
   layout/
      main.xml
   layout-ar/
      main.xml
```

UI components

- Usually the two approaches are mixed

Declarative approach

XML Code → Defines the layouts and resources used by the application

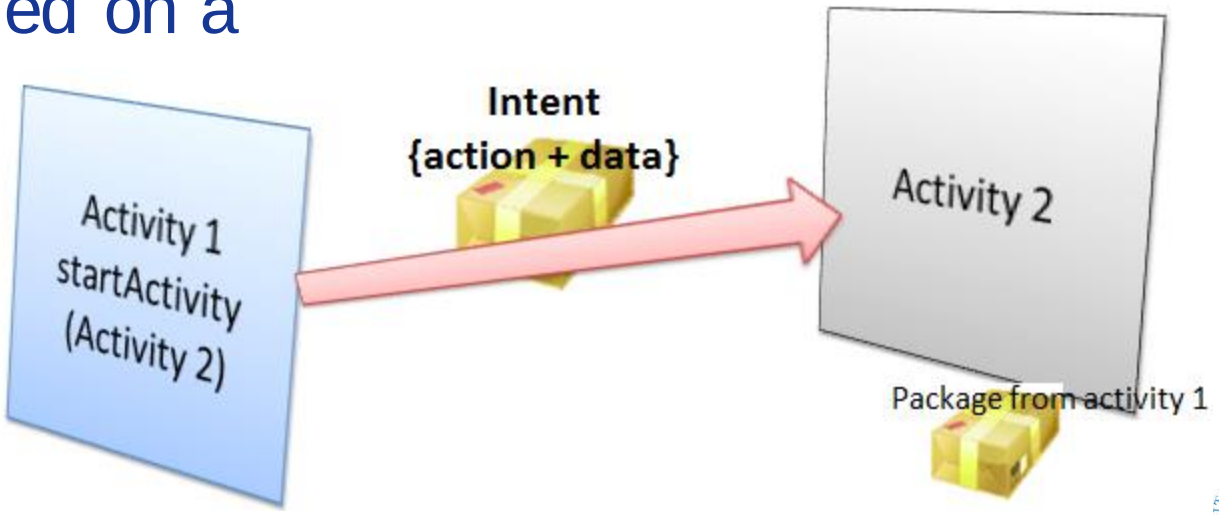
Programmatic approach

Java/Kotlin Code → Manages the events and handles the interaction with the user

Intent

Intents

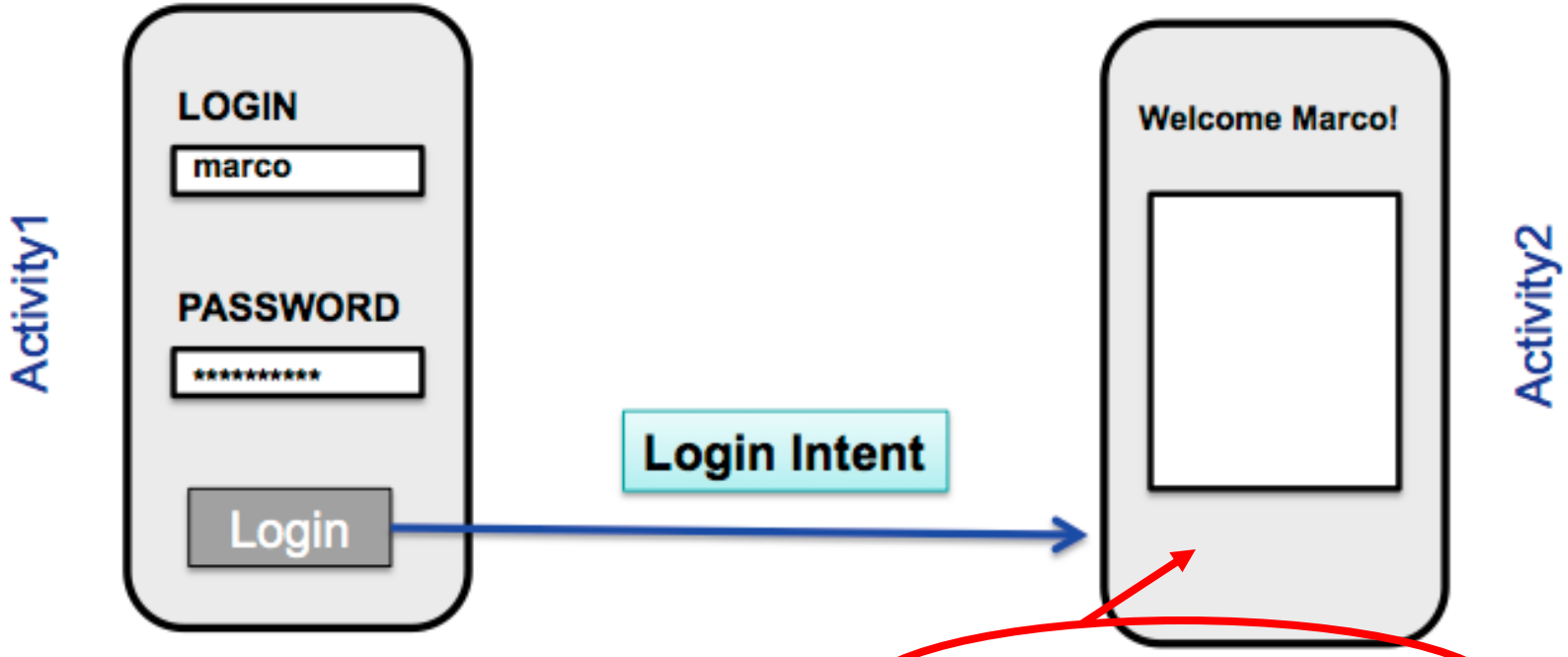
- **Inter-application message-passing framework.**
- They allow:
  - to start and stop Activities and Services
  - to broadcast messages system-wide or to an explicit Activity, Service, or Broadcast Receiver
  - to request an action be performed on a particular piece of data.
- Two kind of intents:
  - **explicit intents**
  - **implicit intents**



Explicit Intents

- A component explicitly specify the target component of the intent.

*Activity1 **explicitly** launches Activity 2 with an explicit intent*



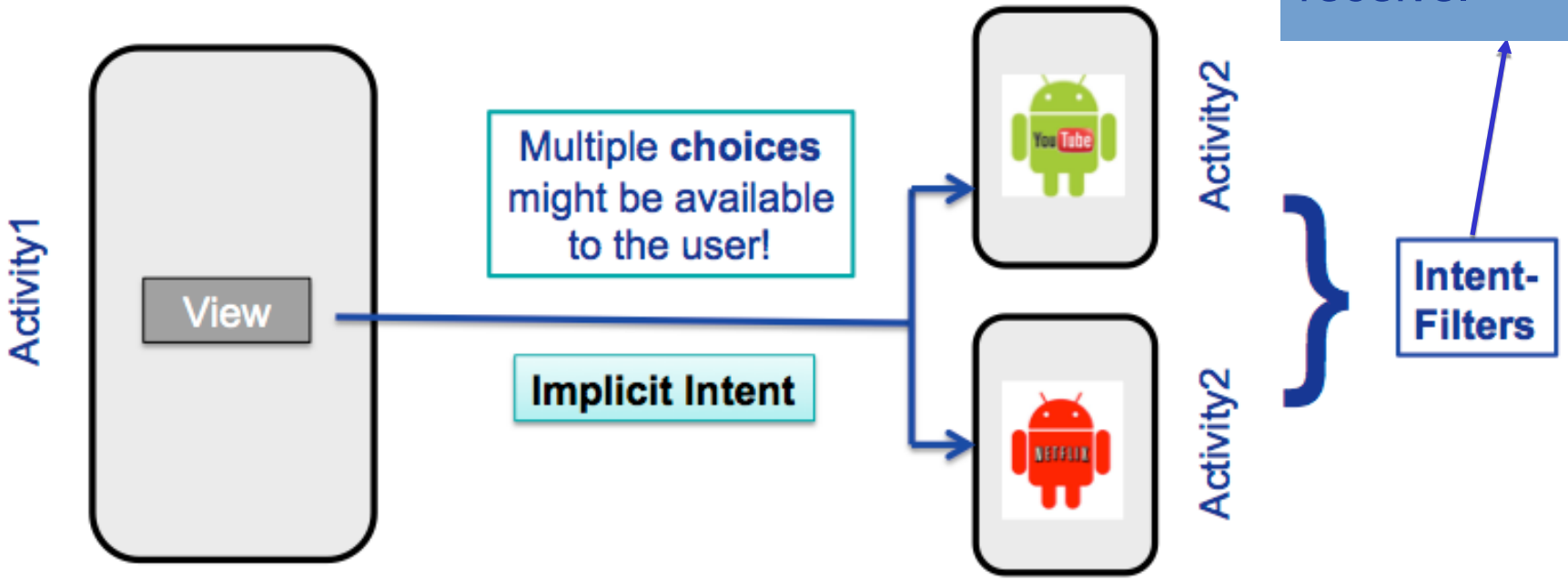

```
val intent = Intent(getApplicationContext(), SignupActivity.::class.java)
startActivity(intent)
```

Implicit Intents

- The component specify the **type** of the intent.

Activity1 specify the intent "view a video"

Expressions that specifies the type of intents
that the component would like to receive.




```
//showWebPage
val intent = Intent(Intent.ACTION_VIEW, Uri.parse("http://www.ab.it"))
startActivity(intent)
```

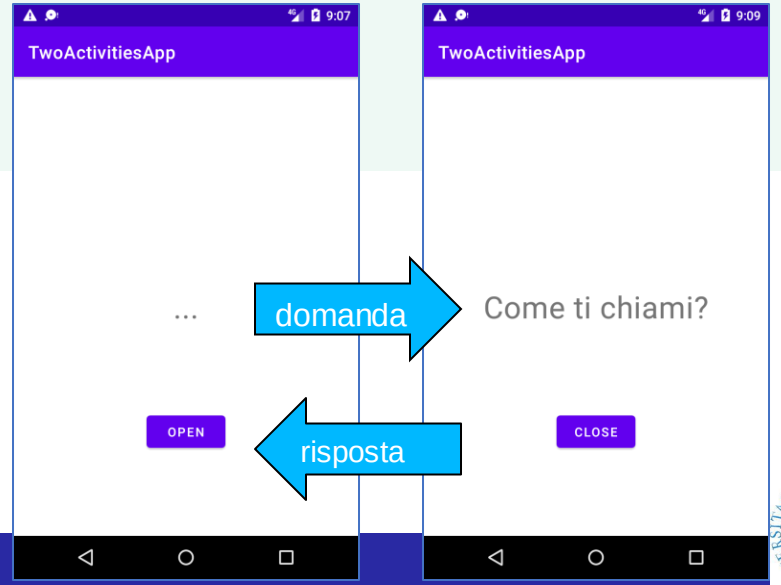
startActivityForResult

- **startActivity** will start a new activity and not care when where and how that activity finishes.
- **startActivity()** will start the activity you want to start without worrying about getting any result from new child activity started by startActivity to parent activity.
- **startActivityForResult()** waits for callbacks when the started activity decided to finish
- **startActivityForResult()** starts another activity from your activity and it expect to get some data from newly started child activity by startAcitvityForResult() and return that to parent activity.

startActivityForResult

```
class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val btn = findViewById<Button>(R.id.btn_open)
    btn.setOnClickListener {
      val intentAct2 = Intent(this, Activity2::class.java)
      intentAct2.putExtra("QUESTION", "Come ti chiami?")
      startActivityForResult(intentAct2, ACT2)
    }
  }

  override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == ACT2){
      if (resultCode == Activity.RESULT_OK){
        val msg = data?.getStringExtra("ANSWER")
        val tv = findViewById<TextView>(R.id.tv_hello)
        tv.text = msg
      }
    }
  }
}
```

```
class Activity2 : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_2)
    val saluti = intent.getStringExtra("QUESTION")

    val tv = findViewById<TextView>(R.id.tv_hello2)
    tv.text = saluti

    val btn = findViewById<TextView>(R.id.btn_close)
    btn.setOnClickListener {
      val returnIntent = Intent()
      returnIntent.putExtra("ANSWER", "Saluti da Tizio...")
      setResult(Activity.RESULT_OK, returnIntent)
      finish()
    }
  }
}
```



startActivityForResult is deprecated!

- The startActivityForResult is deprecated
- Activity Result API available from **androidx** 1.2.0 releases as an alternative to the **startActivityForResult**
- The **registerForActivityResult** method takes the following two parameters and returns **ActivityResultLauncher** as output
  - 1. ActivityResultContracts
  - 2. ActivityResultCallback

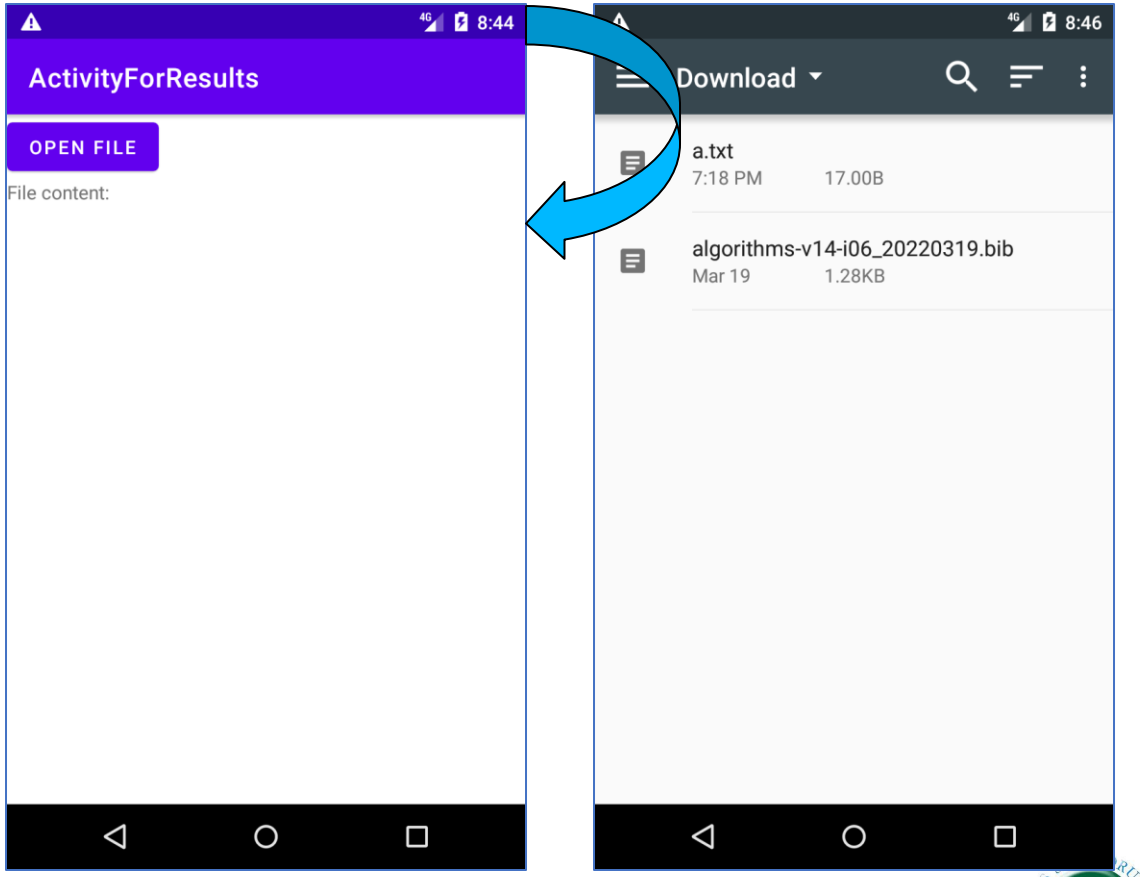
```
val intent = Intent(this, ActivityB::class.java)
resultLauncher.launch(intent)
```

```
var resultLauncher = registerForActivityResult(ActivityResultContracts.StartActivityForResult() ) { result ->
  if (result.resultCode == Activity.RESULT_OK){
    // parse results and perform actions
  }
}
```

https://developer.android.com/reference/androidx/activity/result/contract/ActivityResultContracts

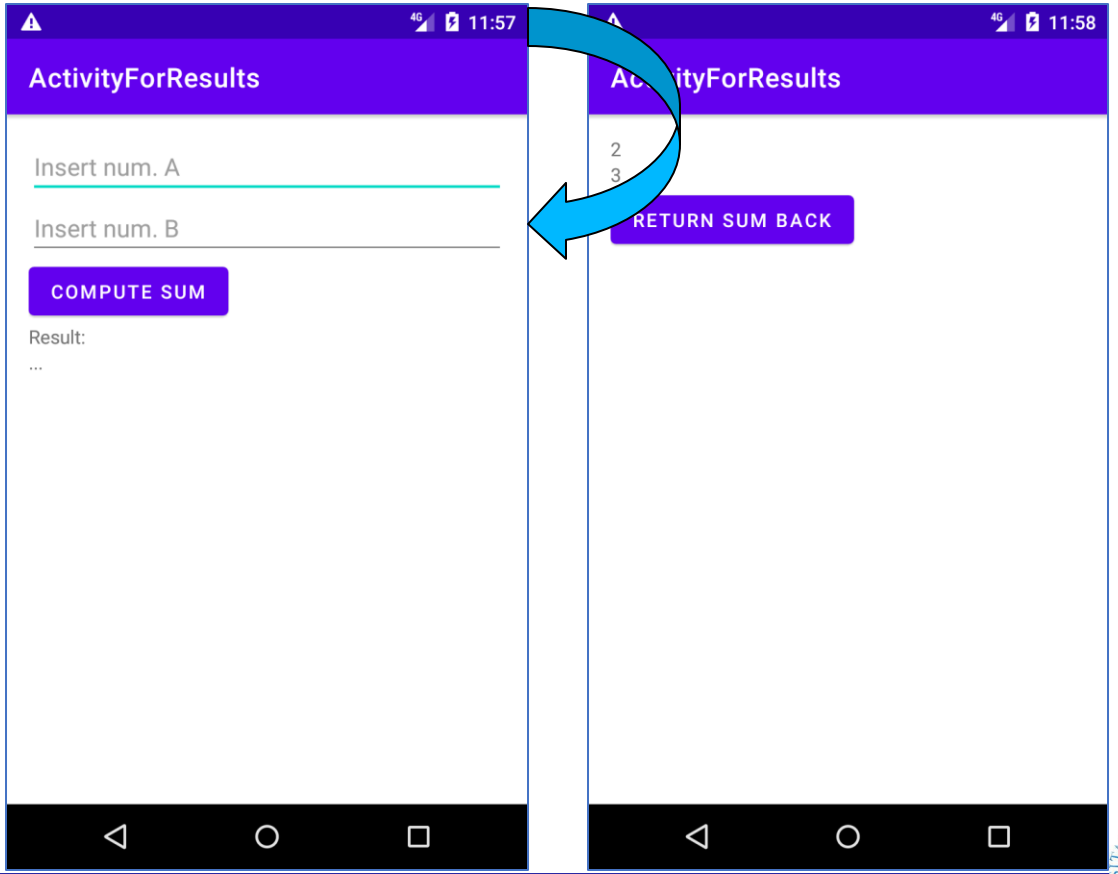
registerForActivityResult example1

```
class MainActivity : AppCompatActivity() {
  var resultLauncher = registerForActivityResult(ActivityResultContracts.GetContent()) { result ->
    val tv = findViewById<TextView>(R.id.tv_content)
    tv.text = result?.path ?: "Empty path!"
  }
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val btnOpen = findViewById<Button>(R.id.btn_open)
    btnOpen.setOnClickListener {
      resultLauncher.launch("*/*")
    }
  }
}
```



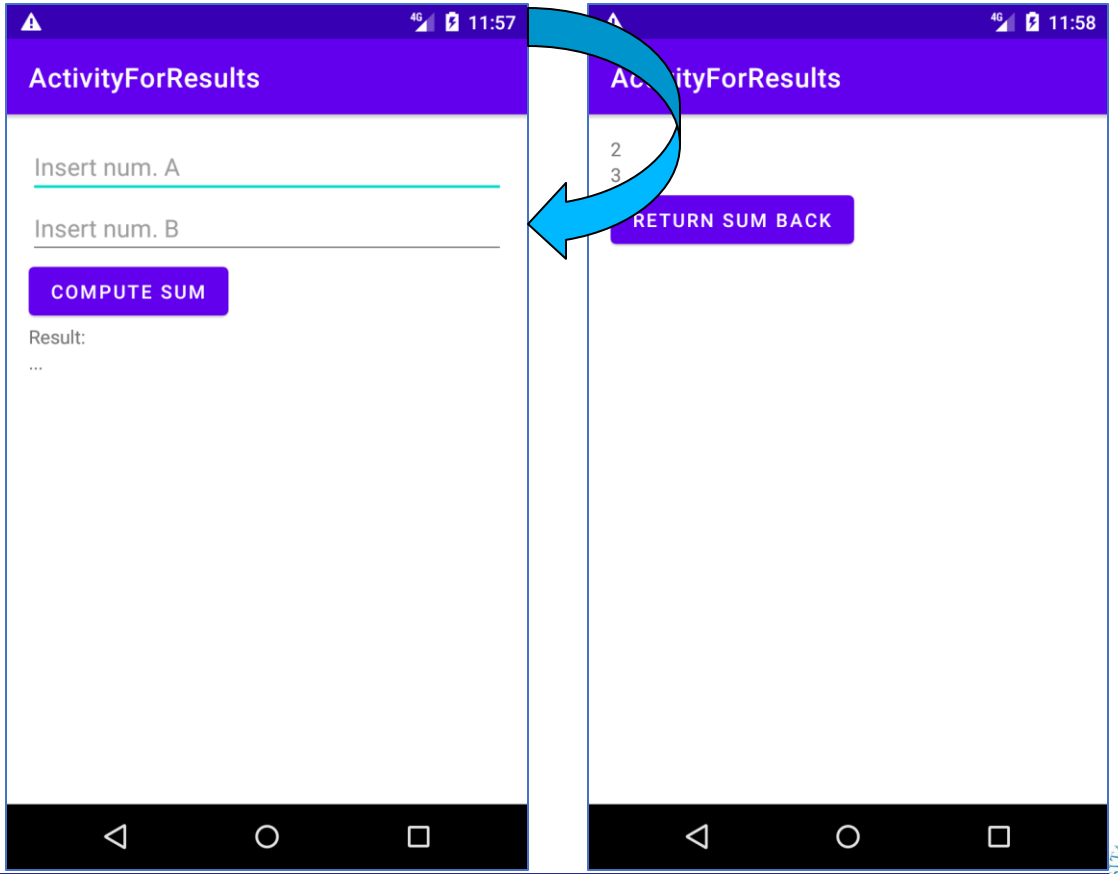
registerForActivityResult example2

```
class MainActivity : AppCompatActivity() {
  var openPostActivity =
      registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
        if(result.resultCode == Activity.RESULT_OK) {
          val sum = result.data?.getIntExtra("SUM", 0)
          Toast.makeText(applicationContext, "Result OK. SUM=${sum}", Toast.LENGTH_LONG).show()
          val tv = findViewById<TextView>(R.id.tv_result)
          tv.text = sum.toString()
        }
      }
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val btnOpen = findViewById<Button>(R.id.btn_sum)
    val tvA = findViewById<EditText>(R.id.et_a)
    val tvB = findViewById<EditText>(R.id.et_b)

    btnOpen.setOnClickListener {
      val intent = Intent(applicationContext, SecondActivity::class.java)
      intent.putExtra("A", tvA.text.toString())
      intent.putExtra("B", tvB.text.toString())
      openPostActivity.launch(intent)
    }
  }
}
```



registerForActivityResult example2

```
class SecondActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_second)
    val ex = intent.extras
    val a = ex?.getString("A")?.toInt()
    val b = ex?.getString("B")?.toInt()
    val tvA = findViewById<TextView>(R.id.tv_param_a)
    val tvB = findViewById<TextView>(R.id.tv_param_b)
    tvA.text = "$a"
    tvB.text = "$b"

    val btnReturn = findViewById<Button>(R.id.btn_return)
    btnReturn.setOnClickListener {
      val resultIntent = Intent()
      resultIntent.putExtra("SUM", a?.plus(b!!))
      setResult(Activity.RESULT_OK, resultIntent)
      finish()
    }
  }
}
```



Applying an Activity Result consists of three steps

- **Step 1**. Creating a contract
  - Contract is a class that implements the *ActivityResultContract<I,O>* interface. Where I defines the type of input data necessary to start the Activity, and O defines the callback result type.
  - For typical tasks, you can use out-of-the-box implementations

Creating a contract [1]

Creating the interface implementation

`ActivityResultContract`

Registering a contract [2]

Registering the result callback with

`registerForActivityResult()`

Launching a contract [3]

Calling to launch the contract

`launch()`

Applying an Activity Result consists of three steps

- **Step 2**. Registering a contract
  - The next step is to register the contract in the activity or fragment by calling registerForActivityResult().
  - You need to pass ActivityResultContract and ActivityResultCallback as parameters.

Creating a contract [1]

Creating the interface implementation

ActivityResultContract

Registering a contract [2]

Registering the result callback with

registerForActivityResult()

Launching a contract [3]

Calling to launch the contract

launch()

Applying an Activity Result consists of three steps

- **Step 3**. Launching a contract

- To start the Activity, we only need to call launch() on the ActivityResultLauncher object that we have obtained in the previous step.

Creating a contract [1]

Creating the interface implementation

ActivityResultContract

Registering a contract [2]

Registering the result callback with

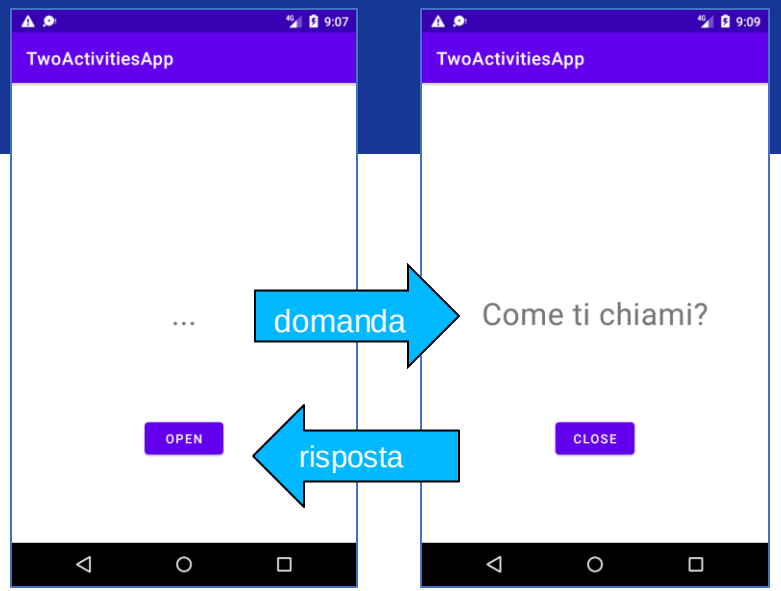
registerForActivityResult()

Launching a contract [3]

Calling to launch the contract

launch()

registerForActivityResult



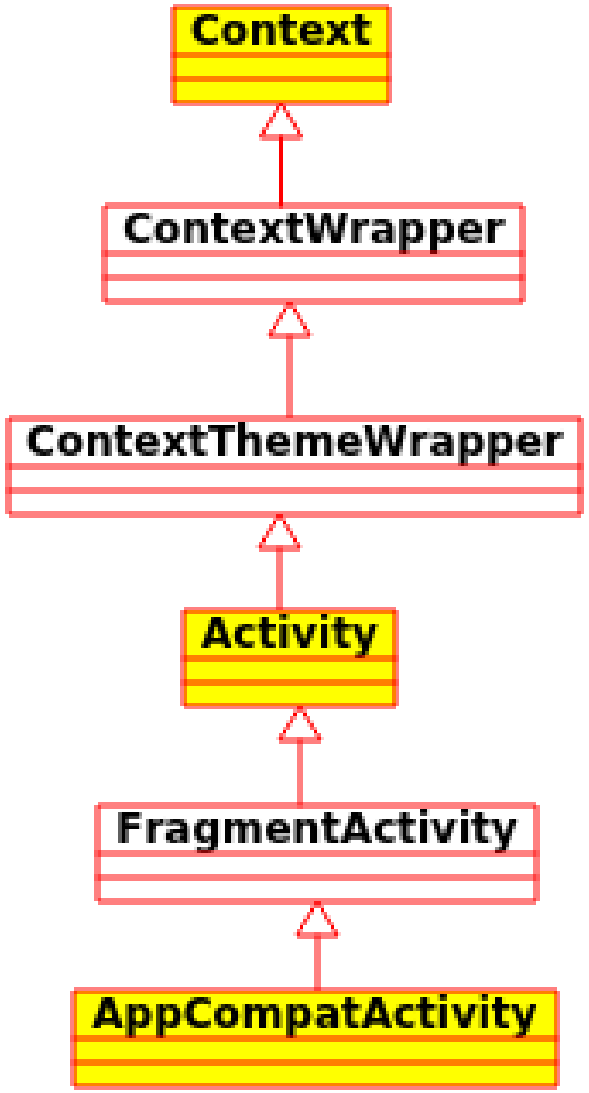

```
class MainActivity : AppCompatActivity() {
    // step 2. register a contract
    val activityLauncher = registerForActivityResult(MySecondActivityContract()) { result ->
    val tv = findViewById<TextView>(R.id.tv_hello)
    tv.text = result
  }
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val btn = findViewById<Button>(R.id.btn_open)
    btn.setOnClickListener {
      // step 3.
      activityLauncher.launch("Come ti chiami?")
    }
  }
}
```

```
// Step 1. creating a contract
class MySecondActivityContract : ActivityResultContract<String, String>() {
  override fun createIntent(context: Context, input: String?): Intent {
    val intentAct2 = Intent(context, Activity2::class.java)
    intentAct2.putExtra("QUESTION", input)
    return intentAct2
  }

override fun parseResult(resultCode: Int, intent: Intent?): String {
   var msg = ""
   if (resultCode == Activity.RESULT_OK){
     msg = intent?.getStringExtra("ANSWER").toString()
   }
   return msg
  }
}
```

Activities life-cycle

Activities

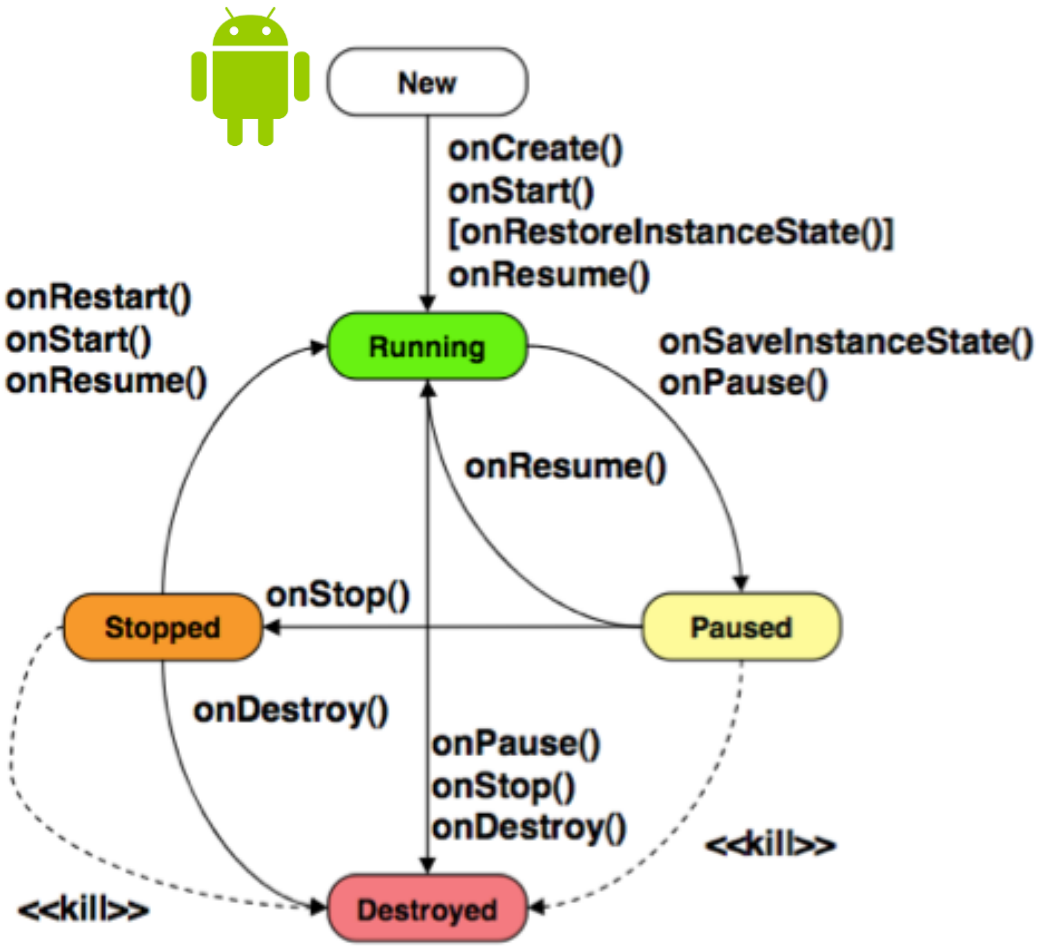
- Context allows access to application-specific **resources** and classes, **launching** activities, broadcasting and receiving **intents**, etc.
- An activity corresponds to a **single screen** of the application.
- Application components must listen for changes in the application state and react accordingly.



Activity states and lifecycle

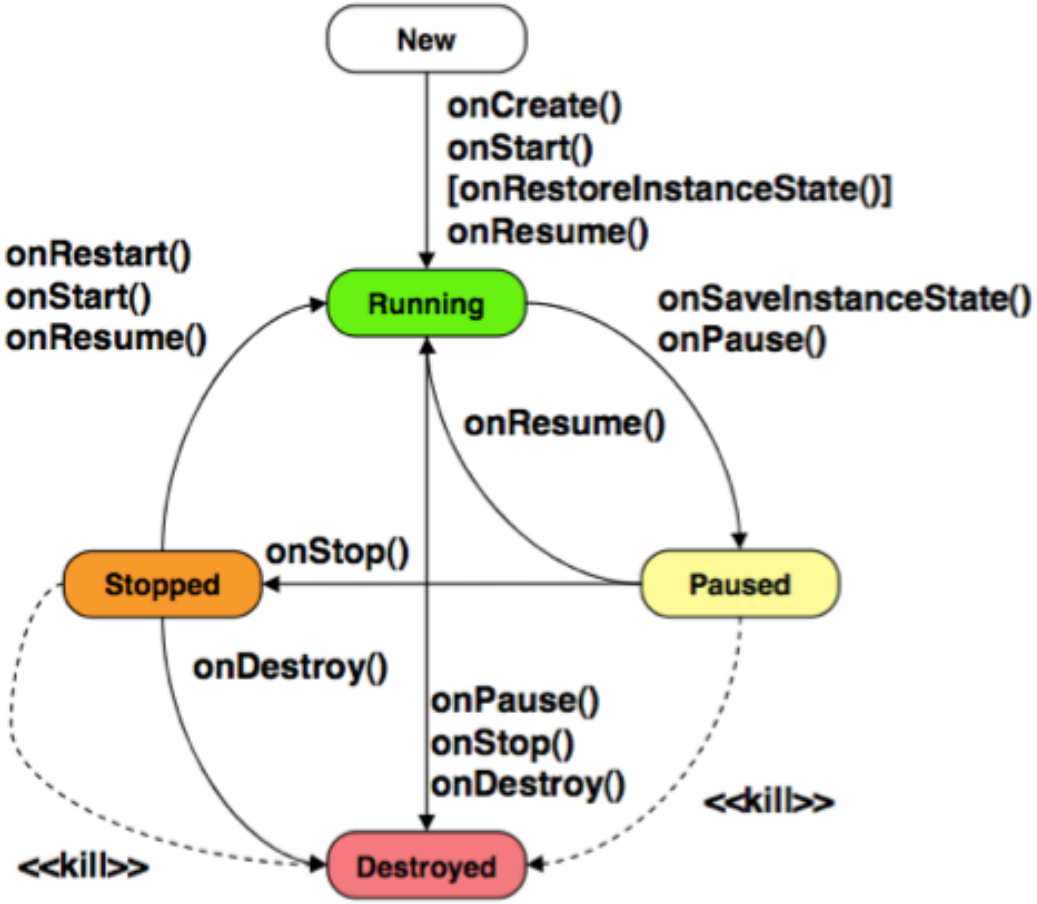
- *Running (resumed)*: Running
  the activity is in the foreground of the screen and has user focus.
- *Paused*: Paused
  another activity is in the foreground and has focus, but this one is still visible
- *Stopped*: Stopped
  the activity is completely obscured by another activity (“background").
- *Destroyed*

Destroied



Activity states and lifecycle

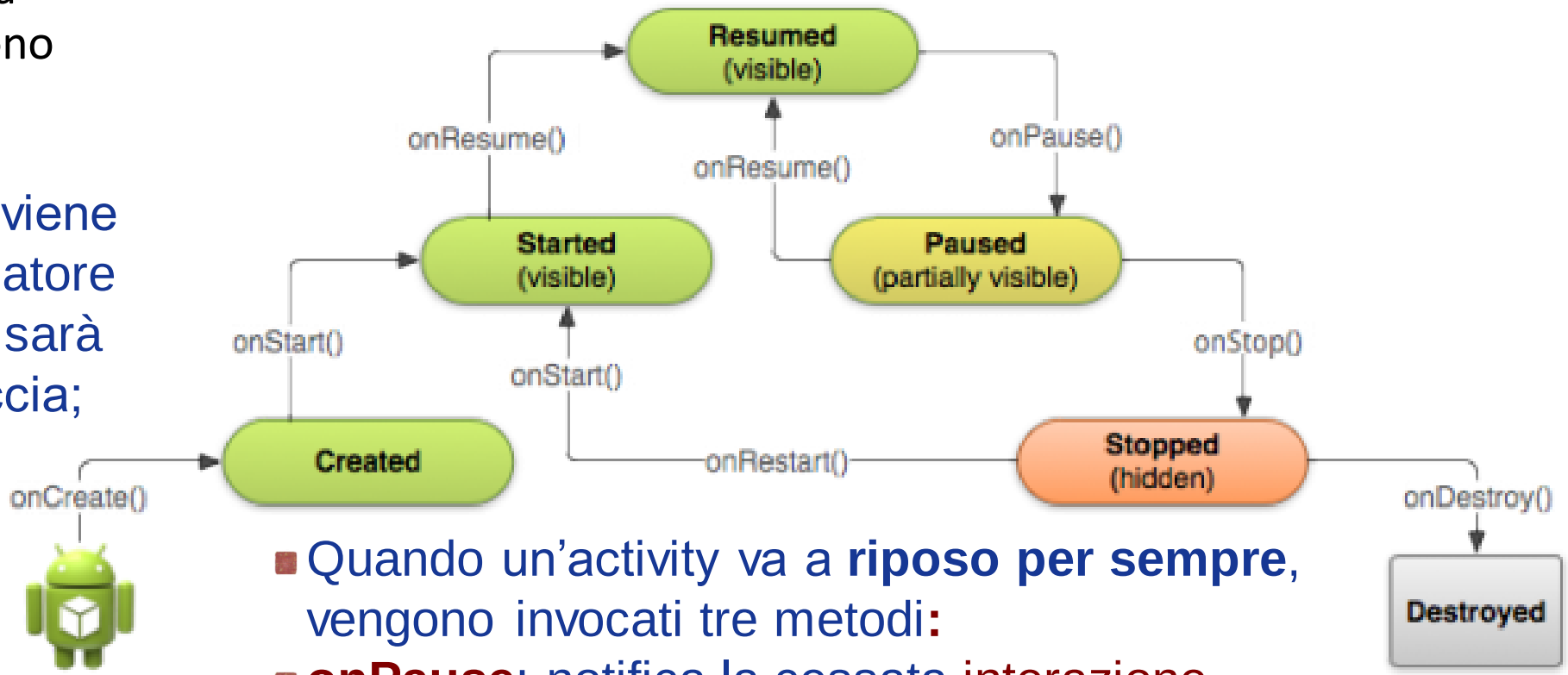
- When an activity transitions into and out of the different states, it is notified through various callback methods.
- To implement an Activity you have to
  extend the `Activity` class
  and
  override the callback methods.



Activity states and callback methods

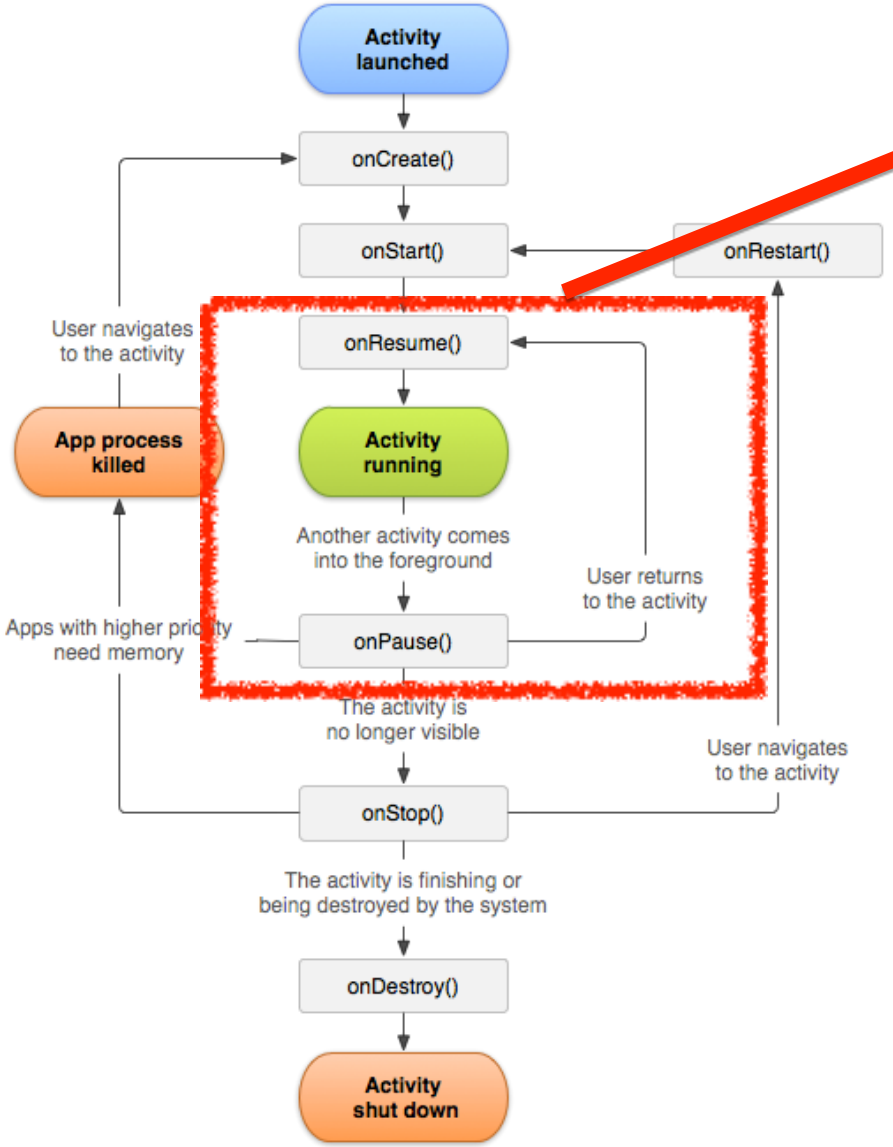
- Quando un'activity va in **esecuzione**, vengono invocati tre metodi:

- **onCreate**: l'activity viene creata. Il programmatore deve definire quale sarà il layout dell'interfaccia;
- **onStart**: l'activity diventa visibile.
- **onResume**: l'activity diventa la destinataria di tutti gli input dell'utente.



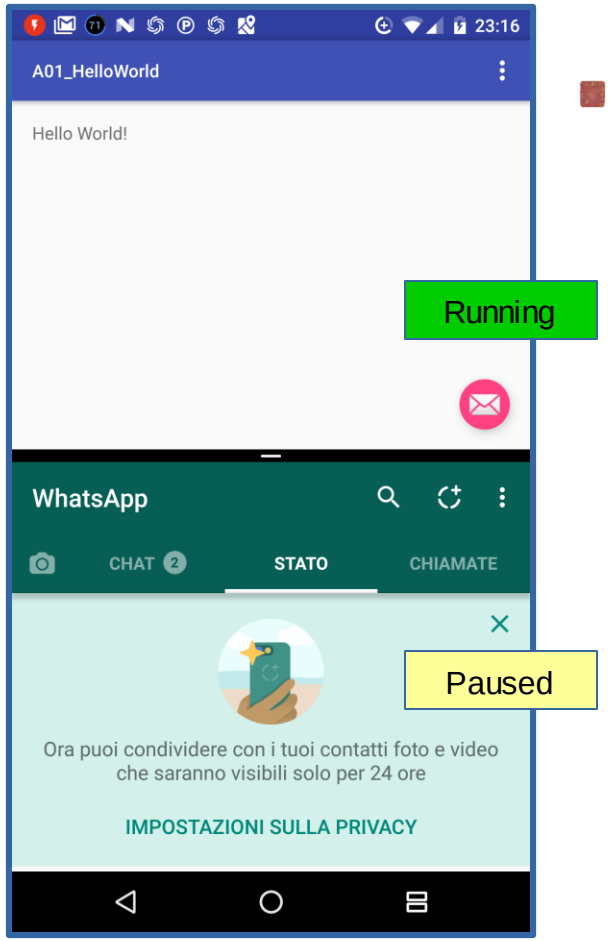
- Quando un'activity va a **riposo per sempre**, vengono invocati tre metodi**:**
- **onPause**: notifica la cessata interazione dell'utente con l'activity
- **onStop**: segna la fine della visibilità dell'activity
- **onDestroy**: segna la distruzione dell'activity.

Activity lifecycle loops: Foreground lifetime



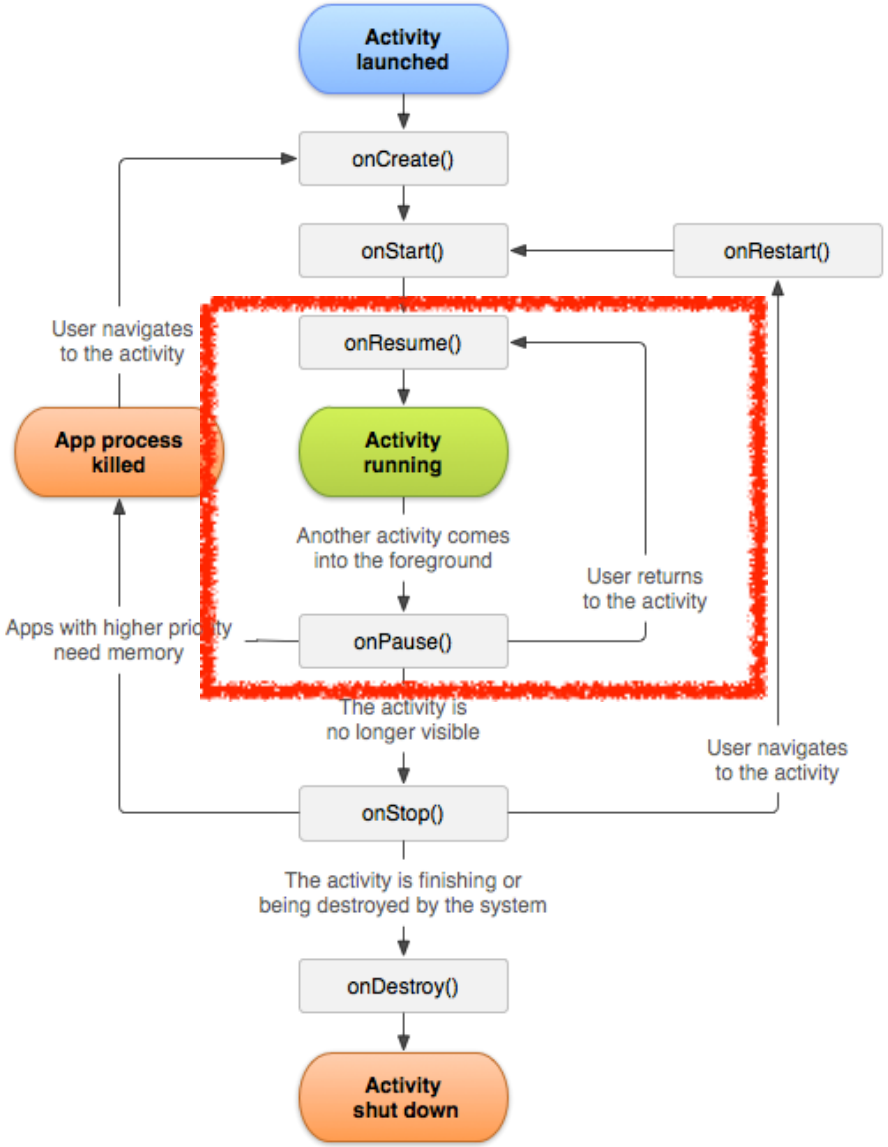
Foreground (active) lifetime

- The activity is in front of all other activities on screen and has user input focus



- Example: In Android 7.0 (API level 24) or higher, multiple apps run in multi-window mode. Because only one of the apps (windows) has focus at any time, the system **pauses** all of the other apps.

Activity lifecycle loops: onPause

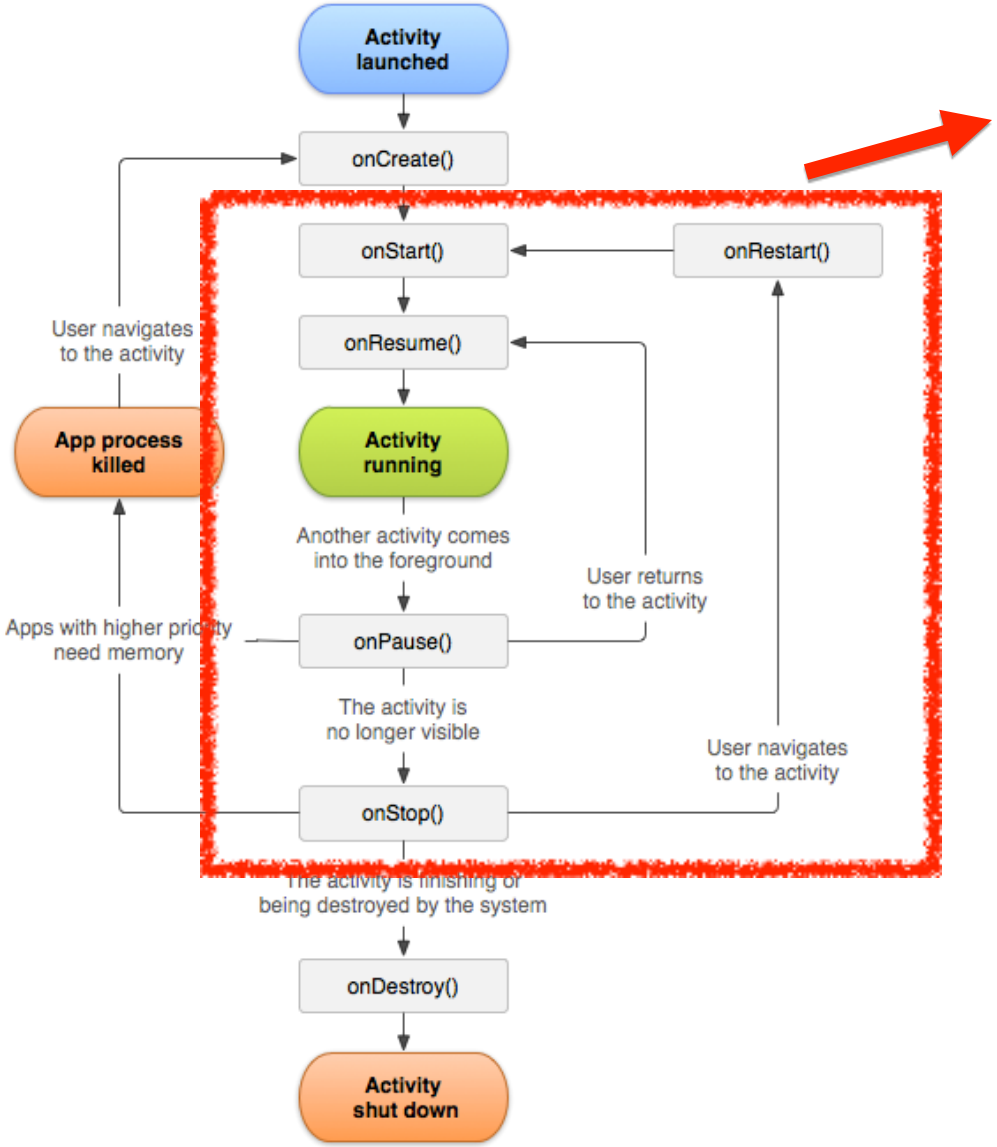


https://developer.android.com/guide/components/activities/activity-lifecycle.html#onpause

There are several reasons why an activity may enter **onPause** state.
For example:

- Some event interrupts app execution. This is the most common case.
- In Android 7.0 (API level 24) or higher, multiple apps run in multi-window mode. Because only one of the apps (windows) has focus at any time, the system pauses all of the other apps.
- A new, semi-transparent activity (such as a dialog) opens. As long as the activity is still partially visible but not in focus, it remains paused.

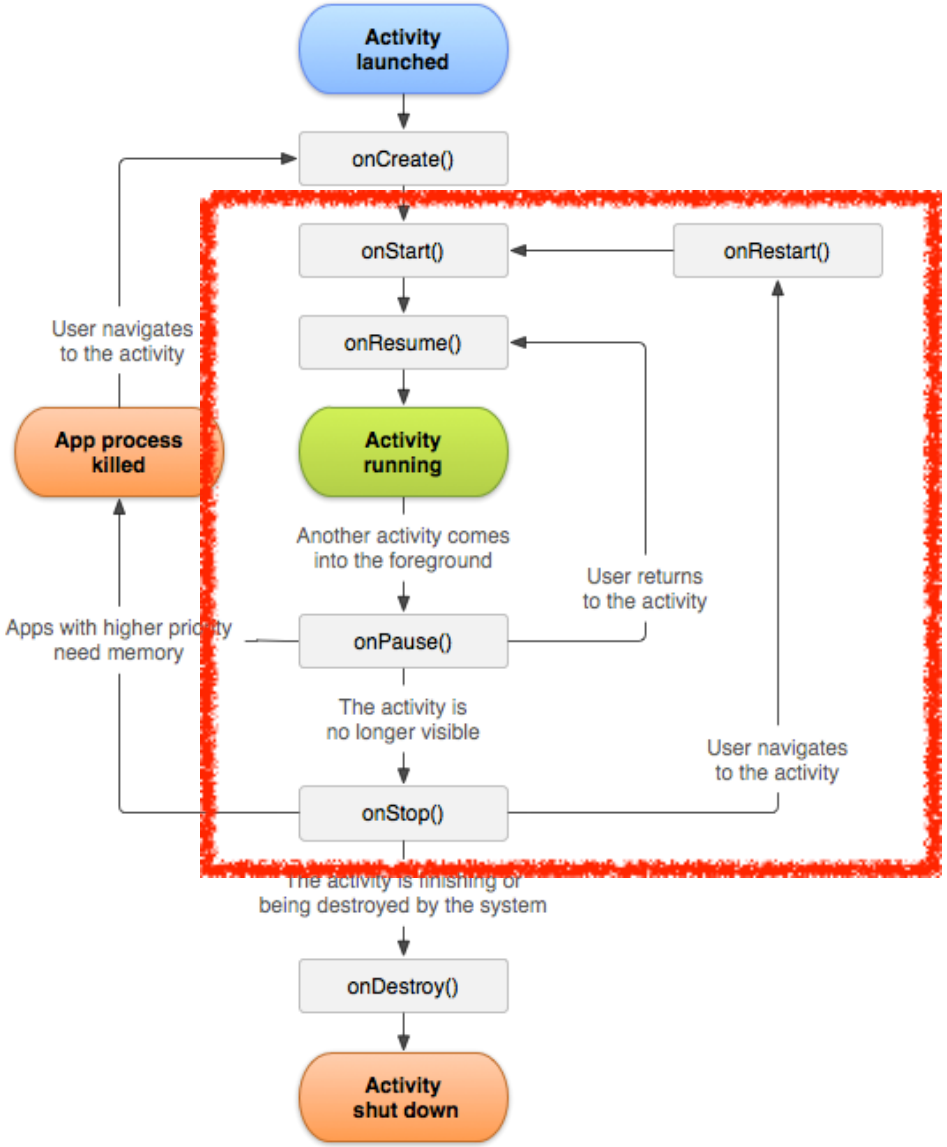
Activity lifecycle loops: Visible lifetime



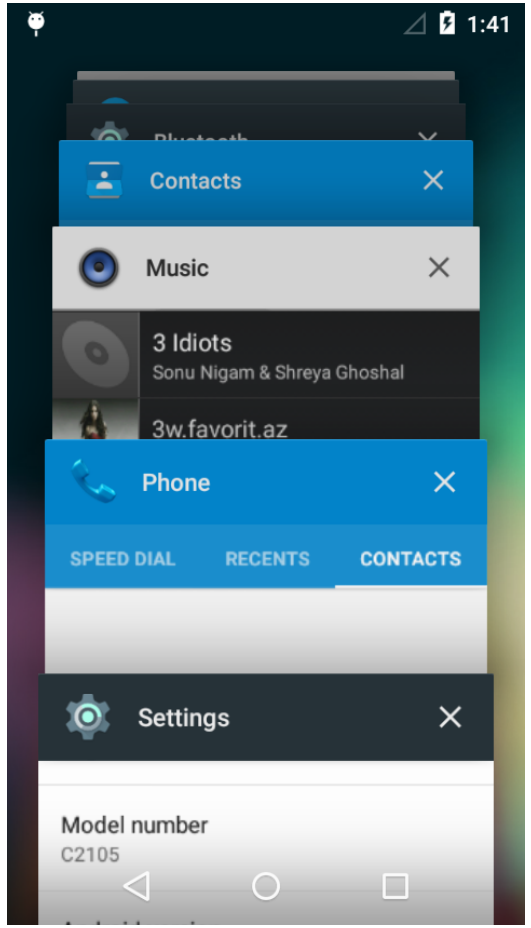
Visible lifetime

- The user can see the activity on-screen and interact with it.
  - `onStop()` is called when a new activity starts and this one is no longer visible.
- Between **onStart/onStop** methods, you can maintain resources that are needed to show the activity to the user.
  - E.g: you can register a BroadcastReceiver in `onStart()` to monitor changes that impact your UI, and unregister it in `onStop()` when the user can no longer see what you are displaying.

Activity lifecycle loops: onStop

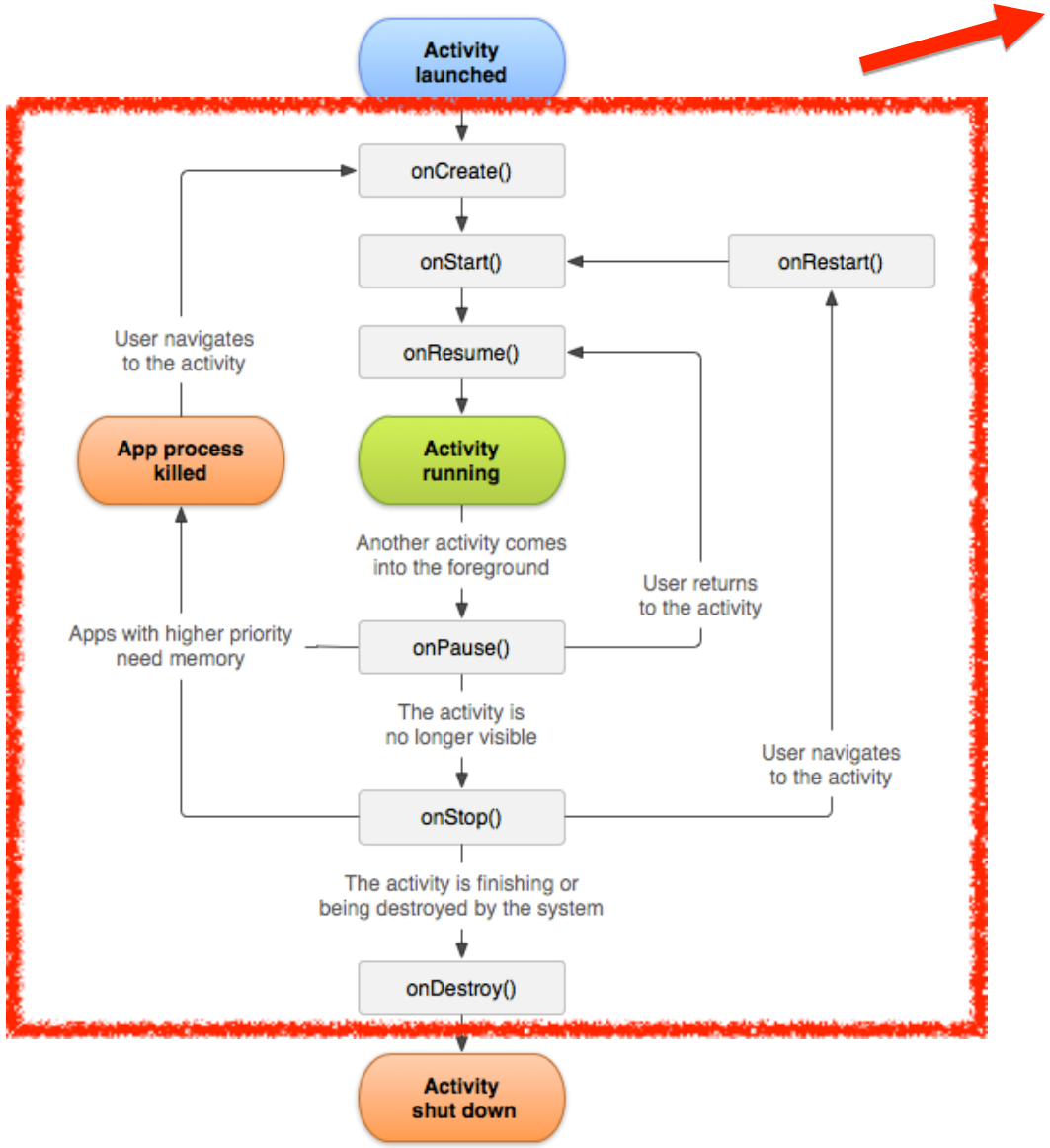


https://developer.android.com/guide/components/activities/activity-lifecycle.html#onstop



The system invokes the **onStop()** callback for example when the user select a different task using the the taskManager

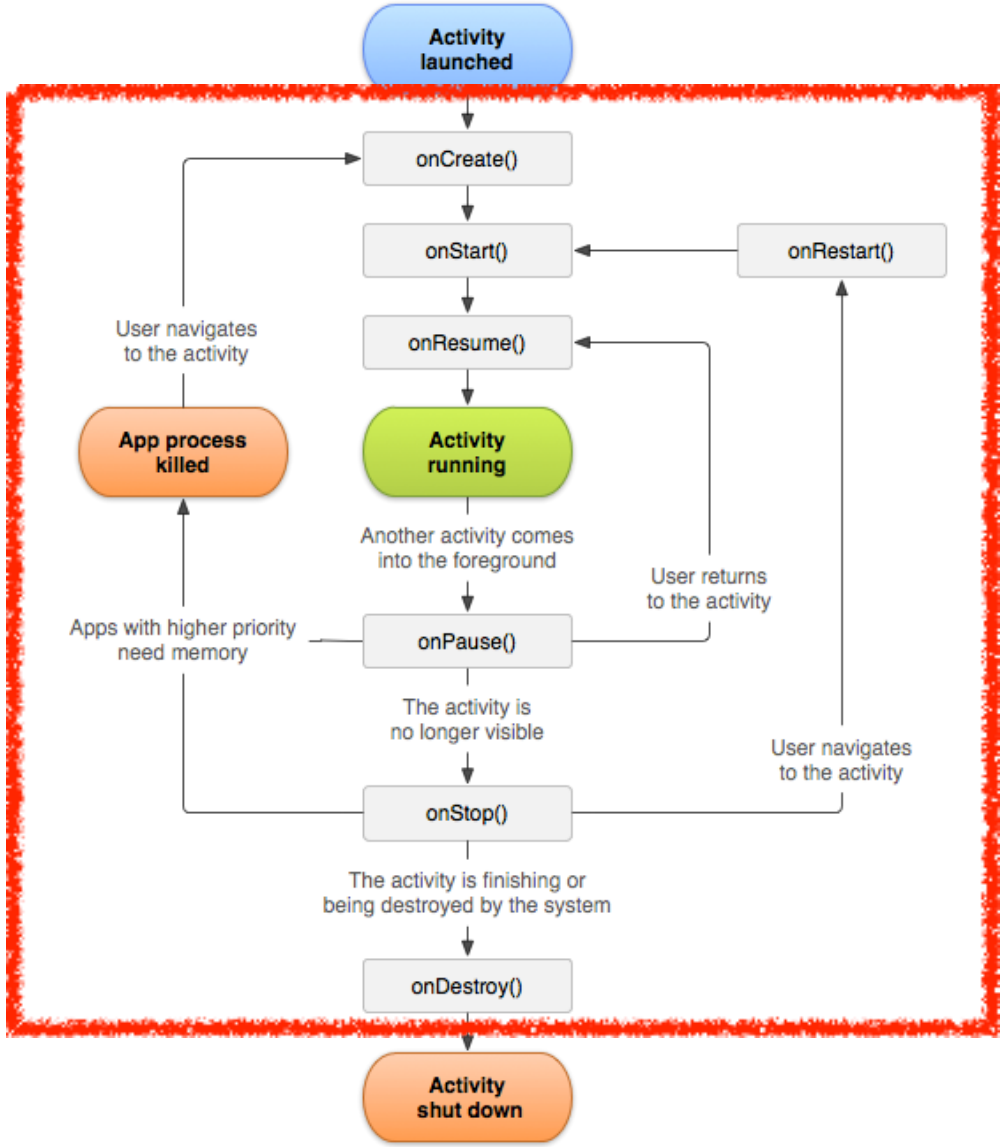
Activity lifecycle loops



Full lifetime

- Your activity should perform setup of "global" state (such as defining layout) in `onCreate()`,
- Release all resources in `onDestroy()`.
  - E.g.: if your activity has a thread running in the background to download data from the network, it might create that thread in `onCreate()` and then stop the thread in `onDestroy()`.

Activity lifecycle loops: onDestroy



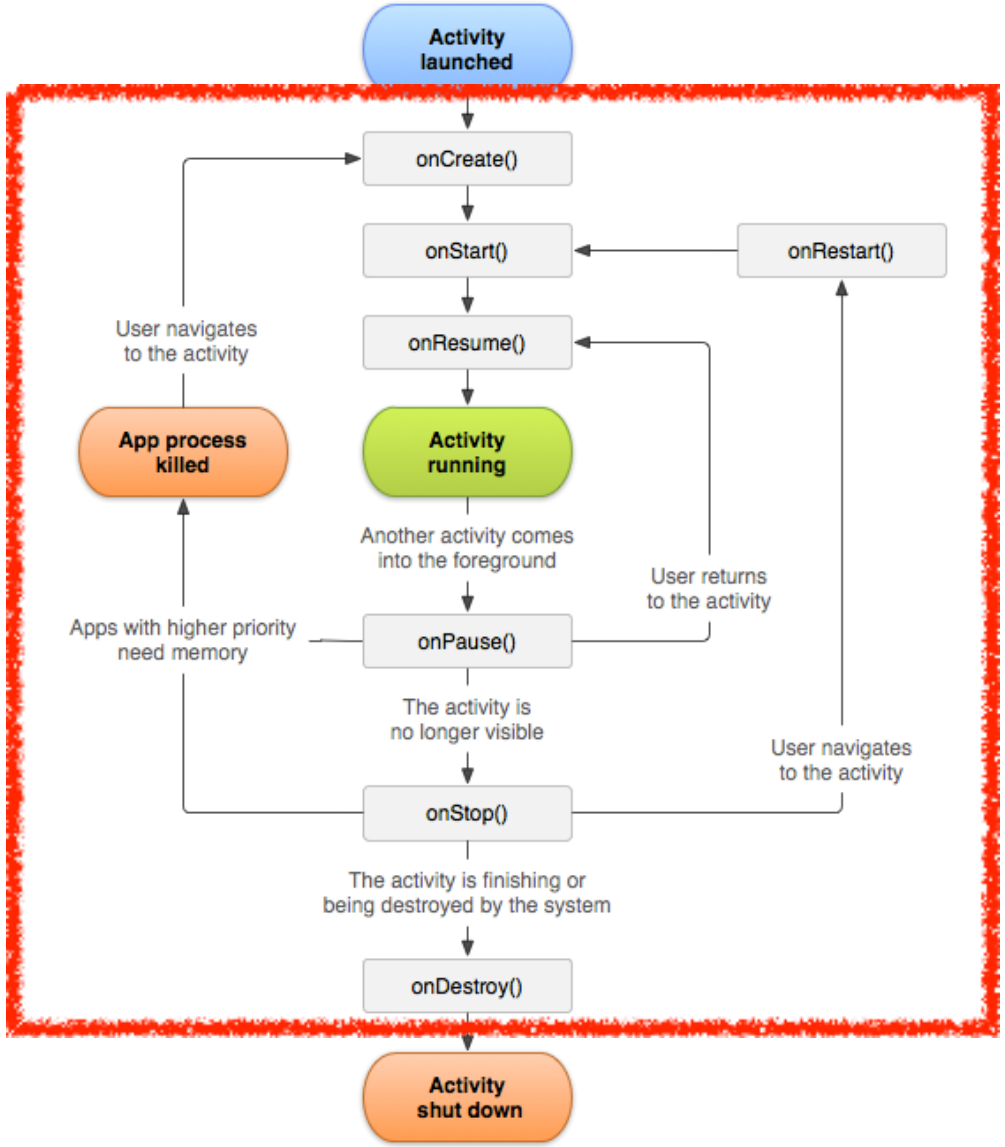
https://developer.android.com/guide/components/activities/activity-lifecycle.html#ondestroy

onDestroy is called before the activity is destroyed. **This is the final call that the activity receives.**

The system either invokes this callback because the activity is finishing due to

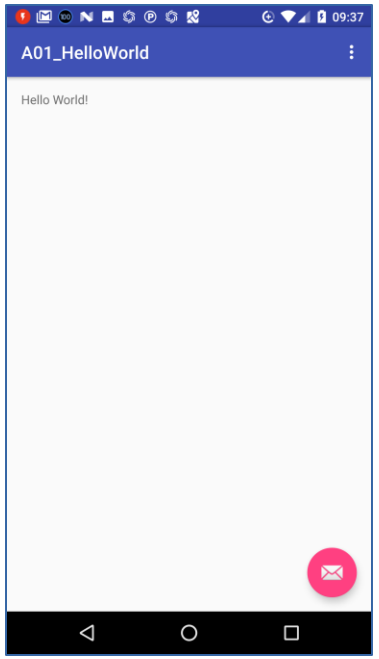
- someone's calling *finish()*,
- the system is temporarily destroying the process containing the activity to save space.
- Etc.

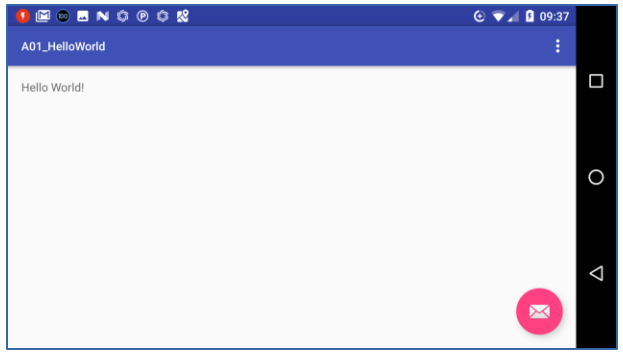
Activity lifecycle loops: onDestroy

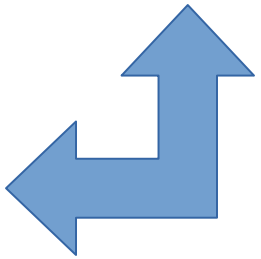


Example: The system destroys the Activities and recreates it Again to change the orientation of the device.

MainActivity: onPause
MainActivity: onStop
MainActivity: **onDestroy**
MainActivity: **onCreate**
MainActivity: onStart
MainActivity: onResume







Activity Class

kotlin.Any
↳ android.content.Context
↳ android.content.ContextWrapper
↳ android.view.ContextThemeWrapper
↳ android.app.Activity

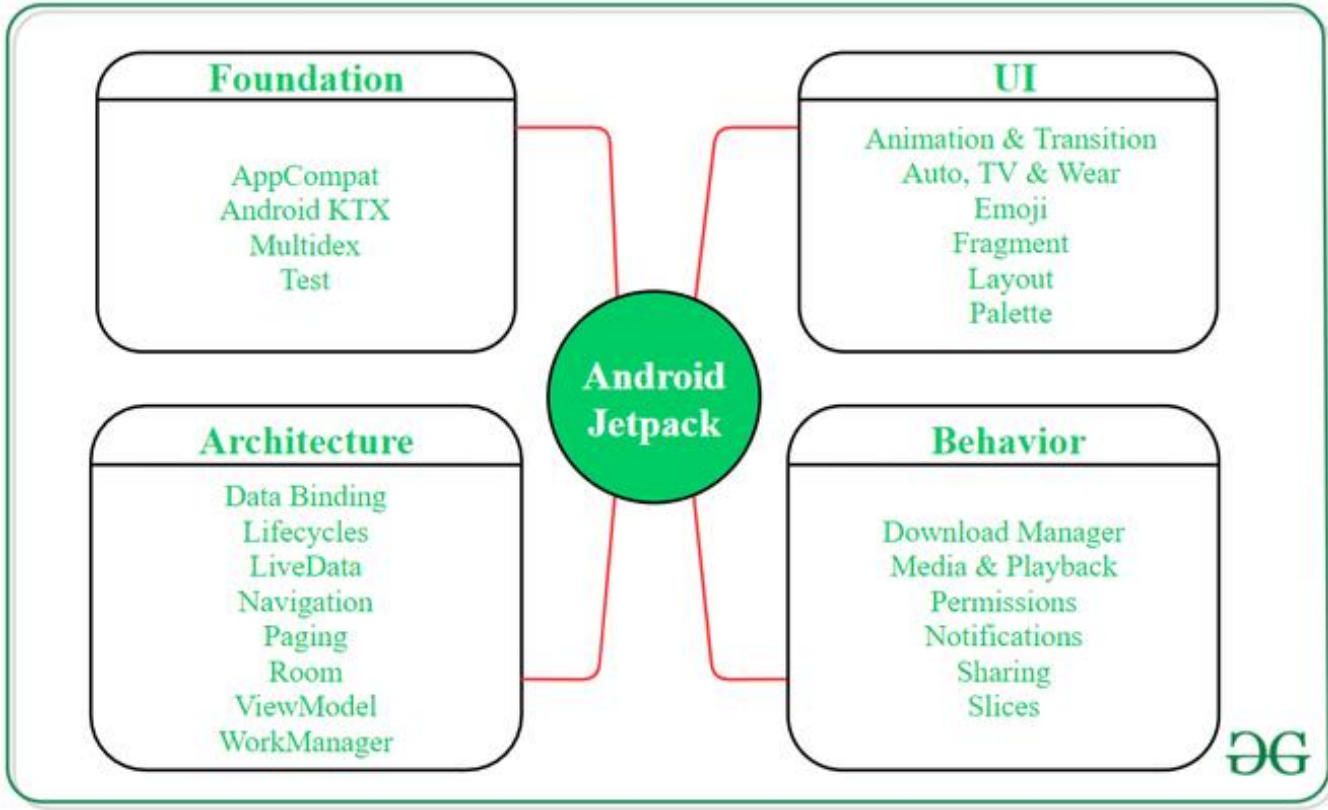
- Activity is the base class of all other activities
- An activity is a single, focused thing that the user can do.
- Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView.

```
class MainActivity : Activity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }
}
class SecondActivity : Activity() {
  //...
}
```

androidx

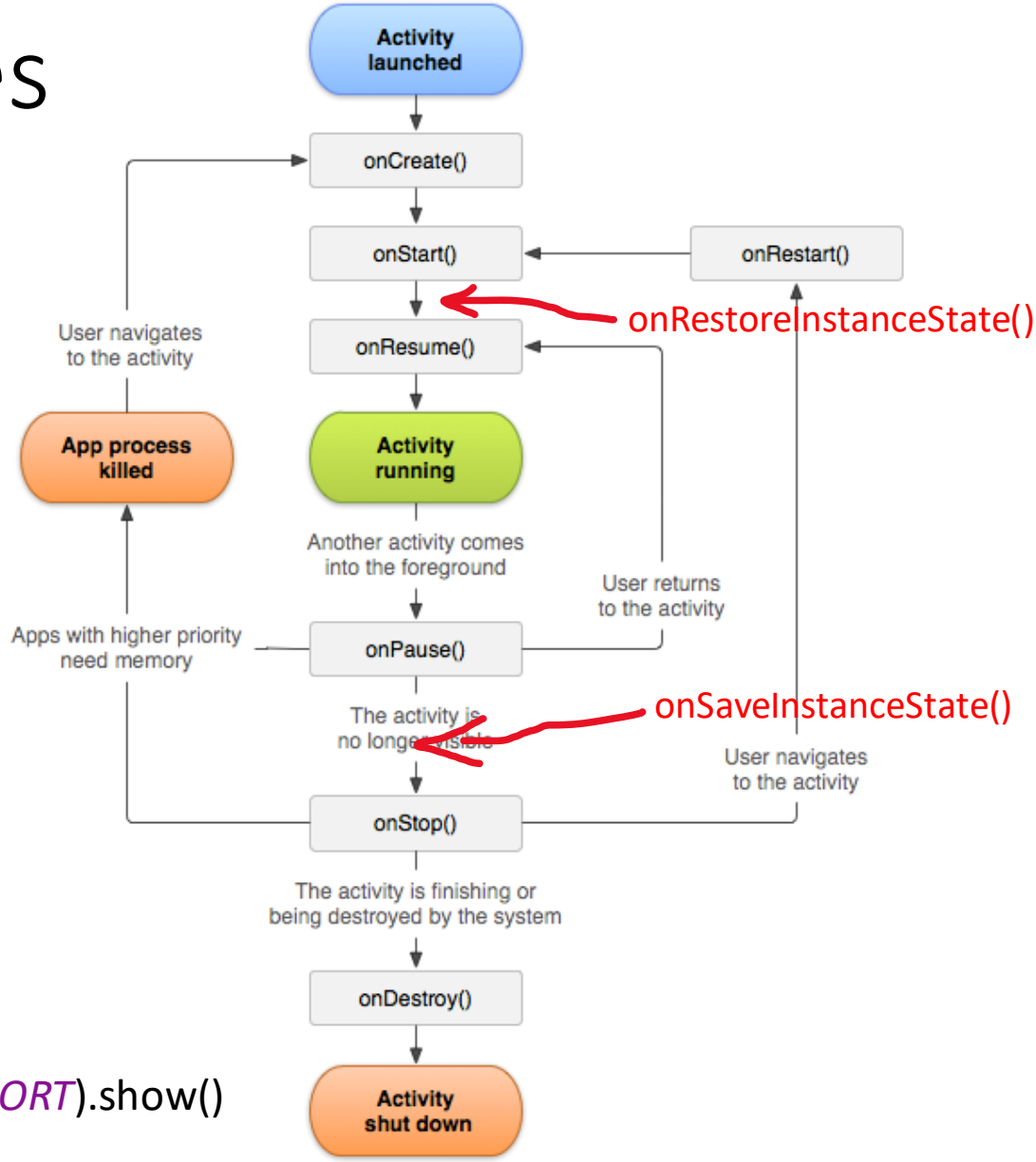
androidx.activity.ComponentActivity
↳ androidx.fragment.app.FragmentActivity
↳ androidx.appcompat.app.AppCompatActivity

- `Androidx.appcompat.app.AppCompatActivity`
- **`AndroidX`** is an improved version of the android support libraries released within the jetpack.
- **Android jetpack** is a set of components and tools designed to accelerate the android development.
- `AppCompatActivity is` Base class for activities that wish to use some of the newer platform features on older Android devices.



Preserving State in Activities

- There are two more callbacks
- onSaveInstanceState()
  onRestoreInstanceState()




```
override fun onSaveInstanceState(outState: Bundle) {
  super.onSaveInstanceState(outState)
  val value_to_store = "This is a test!"
  outState.putString("hello", value_to_store)
}
override fun onRestoreInstanceState(savedInstanceState: Bundle?) {
  super.onRestoreInstanceState(savedInstanceState)
  val restored_value = savedInstanceState!!.getString("hello")
  Toast.makeText(this@MainActivity, restored_value, Toast.LENGTH_SHORT).show()
}
```

Activity states and callback methods in HelloWorld

A01-HelloWorld: Activity states and callback methods

A01_HelloWorld - [~/Documents/Courses_/PDM/src/A01_HelloWorld] - [app] - ~/Documents/Courses_/PDM/src

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

A01_HelloWorld app src main java it insubria pdm a01_helloworld MainActivity

MainActivity.java

MainActivity onResume()

```
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {...}
34
35      @Override
36      protected void onStart(){
37          super.onStart();
38          Log.v(TAG, "onStart");
39      }
40
41      @Override
42      protected void onResume(){
43          super.onResume();
44          Log.v(TAG, "onResume");
45      }
```

Android Monitor

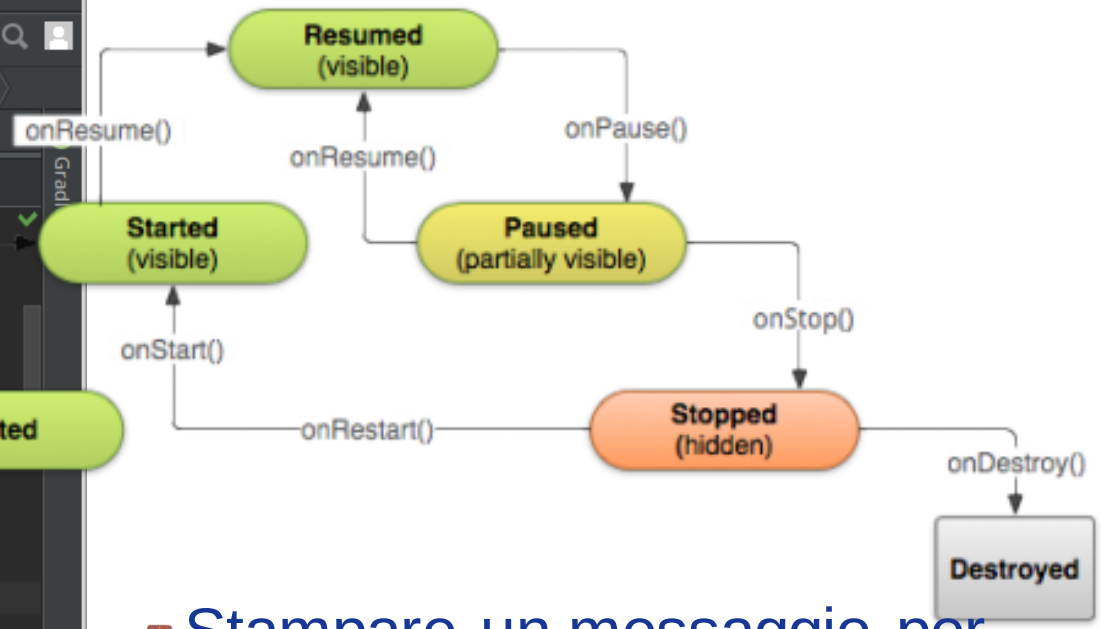
Motorola Nexus 6 Android 7.0, API 24 | it.insubria.pdm.a01_helloworld (18518)

logcat Monitors Verbose Regex Show only selected application

```
03-04 13:11:23.897 18518-18518/? W/System: ClassLoader referenced unknown path: /da
03-04 13:11:23.933 18518-18518/? W/art: Before Android 4.1, method android.graphics
03-04 13:11:23.939 18518-18518/? V/MainActivity (Ignazio): onCreate
03-04 13:11:24.017 18518-18518/? V/MainActivity (Ignazio): onStart
03-04 13:11:24.020 18518-18518/? V/MainActivity (Ignazio): onResume
03-04 13:11:24.113 18518-18535/? I/Adreno: QUALCOMM build                    : 74df4
```

4: Run TODO 6: Android Monitor 0: Messages Terminal Event Log Gradle Console

Gradle build finished in 10s 575ms (29 minutes ago) 44:29 LF UTF-8 Context: <no context>

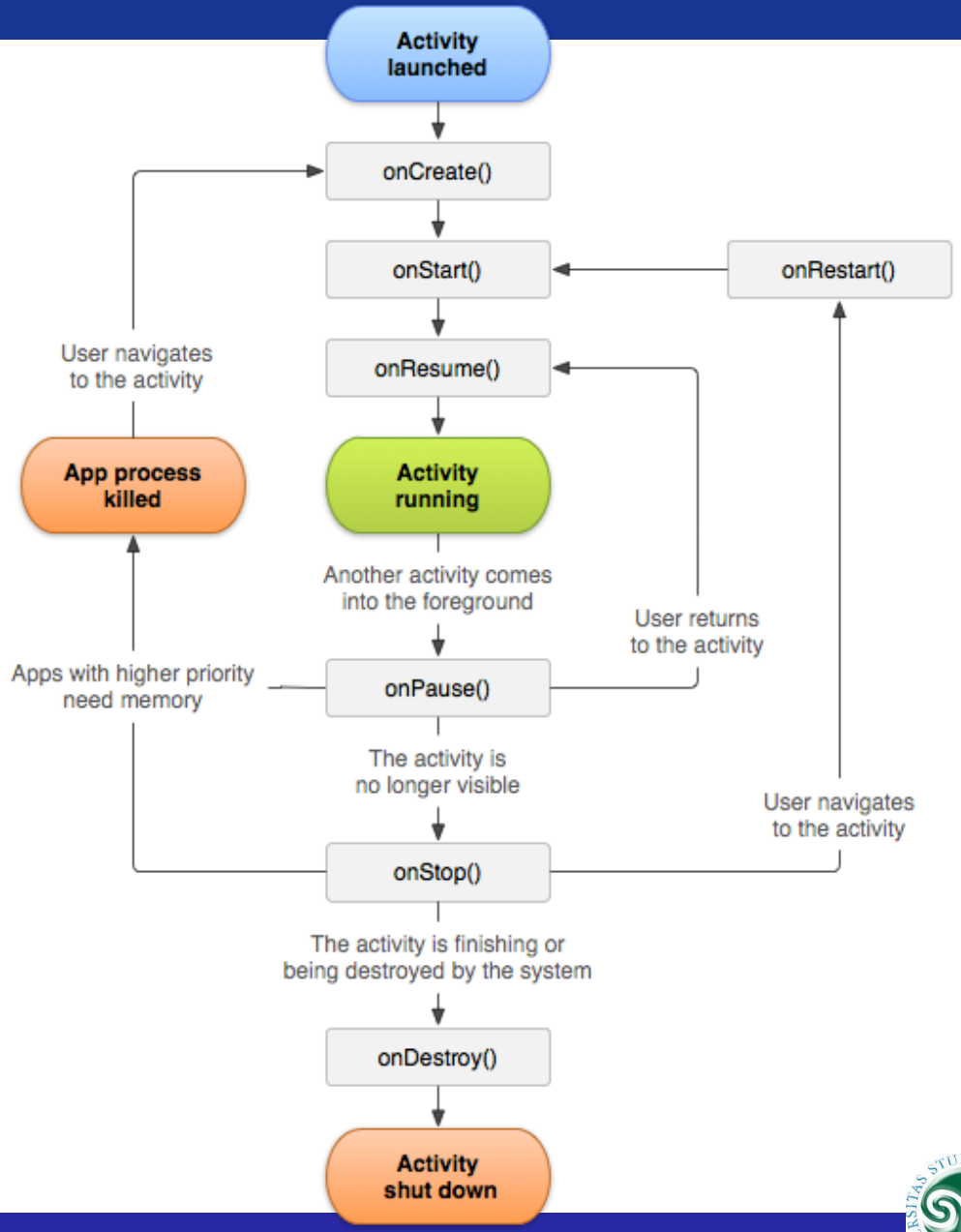


- Stampare un messaggio per ognuna delle callback relative ad una Activity che va in **esecuzione** e a **riposo**
- Provare a far stampare tutti i messaggi di log

Activity lifecycle

```
class MainActivity : AppCompatActivity() {
  val TAG = "MainActivity"
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    Log.v(TAG, "onCreate")
  }

  override fun onStart() {
    super.onStart()
    Log.v(TAG, "onStart");
  }

  override fun onResume() {
    super.onResume()
    Log.v(TAG, "onResume")
  }

}
```

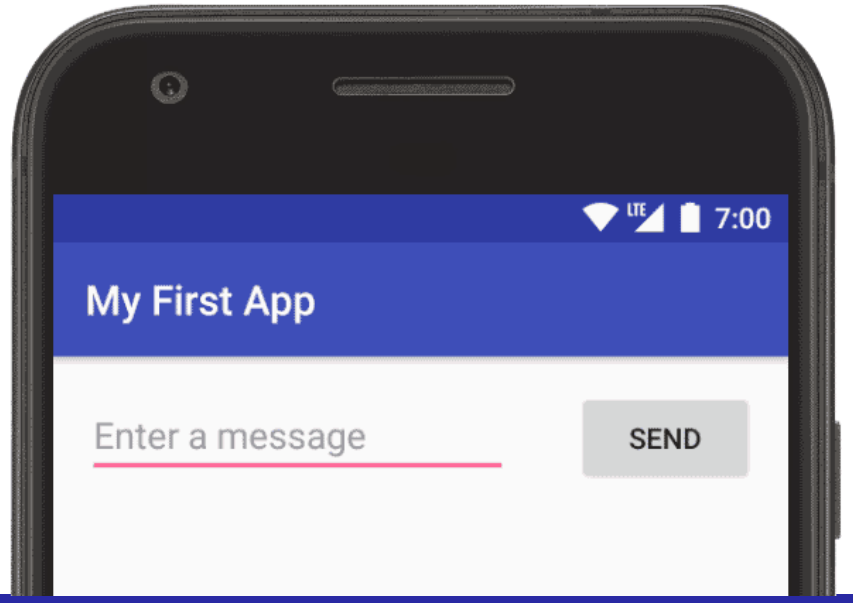
```
override fun onPause() {
   super.onPause()
   Log.v(TAG,"onPause")
}

override fun onStop() {
   super.onStop()
   Log.v(TAG, "onStop")
}

override fun onDestroy() {
   super.onDestroy()
   Log.v(TAG, "onDestroy")
}
```



Build a Simple User Interface

This laboratory exercise teaches you to

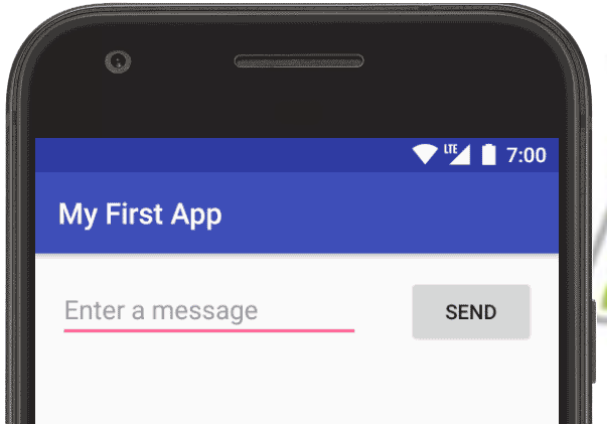
- Create an Android Project
- Open the Layout Editor
- Add a text box
- Add a button
- Change the UI strings
- Make the text box size flexible



Create an Android Project

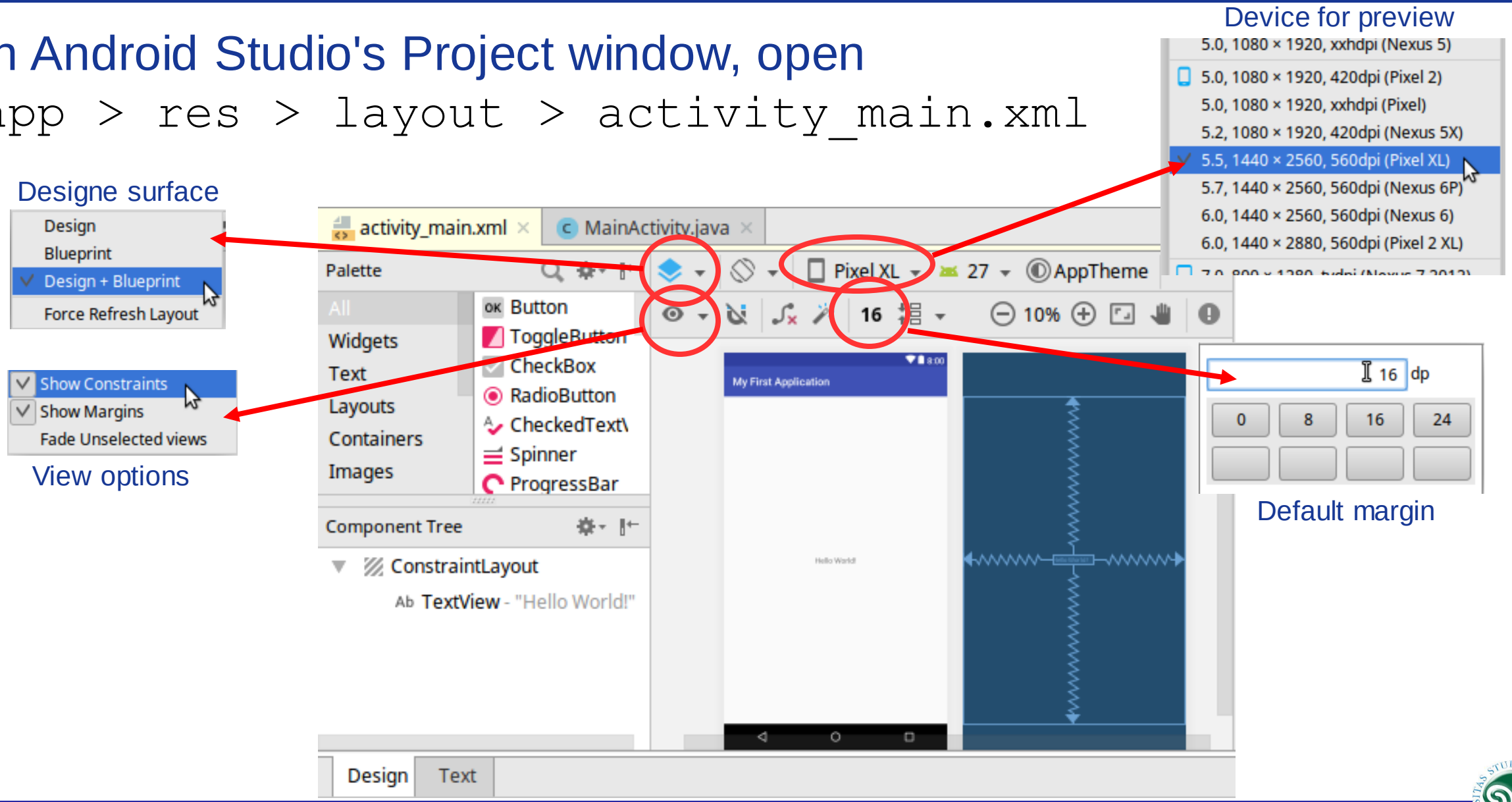
- Click Start a new Android Studio project or select File > New Project.
- In the **Create New Project** window, enter the following values:
  - Application Name: "My First App"
  - Company Domain: "pdm.unindubria.it"
- Click Next and Next.
- In the Add an Activity to Mobile screen, select Empty Activity and click Next.
- In the Configure Activity screen, keep default and click Finish.

Open the Layout Editor



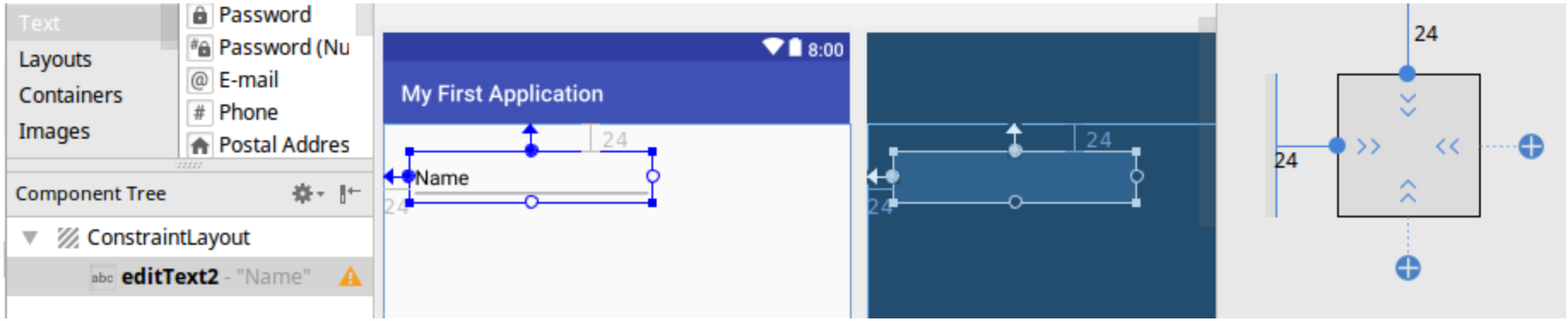
Open the Layout Editor

- In Android Studio's Project window, open
  `app > res > layout > activity_main.xml`



Add a text box

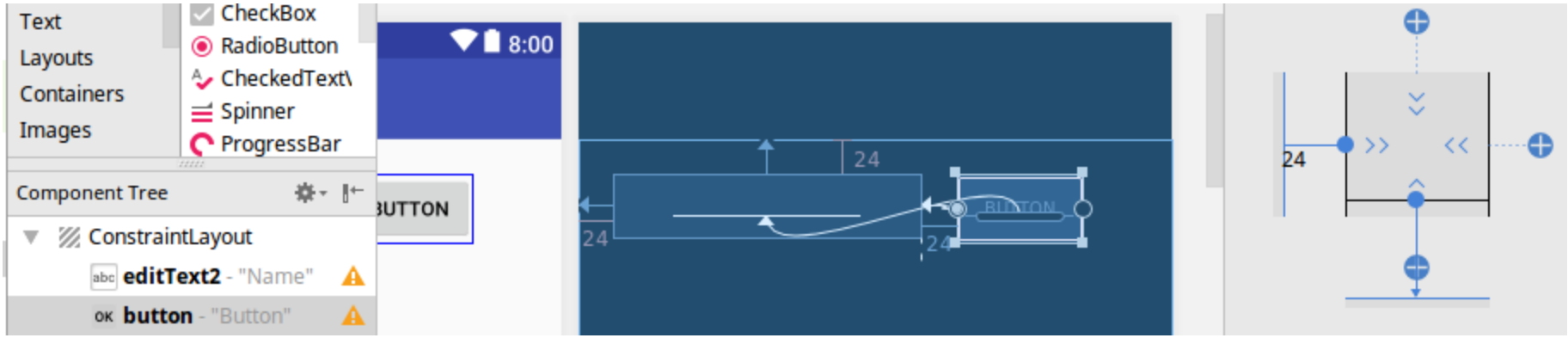
- First, remove what's already in the layout.
- In the Palette, click Text to show the available text controls.
- Drag Plain Text into the design editor and drop it near the top of the layout. This is an EditText widget that accepts plain text input.
- Click the view. You can now see the constraint anchors on each side (circles).
- Click-and-hold the anchor on the top side and left side



Add a button

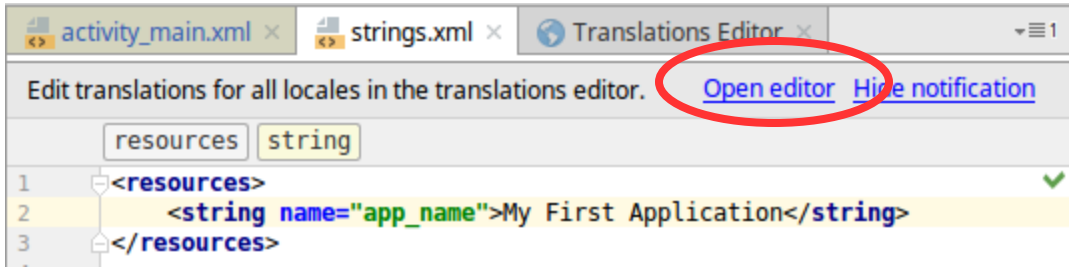
In the Palette, click Widgets or Buttons.

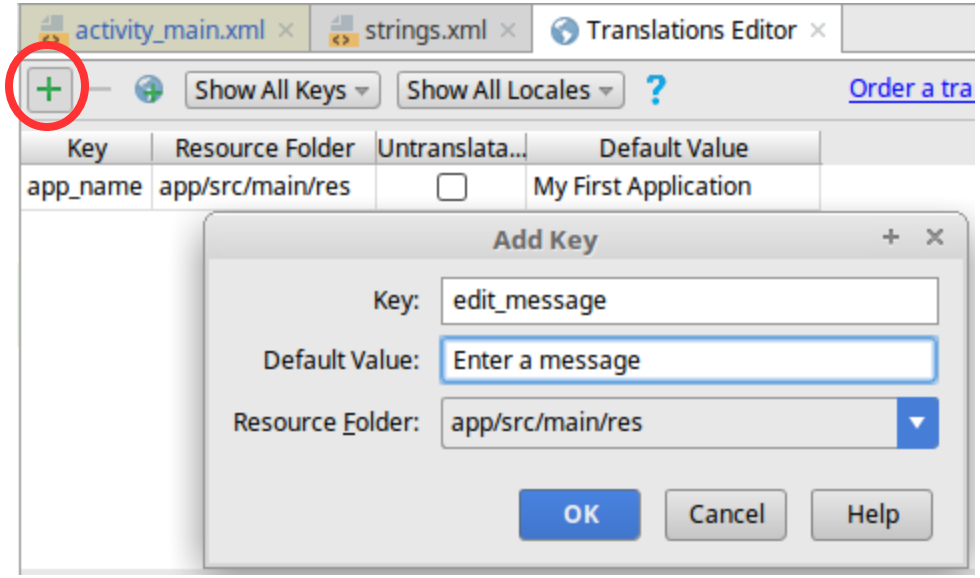
- Drag Button into the design editor and drop it near the right side.
- To constrain the views in a horizontal alignment, you need to create a constraint between the text baselines. So click the button, and then click Edit Baseline .
- The baseline anchor appears inside the button. Click-and-hold on this anchor and then drag it to the baseline anchor in the text box.



Change the UI strings

- Open the Project window and then open `app > res > values > strings.xml`.
- Click Open editor at the top of the editor window.
- Click Add Key to create a new string as the "hint text" for the text box.
- Add another key named "button_send" with a value of "Send"

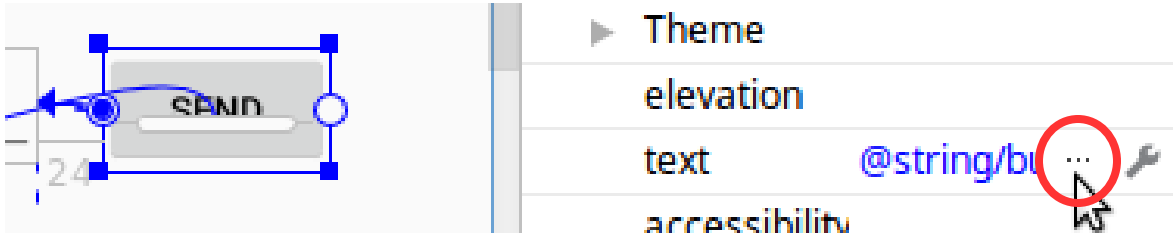


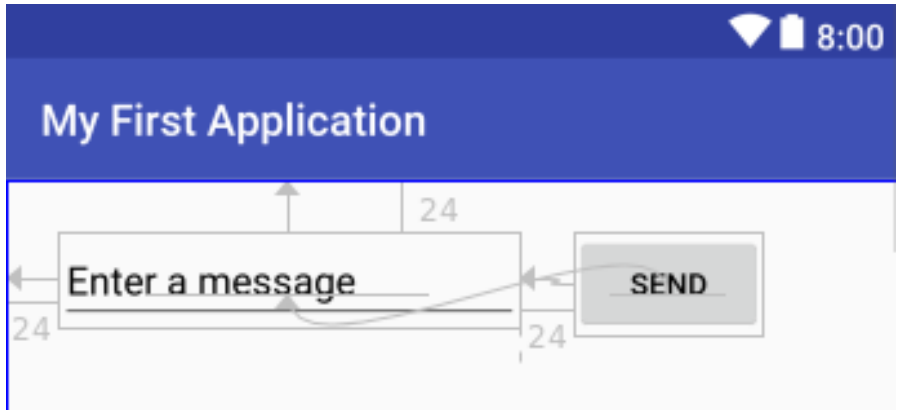



```
<resources>
    <string name="app_name">My First Application</string>
    <string name="edit_message">Enter a message</string>
    <string name="button_send">Send</string>
</resources>
```

Change the UI strings

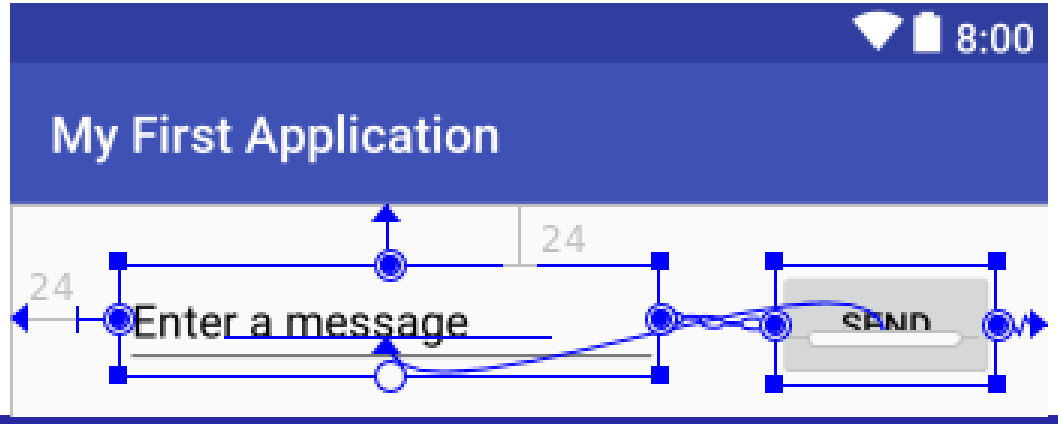
- Set these strings for each view.
- Click the text box in the layout and click Attributes on the right sidebar.
- Locate the text property (currently set to "Name")
- Click Pick a Resource to the right of the text box. In the dialog that appears, double-click on edit_message from the list.
- Now click the button in the layout, and then select button_send.





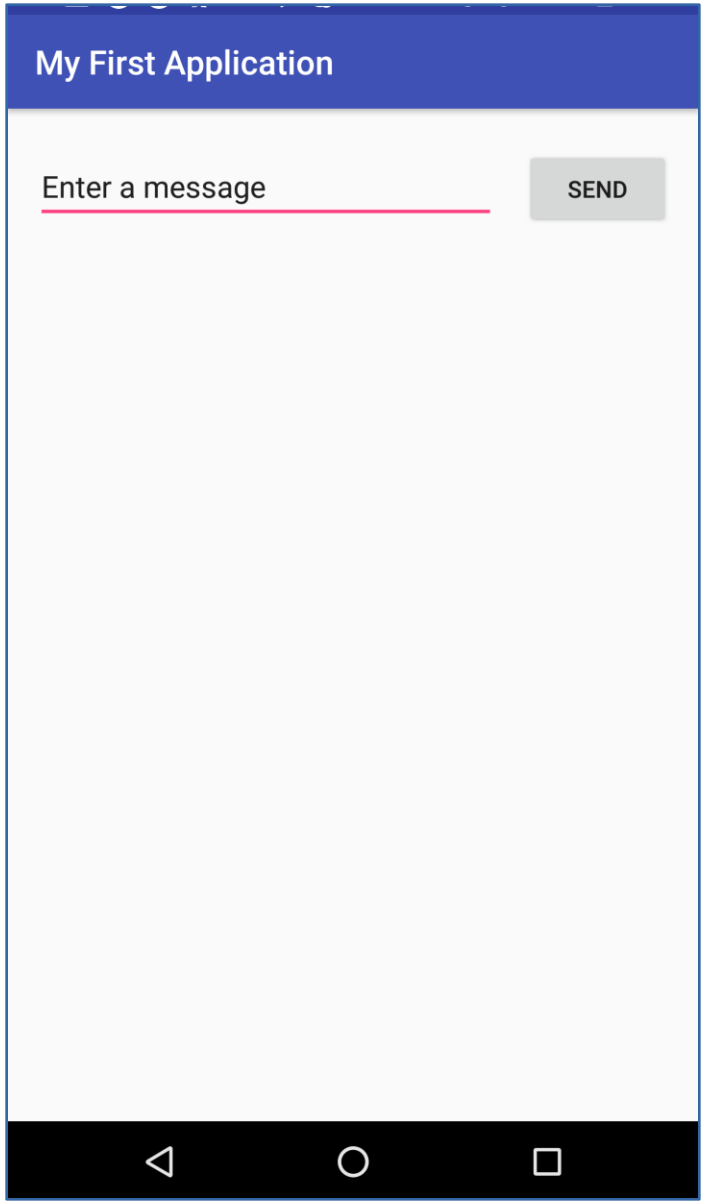
Make the text box size flexible

- To create a layout that's responsive to different screen sizes, you'll now make the text box stretch to fill all remaining horizontal space
- Select both views (click one, hold Shift, and click the other), and then right-click either view and select Chain > Create Horizontal Chain.
- A chain is a bidirectional constraint between two or more views that allows you to lay out the chained views in unison.



Run the app

- If your app is already installed on the device, simply click Apply Changes in the toolbar to update the app with the new layout.
- Or click Run to install and run the app.



Start Another Activity

Respond to the send button

- Add some code to MainActivity that starts a new activity to display the message when the user taps Send.
- Add a sendMessage() method to the MainActivity class that's called by the button
- Now return to the activity_main.xml file to call this method from the button:
  - Click to select the button in the Layout Editor.
  - In the Attributes window, locate the onClick property and select sendMessage [MainActivity] from the drop-down list.

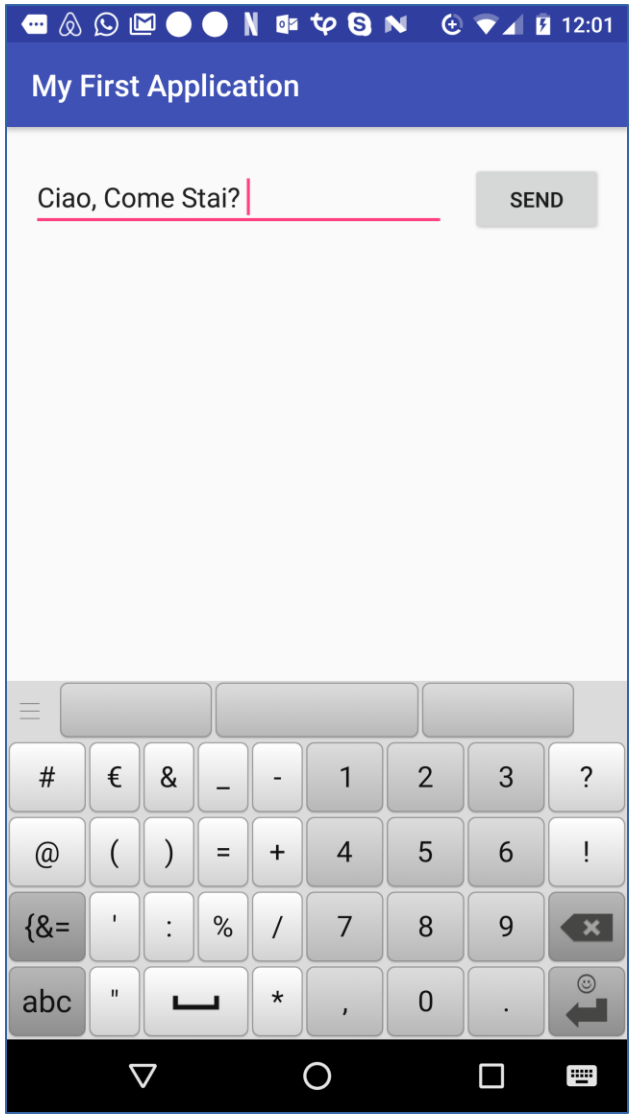
```
class MainActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?)
{
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)
}

// Called when the user taps the Send button
fun sendMessage(v: View) {

}
```

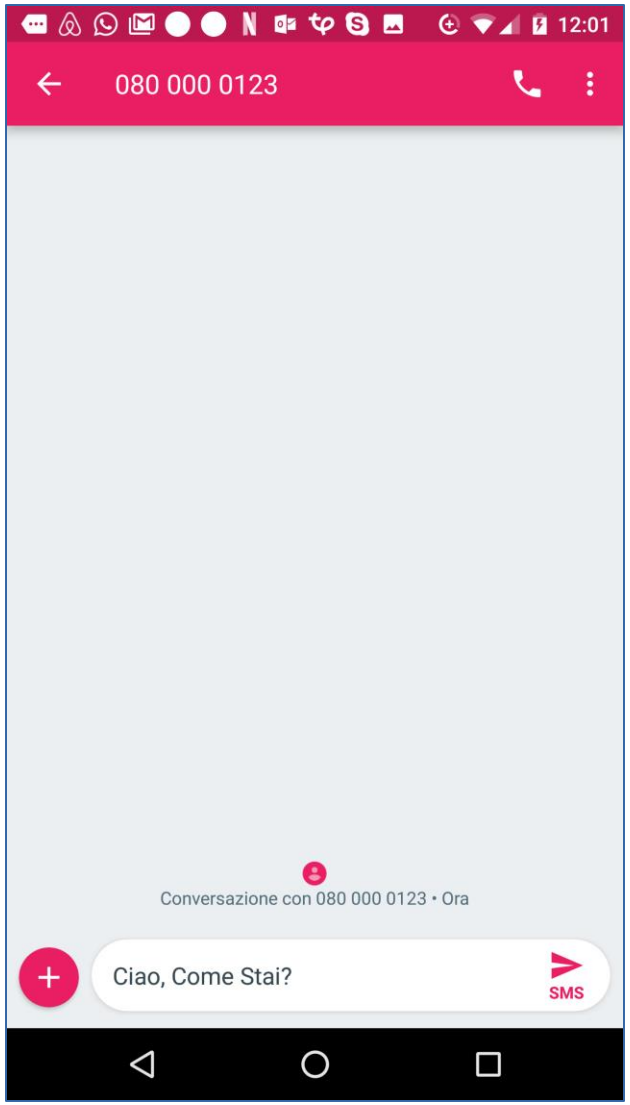
Build an Intent in Kotlin

- An Intent is an object that provides runtime binding between separate components, such as two activities.
- The Intent represents an app's "intent to do something."
- You can use intents for a wide variety of tasks, but here, your intent starts another activity.

```
fun sendMessage(v: View) {
    val message = textViewMessage.text.toString()

    val uri: Uri = Uri.parse("smsto:0800000123")
    val it = Intent(Intent.ACTION_SENDTO, uri)
    it.putExtra("sms_body", message)
    startActivity(it)
}
```

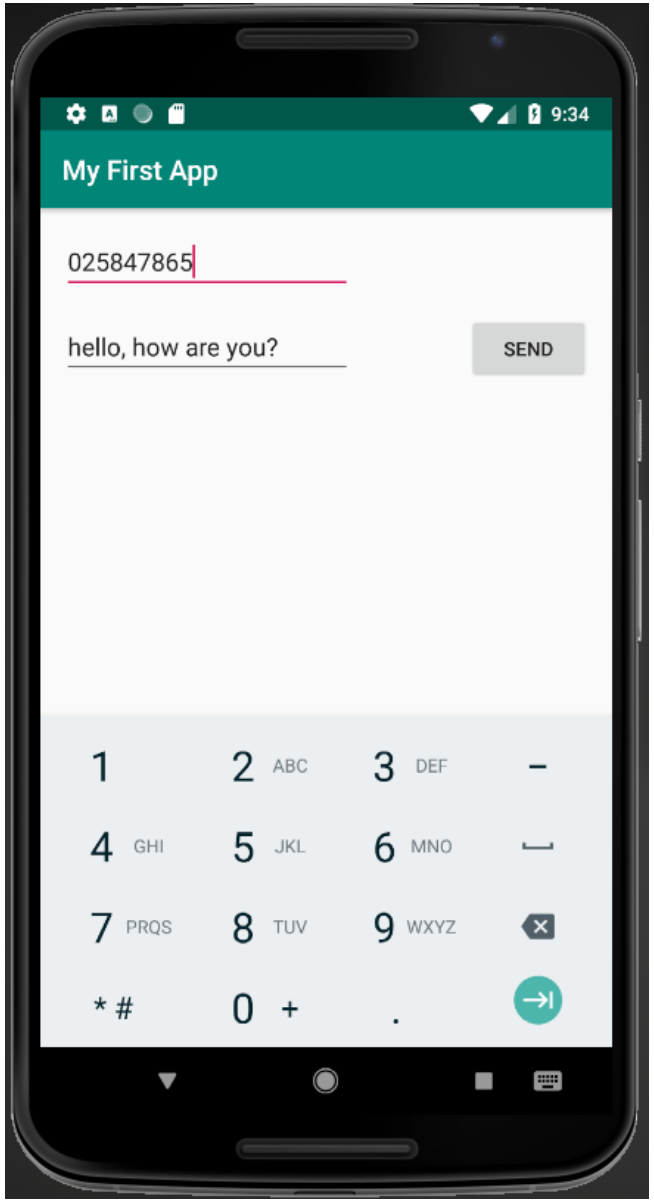
Result

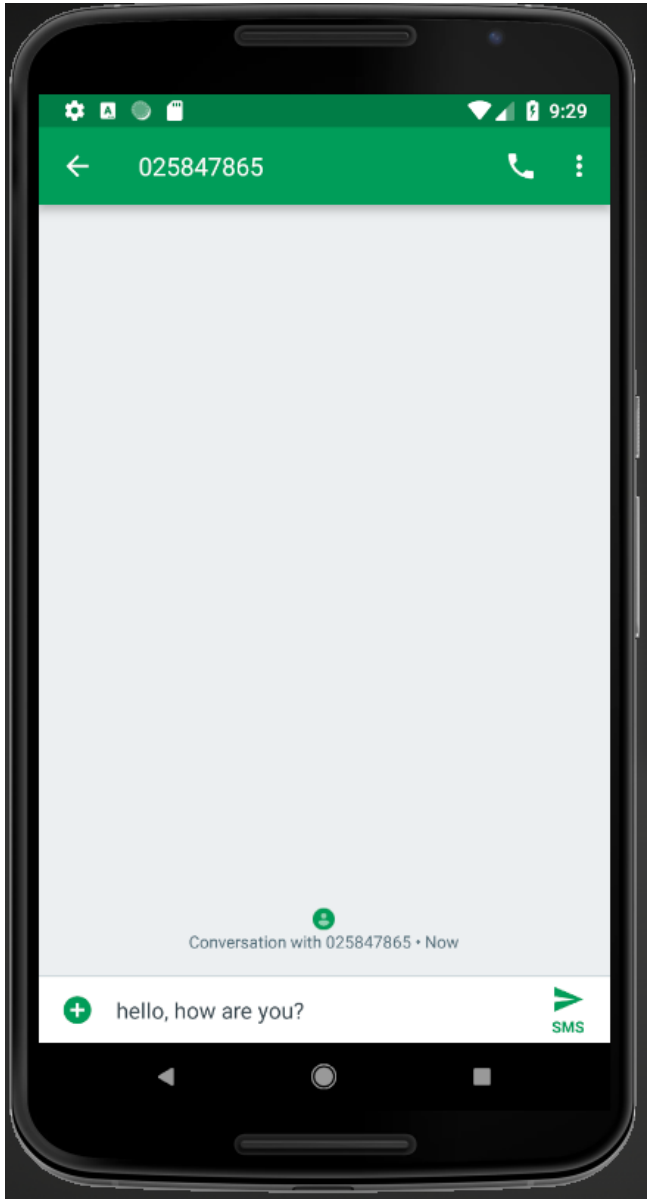

12:01
My First Application
Ciao, Come Stai?
SEND


12:01
080 000 0123
Conversazione con 080 000 0123 • Ora
Ciao, Come Stai?
SMS

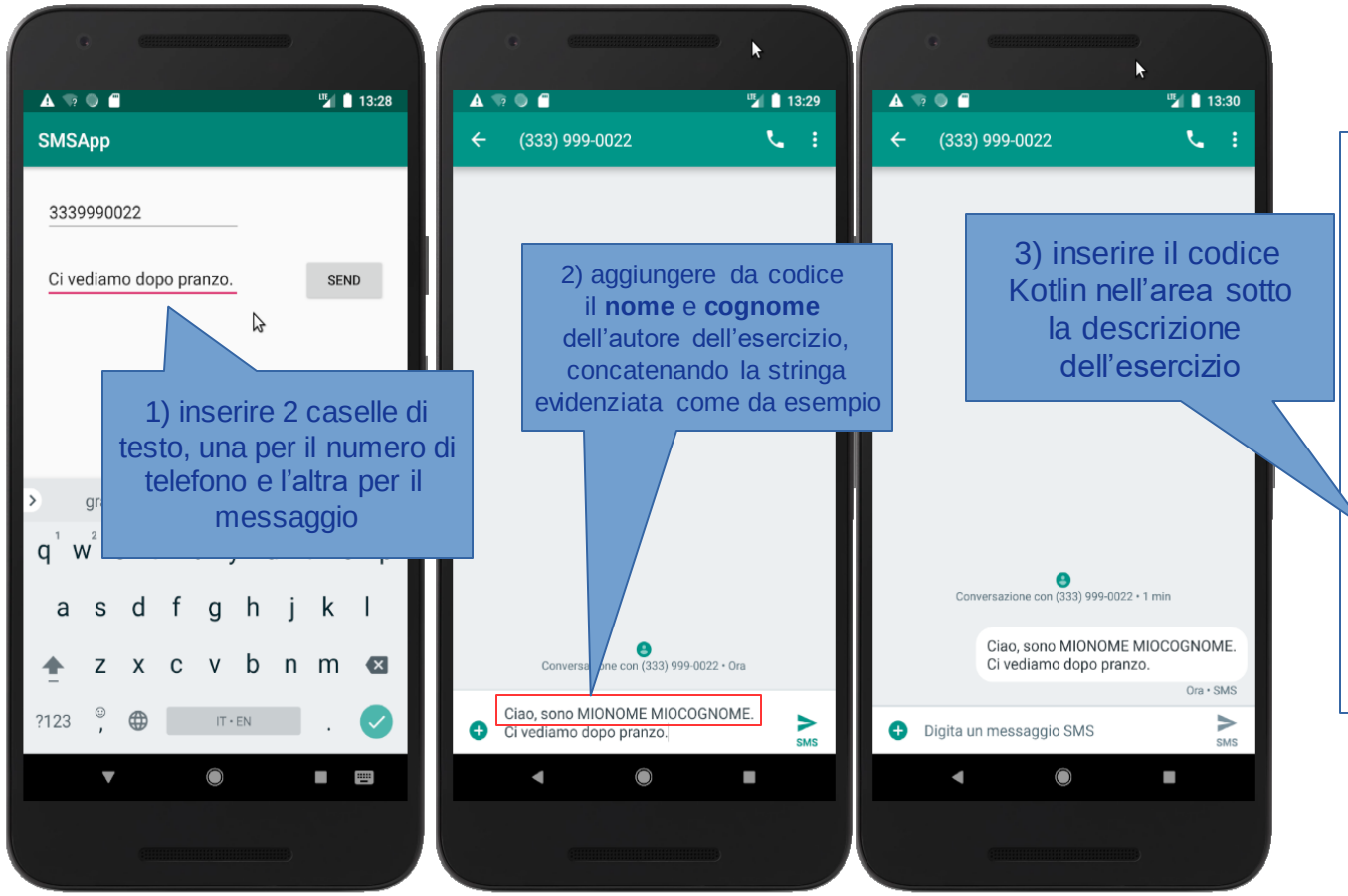
Add phone number EditText

Result


9:34
My First App
025847865
hello, how are you?
SEND


9:29
025847865
Conversation with 025847865 • Now
hello, how are you?
SMS

Assignment



1) inserire 2 caselle di testo, una per il numero di telefono e l'altra per il messaggio

2) aggiungere da codice il **nome** e **cognome** dell'autore dell'esercizio, concatenando la stringa evidenziata come da esempio

3) inserire il codice Kotlin nell'area sotto la descrizione dell'esercizio

```java
public class MainActivity extends AppCompatActivity {

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

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        // Do something in response to button
    }
}
```