

## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

### OBJECTIVES

- Understanding Activity
- Understanding Activity Life Cycle
- Understanding Android Layouts
- Practice Activities

### OBJECTIVE 1: Understanding Activity Life Cycle

- An Android activity is one screen of the Android app's user interface. In that way an Android activity is very similar to windows in a desktop application. An Android app may contain one or more activities, meaning one or more screens. The Android app starts by showing the main activity, and from there the app may make it possible to open additional activities.
- The mobile-app experience differs from its desktop counterpart in that a user's interaction with the app doesn't always begin in the same place. Instead, the user journey often begins non-deterministically. For instance, if you open an email app from your home screen, you might see a list of emails. By contrast, if you are using a social media app that then launches your email app, you might go directly to the email app's screen for composing an email
- The [Activity](#) class is designed to facilitate this paradigm. When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole. In this way, the activity serves as the entry point for an app's interaction with the user. You implement an activity as a subclass of the [Activity](#) class.

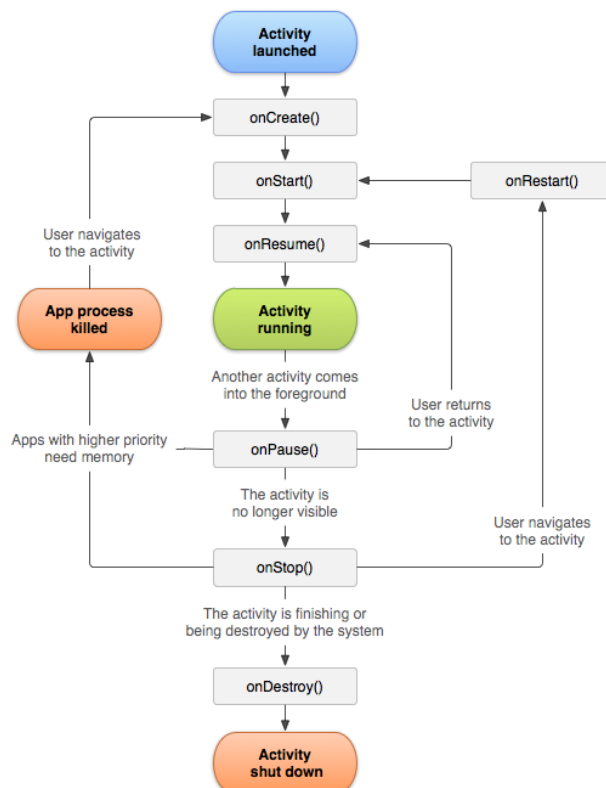
### Declare activities

To declare your activity, open your manifest file and add an <activity> element as a child of the <application> element. For example

```
<manifest ... >
  <application ... >
    <activity android:name=".ExampleActivity" />
    ...
  </application ... >
  ...
</manifest >
```

The only required attribute for this element is **android:name**, which specifies the class name of the activity. You can also add attributes that define activity characteristics such as label, icon, or UI theme.

## OBJECTIVE 2: Understanding Activity Life Cycle



Over the course of its lifetime, an activity goes through a number of states. You use a series of callbacks to handle transitions between states. The following sections introduce these callbacks.

## onCreate()

You must implement this callback, which fires when the system creates your activity. Your implementation should initialize the essential components of your activity: For example, your app should create views and bind data to lists here. Most importantly, this is where you must call `setContentView()` to define the layout for the activity's user interface.

## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

When `onCreate()` finishes, the next callback is always `onStart()`.

### **`onStart()`**

As `onCreate()` exits, the activity enters the Started state, and the activity becomes visible to the user. This callback contains what amounts to the activity's final preparations for coming to the foreground and becoming interactive.

### **`onResume()`**

The system invokes this callback just before the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, and captures all user input. Most of an app's core functionality is implemented in the `onResume()` method.

The `onPause()` callback always follows `onResume()`.

### **`onPause()`**

The system calls `onPause()` when the activity loses focus and enters a Paused state. This state occurs when, for example, the user taps the Back or Recents button. When the system calls `onPause()` for your activity, it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.

An activity in the Paused state may continue to update the UI if the user is expecting the UI to update. Examples of such an activity include one showing a navigation map screen or a media player playing. Even if such activities lose focus, the user expects their UI to continue updating.

You should not use `onPause()` to save application or user data, make network calls, or execute database transactions. For information about saving data, see [Saving and restoring activity state](#).

Once `onPause()` finishes executing, the next callback is either `onStop()` or `onResume()`, depending on what happens after the activity enters the Paused state.

### **`onStop()`**

The system calls `onStop()` when the activity is no longer visible to the user. This may happen because the activity is being destroyed, a new activity is starting, or an existing activity is entering a Resumed state and is covering the stopped activity. In all of these cases, the stopped activity is no longer visible at all.

The next callback that the system calls is either `onRestart()`, if the activity is coming back to interact with the user, or by `onDestroy()` if this activity is completely terminating.

### **`onRestart()`**

The system invokes this callback when an activity in the Stopped state is about to restart. `onRestart()` restores the state of the activity from the time that it was stopped.

This callback is always followed by `onStart()`.

## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

### **onDestroy()**

The system invokes this callback before an activity is destroyed.

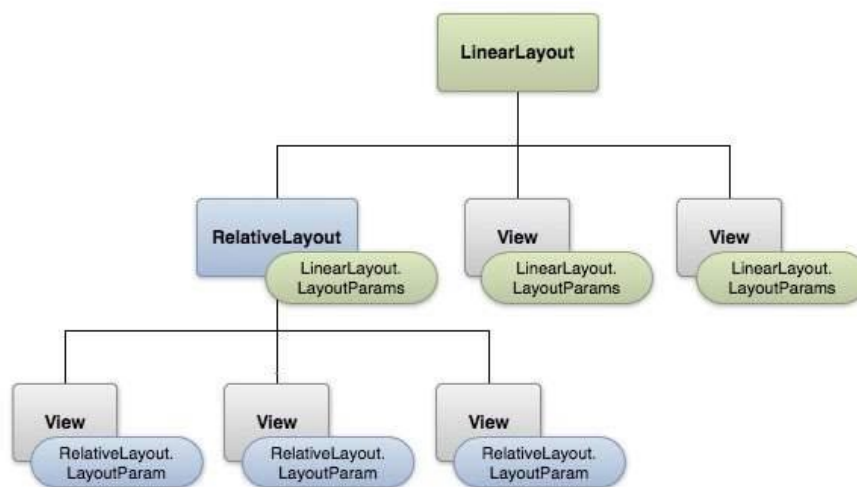
This callback is the final one that the activity receives. `onDestroy()` is usually implemented to ensure that all of an activity's resources are released when the activity, or the process containing it, is destroyed.

This section provides only an introduction to this topic. For a more detailed treatment of the activity lifecycle and its callbacks, see [The Activity Lifecycle](#).

## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

### OBJECTIVE 2: Understanding Android Layouts

- The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.
- The **ViewGroup** is a subclass of View and provides invisible container that hold other Views or other ViewGroups and define their layout properties.
- At third level we have different layouts which are subclasses of **ViewGroup** class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file main\_layout.xml which is located in the **res/layout** folder of your project



Layout Attributes

- + CoordinatorLayout
- + RelativeLayout
- + LinearLayout
- + AbsoluteLayout

Layout Params

- layout\_behavior
- layout\_gravity
- layout\_weight
- layout\_above
- layout\_below
- layout\_alignLeft/Top/Right/Bottom
- layout\_alignParentLeft/etc
- layout\_toLeftOf/etc
- layout\_alignBaseline
- layout\_centerInParent

LinearLayout

orientation="horizontal"

orientation="vertical"

FrameLayout

layout\_gravity="center"

layout\_gravity="bottom|end"

layout\_margin="16dp"

Am I a ViewGroup?

- X Button
- X TextView
- X Checkbox
- ✓ Toolbar
- ✓ FrameLayout
- ✓ LinearLayout

These can contain other views aka "children"

Layout Params

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage"
/>
```








#BuildBetterApps

Protip: don't hardcode your layouts. Use @dimen

## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

### Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel

Sr.No	Layout & Description
1	Linear Layout <a href="#"></a> LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout <a href="#"></a> RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout <a href="#"></a> TableLayout is a view that groups views into rows and columns.
4	Absolute Layout <a href="#"></a> AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout <a href="#"></a> The FrameLayout is a placeholder on screen that you can use to display a single view.
6	List View <a href="#"></a> ListView is a view group that displays a list of scrollable items.
7	Grid View <a href="#"></a> GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

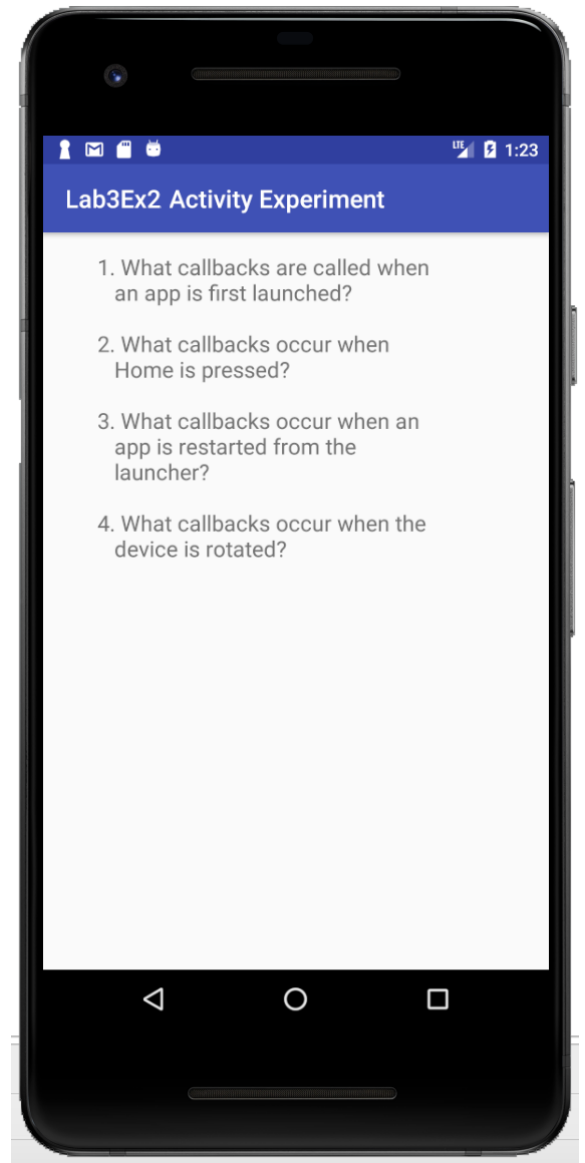
### PRACTICE ACTIVITIES:

**Activity 1** : Create an app that explores the life-cycle of an activity.

- The main objective of this app is to experience callback methods first hand.

Write definition of all these methods and observe the results.

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onRestart()
- onDestroy()

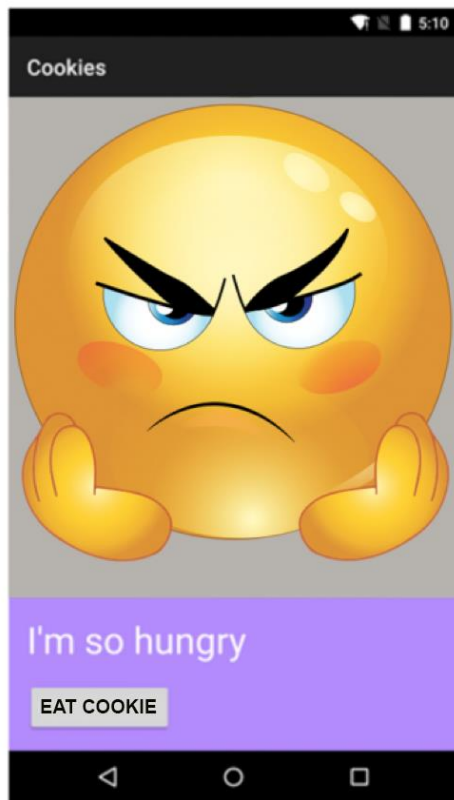


## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

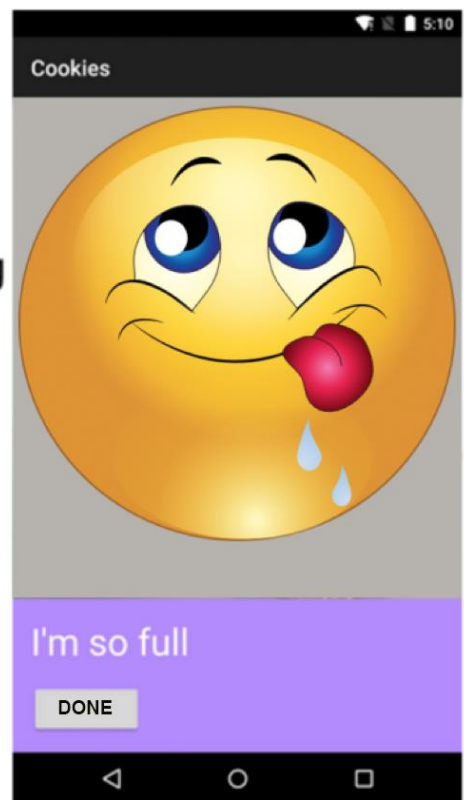
**Activity 2 :** Develop an app to describe the mood.

- Add following Components

Android View	Value	Event
ImageView	Mood Image	<i>None</i>
TextView	"I'm So Hungry" or "I'm so Full"	<i>None</i>
Button	"Eat Cookie"	onEatingCookie()



After pressing  
the button  
becomes

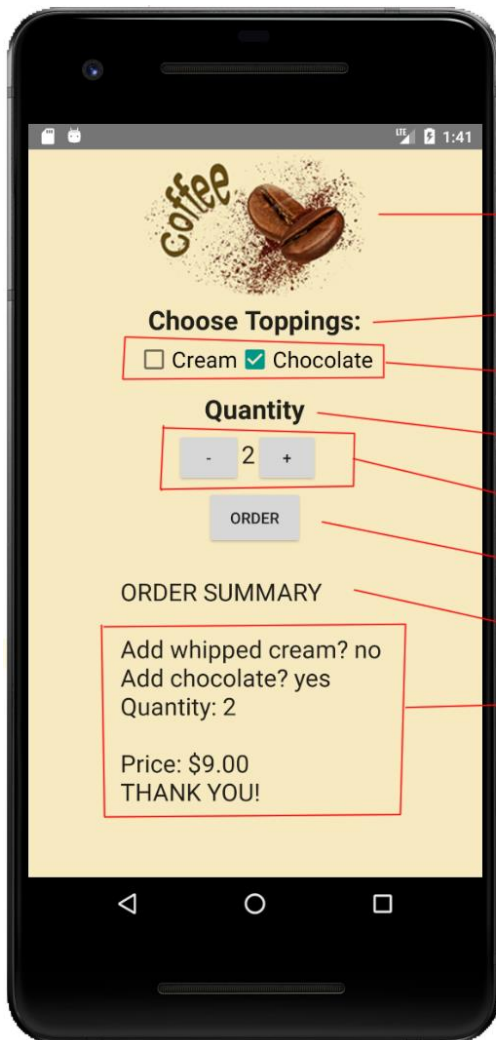




## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

**Activity 3 :** Build the Coffee Ordering app shown below.

- Assume a single coffee costs \$4.00. Charge an additional \$1.00 for chocolate and \$.50 for whipped cream, per cup.
- Background Color : #f7eac1



Root View: LinearView

ImageView

TextView

LinearView containing two CheckBoxes

TextView

LinearView containing two buttons and a TextView

Button

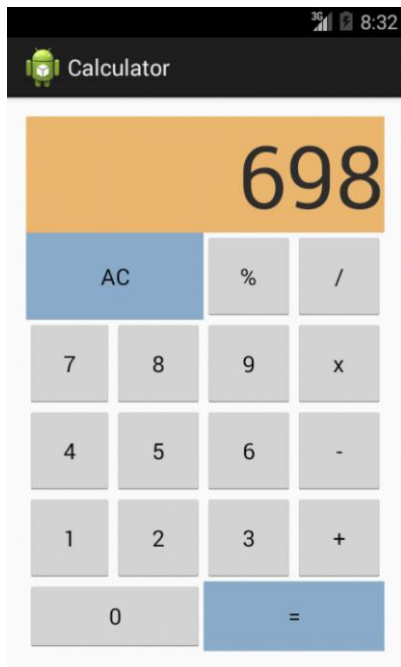
TextView

TextView

## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

### Activity 4: Calculator

Create the simple calculator shown below. This app should use a `TableLayout`.



## MOBILE APPLICATION DEVELOPMENT (MAD) : LAB 3

### Activity 5: Hangman Game (Assignment # 1)

#### Hangman

Make a basic **Hangman** game that displays an image of a gallows and a hanging man, along with a word that the player is trying to guess. The word is chosen randomly from a provided dictionary. At all times the game displays a "**clue**" of the letters the player has guessed correctly; for example, if the word is "**apples**" and the player has guessed **e**, **k**, **p**, and **t**, the clue would be "**?pp?e?**". The user can type single-letter guesses into an **EditText**. (The **EditText** allows the user to type multi-letter strings and non-letters; a robust game would handle such attempts gracefully, as well as other errors like trying to guess the same letter twice, etc.) You can display a message such as a **Toast** when the user guesses the word correctly or runs out of guesses and ends the game.

